NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, APPROVED BY AICTE & GOVT.OF KARNATAKA)

Yelahanka, Bangalore - 560 064

COURSE PROJECT REPORT

of

TOOL BASED SOFTWARE TESTING (22CSA583)

on

"BIGBASKET AUTOMATION USING SELENIUM AND JAVA"

Submitted in the Partial fulfillment of the requirements of Semester-5 of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by:

PHILIP JOHN
1NT22CS130

SANJAY PATIL M
1NT22CS180

SUDEEP S A
1NT22CS197

VISHNU V
1NT22CS214

Under the Guidance of

Dr. Uma R

Associate Prof., Dept. of CSE

Department of Computer Science and Engineering
(Accredited by NBA Tier-1)

2024 – 2025

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

Yelahanka, Bangalore - 560 064
Affiliated to Visveswaraya Technological University, Belgaum.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Course Project Work titled "**BIGBASKET AUTOMATION USING SELENIUM AND JAVA**", carried out by **Philip John** bearing **USN: 1NT22CS130, Sanjay Patil M** bearing **USN: 1NT22CS180, Sudeep S A** bearing **USN: 1NT22CS197 and Vishnu V** bearing **USN: 1NT22CS214** bonafide students of **Nitte Meenakshi Institute of Technology** in partial fulfilment of Semester-5 of Bachelor of Engineering Degree in Computer Science & Engineering under Visvesvaraya Technological University, Belagavi during the year 2024-2025. It is certified that all corrections/ suggestions indicated for Internal Assesment have been incorporated in the Report deposited in the Departmental Library. The Report has been approved as it satisfies the Academic requirements in respect of the Course Project Work prescribed for the said Degree.

Signature of Guide

……..
Dr Uma R
Associate Professor,
Dept. of CSE, NMIT.

DECLARATION

We hereby declare that

(i) The project work is our original work

(ii) This Project work has not been submitted for the award of any degree or examination at any other university/College/Institute.

(iii) This Project Work does not contain other persons' data, pictures, or other information, unless specifically acknowledged as being sourced from other persons.

(iv) This Project Work does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

a) their words have been re-written but the general information attributed to them has been referenced;

b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

(v) This Project Work does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

NAME	USN	Signature
Philip John	1NT22CS130	
Sanjay Patil M	1NT22CS180	
Sudeep S A	1NT22CS197	
Vishnu V	1NT22CS214	

Date:

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned our effort with success. I express my sincere gratitude to our Principal **Dr. H. C. Nagaraj**, Nitte Meenakshi Institute of Technology for providing facilities.

We wish to thank our HoD, **Dr. S Meenakshi Sundaram** for the excellent environment created to further educational growth in our college. We also thank him for the invaluable guidance provided which has helped in the creation of a better project.

We are very grateful to our guide **Dr Uma R**, Associate Professor, Dept. of CSE for her valuable inputs in making us understanding the concepts and for constatntly supporting us during the course of this project work.

We also thank all our friends, teaching and non-teaching staff at NMIT, Bangalore, for all the direct and indirect help provided in the completion of the project.

NAME	USN	Signature
Philip John	1NT22CS130	
Sanjay Patil M	1NT22CS180	
Sudeep S A	1NT22CS197	
Vishnu V	1NT22CS214	

Date:

Abstract

This project implements an automated testing framework for the BigBasket e-commerce platform using Selenium WebDriver with Java. The focus is on verifying the "Add to Cart" functionality, a critical feature of any online shopping platform. The automation script simulates the complete user journey, starting from launching the BigBasket website, handling login and manual OTP-based authentication, navigating through the product catalog, and finally adding selected items to the shopping cart.

The framework employs advanced Selenium techniques, including dynamic waits for asynchronous web elements, JavaScript execution for interacting with complex UI components like SVG icons, and automated scrolling to ensure proper element visibility. Additionally, the script captures screenshots of the shopping cart to validate the successful addition of items. Error handling mechanisms are integrated to manage scenarios such as unexpected delays, element unavailability, or click interception.

This project showcases the practical application of Selenium WebDriver in testing real-world e-commerce workflows. It highlights best practices in web automation, including synchronization, exception handling, and user interaction simulation. The "Add to Cart" test is an essential aspect of functional testing, ensuring that one of the most fundamental features of BigBasket operates seamlessly, contributing to the platform’s overall reliability and user satisfaction.

Table of contents

CHAPTER 1: INTRODUCTION

This project is an automation solution for testing the "Add to Cart" functionality on the BigBasket website, implemented using Java Selenium WebDriver. It is designed to mimic user interactions with the website, ensuring that the process of adding items to the cart works seamlessly. By automating this critical e-commerce feature, the project demonstrates the potential of Selenium for end-to-end testing of web applications.

