***GIT-HUB LINK -*** [***https://github.com/Utkarsh9991/Testing.git***](https://github.com/Utkarsh9991/Testing.git)

***WRITEUP FOR MAKEMYTRIP WEB ELEMENTS***

***1. Accessing the Website and Taking Screenshots:***

***WebDriver Initialization: Starts the Chrome browser and maximizes the window for optimal visibility.***

***Navigation to MakeMyTrip: Directs the browser to the MakeMyTrip website.***

***Initial Screenshot Capture: Takes a screenshot of the initial page and saves it for reference or documentation purposes.***

***2. Handling Pop-ups:***

Closing the Initial Pop-up:

Frame Switching: Uses WebDriverWait to locate and switch to a specific frame responsible for a notification.

Dismissal of the Pop-up: Identifies and clicks on the close button to eliminate the initial pop-up that might hinder the user experience.

Screenshot After Pop-up Closure: Captures another screenshot after the initial pop-up is closed for record-keeping.

Managing Login-related Pop-up:

Return to Main Frame: Shifts back to the main frame context.

Closing the Login Pop-up: Locates the close button of a potential login-related pop-up and dismisses it.

Screenshot After Handling Login Pop-up: Captures another screenshot to document the action taken.

***3. Interacting with Input Fields:***

Entering Location Details:

Utilizes XPaths to interact with the 'fromCity' and 'toCity' input fields, filling them with 'Delhi' and 'Mumbai', respectively.

Screenshot After Location Entry: Captures a screenshot after providing the location details for reference.

***4. Selecting Location Suggestions:***

Choosing Suggested Location:

Interacts with the suggested location list associated with 'fromCity', possibly to refine the search or confirm the input.

***5. Date Selection:***

Picking a Specific Date:

Interacts with the date picker interface, selecting a particular date crucial for the trip planning.

Screenshot After Date Selection: Captures another screenshot after confirming the date for record-keeping.

***6. Performing a Search:***

Initiating Search Action:

Triggers the search process by clicking the 'Search' button.

***7. Handling a Page Refresh:***

Preparation for Page Refresh Handling:

Implements a try-catch block and a longer wait duration to anticipate a page refresh or any related pop-ups.

Handling Post-Refresh Pop-up: If a specific pop-up emerges, the script identifies and dismisses it.

Screenshot After Handling Potential Page Refresh: Captures a screenshot to document the action taken.

***8. Script Completion:***

Finalization of Automated Steps:

After executing all the interactions and potential pop-up handling, the script concludes its sequence, marking the completion of the automated process.

***2. AUTO-IT WITH SELENIUM***

**This code utilizes Selenium, a popular automation framework for web applications, to interact with a web page and also uses the auto-it to interact with the applications. It automates browser actions in the Chrome browser using Java. The code involves:**

**WebDriver Initialization**: A ChromeDriver instance is created to manipulate the browser.

**Maximizing Window:** The browser window is maximized for a better view of the web elements.

**Implicit Wait:** An implicit wait is set to 10 seconds to allow the driver to wait for elements to be located before throwing an exception.

**Navigating to a Web Page:** The driver opens the "remove.bg" website.

**Taking a Screenshot:** A screenshot of the webpage is taken and saved to a specific location on the local system.

**Locating and Interacting with Web Elements:**

A web element is located using an XPath selector.

Click action is performed on the located element.

**Thread Sleep:** The code pauses execution for 2000 milliseconds (2 seconds).

**Executing External Process:** It triggers an external process (AutoIt script) located at "C:\Users\utkar\OneDrive\Desktop\autoitfolder\AutoitScript.exe".

This AutoIt script could be called from the Java code, using **Runtime.getRuntime().exec()**

***3. JDBC with Selenium***

Trying to combine web automation using Selenium with database interaction through JDBC (Java Database Connectivity).

**1. Set Up WebDriver**: Selenium to automate a web browser. In this case, it's Chrome. You're maximizing the window and navigating to Rediff Mail ([**https://mail.rediff.com/cgi-bin/login.cgi**](https://mail.rediff.com/cgi-bin/login.cgi)**).**

**2. Retrieve Page Information** Using Selenium, you're getting the title of the web page and the current URL.

**3. Database Connection** You're establishing a connection to a MySQL database named "seleniumtest" running on localhost, using the root username and password.

**4. Data Insertion** After connecting to the database, you're preparing and executing SQL queries to insert the title and URL data from the Rediff Mail page into a table named "webtest".

**5. Navigate to Another Page** You're then navigating to GitHub (https://github.com/) using Selenium.

Retrieve New Page Information: Again, you're retrieving the title and URL of the GitHub page.

**6. Insert New Data:** Similar to before, you're preparing and executing SQL queries to insert the new title and URL data from the GitHub page into the "webtest" table.

**7. Close Database Connection:** Finally, you're closing the connection to the database