The application begins by setting up Selenium WebDriver, specifically utilizing ChromeDriver to launch the BigBasket website. The browser is configured to handle dynamic web elements by implementing both implicit and explicit waits, ensuring robust handling of delayed or dynamically loaded components. Once the website is loaded, the automation workflow navigates to the login or sign-up section, prompting the user to provide their phone number or email. For authentication, the process pauses to allow manual OTP entry, a practical approach for handling scenarios where automated OTP input is not feasible.

After authentication, the automation proceeds to interact with the "Add to Cart" button for a chosen product. It uses techniques such as scrolling to bring the button into view and handles any potential exceptions, such as ElementClickInterceptedException, to ensure smooth interaction. Once the item is added to the cart, the script navigates to the cart page to verify that the operation was successful. To validate this step, it captures a screenshot of the cart, providing visual confirmation of the automation’s effectiveness.

The project emphasizes modular and reusable code design by encapsulating specific functionalities, such as button clicks, text entry, and screenshot capturing, into dedicated methods. This structure enhances the maintainability and scalability of the code. Additionally, comprehensive error handling and detailed logging are implemented throughout the workflow, enabling easier debugging and tracking of each automation step.

This project showcases the power of Selenium for web application testing, particularly for e-commerce platforms where reliability is crucial. By automating key user interactions and providing mechanisms for validation, it ensures that the "Add to Cart" feature functions as expected, delivering a seamless experience for users. It also serves as a foundation for expanding automation to other areas, such as checkout processes or inventory management.

.

CHAPTER 2: LITERATURE SURVEY

1. Selenium-Based Automated Testing for E-Commerce Applications
Source: International Journal of Software Engineering and Knowledge Engineering
This paper examines the application of Selenium WebDriver in automating test cases for e-commerce platforms. The authors emphasize the importance of testing critical functionalities, such as login, search, and cart management. They highlight how tools like Selenium improve testing efficiency by reducing manual intervention, particularly for dynamic web elements. The study also provides insights into handling challenges like AJAX-based components and real-time validations, directly aligning with the challenges addressed in the BigBasket automation project.

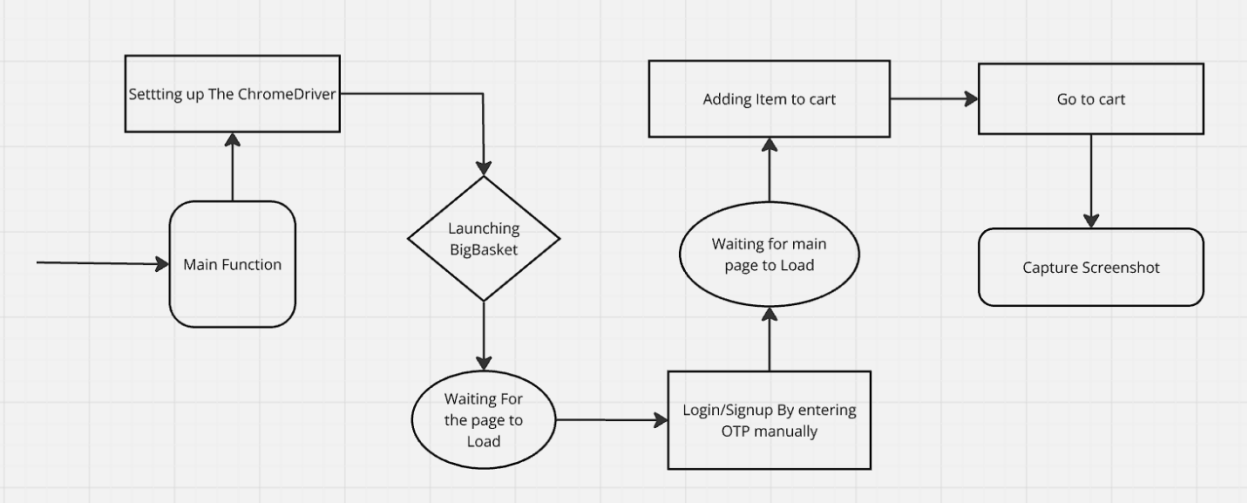
2. Comparative Analysis of Automation Testing Tools for E-Commerce Platforms
Source: Journal of Information Technology Research
The paper compares different automation testing tools, including Selenium, for their effectiveness in testing e-commerce websites. It outlines Selenium's flexibility in integrating with different browsers and programming languages. The research discusses common scenarios tested in e-commerce, such as payment gateway integration and cart management, concluding that Selenium is particularly suited for projects requiring a high degree of customization, such as the "Add to Cart" automation in this project.

3. Dynamic Website Testing: Challenges and Solutions Using Selenium
Source: IEEE Transactions on Software Engineering
This research focuses on the challenges posed by dynamic and AJAX-driven websites during automation testing. The authors propose methods to handle dynamic content, such as explicit waits and JavaScriptExecutor, techniques also used in the BigBasket automation. They demonstrate how these approaches improve test reliability and highlight their application in user-driven workflows, such as navigating through multi-step processes like login and cart validation.

4. Enhancing User Experience Testing for E-Commerce Systems
Source: ACM Transactions on Internet Technology
The study explores the impact of automated testing on the overall user experience in e-commerce applications. It argues that automated validation of features like cart management and seamless navigation significantly improves user satisfaction. The authors present a case study where Selenium was used to simulate real-world scenarios, showing its effectiveness in identifying usability issues and ensuring that core features, such as adding items to a cart, function reliably under various conditions.

5. Test Automation Frameworks for E-Commerce: A Case Study Approach
Source: International Conference on Software Quality Assurance
This paper presents a case study on developing a test automation framework for an e-commerce application using Selenium. The framework was designed to test functionalities like login, product search, and cart operations. The authors discuss the importance of modular design in creating reusable components for tasks like input handling and button clicks. The study aligns with the modular approach taken in the BigBasket automation project, where reusable methods for scrolling, clicking, and entering text were implemented.

CHAPTER 3: PROPOSED SYSTEM



The main function serves as the entry point. It initializes the automation flow by calling the following key methods in sequence:

1. setupDriver()
    - A new instance of the ChromeDriver is created and configured.
    - Browser settings, like window maximization and implicit wait, are applied to ensure smoother interactions.
2. driver.get("https://www.bigbasket.com/")

    - The BigBasket website is launched, and the title of the page is printed to confirm successful navigation.
3. waitForPageLoad()

    - Ensures that the web page has completely loaded before proceeding, using JavaScript to check the document.readyState.
4. handleLoginSignUp()

    - Simulates the user entering their phone number or email address into the login/signup form and clicking the "Continue" button.
5. performSignUpWithManualOTP()

    - Prompts the user to manually enter the OTP received on their phone/email.
    - Verifies the OTP by filling the input field and clicking the "Verify & Continue" button.

6. addItemToCartAndCaptureScreenshot()
    - Performs the main functionality:
    - Locates and clicks the "Add to Cart" button for an item.
    - Scrolls to ensure visibility of the item or button as needed.
    - Clicks on the cart icon to load the shopping cart view.
    - Captures and saves a screenshot of the cart for confirmation of success.
7. cleanup()

    - Closes the browser and cleans up resources, ensuring that the driver instance is properly terminated.

Key Supporting Functions

- clickButton(By locator)
  Waits for a button to become clickable, scrolls to its position, and performs a click operation.

- enterText(By locator, String text)
  Waits for an input field to be visible and sends the specified text to it.

- scrollIntoView(WebElement element)
  Scrolls the browser to make a specific element visible in the viewport using JavaScript.

- scrollToTop()
  Scrolls the browser window to the top of the page for better navigation and visibility.

- captureScreenshot(String fileName)
  Captures the current browser screen and saves it as a PNG file in the "screenshots" directory.

Flow Summary

The program launches the BigBasket website and ensures the page has fully loaded.
It simulates the login or signup process by accepting user input (phone/email and OTP).
After successful login, it navigates to a product, clicks "Add to Cart," and verifies the cart contents.
Finally, a screenshot is captured to confirm the action, and the browser session is closed.
This flow ensures a seamless simulation of the user journey, from login to adding an item to the cart, with robust error handling for dynamic web elements.

CHAPTER 4: IMPLEMENTATION

```
public static void main(String[] args) {
        try {
            setupDriver();
            driver.get("https://www.bigbasket.com/");
            System.out.println("Site launched successfully: " +
driver.getTitle());

            waitForPageLoad();
            handleLoginSignUp();
            performSignUpWithManualOTP();
            addItemToCartAndCaptureScreenshot();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            cleanup();
        }
    }
```

The main method serves as the entry point for the program. It initializes the automation process by setting up the ChromeDriver, navigating to the BigBasket website, and ensuring the page loads fully. It handles user login or signup, simulates entering a manual OTP, and automates adding an item to the cart. Finally, it captures a screenshot of the cart and ensures proper cleanup by closing the browser, even if an exception occurs during execution.

```
private static void setupDriver() {
        try {
            driver = new ChromeDriver();
            driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
            System.out.println("Driver setup complete.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

The setupDriver method initializes the ChromeDriver instance, maximizes the browser window, and sets an implicit wait time of 10 seconds to handle dynamic elements. It ensures the WebDriver

is ready for use and prints a confirmation message. If any error occurs during setup, it is caught and printed.

```
private static void waitForPageLoad() {
        try {
            new WebDriverWait(driver, Duration.ofSeconds(20)).until(
                webDriver -> ((String)
((org.openqa.selenium.JavascriptExecutor) driver)
                        .executeScript("return
document.readyState")).equals("complete"));
            System.out.println("Page loaded completely.");
        } catch (Exception e) {
            System.err.println("Page did not load completely.");
        }
    }
```

The waitForPageLoad method ensures the web page has fully loaded before proceeding. It uses a WebDriverWait with a 20-second timeout to check the document.readyState via JavaScript. If the state equals "complete," the page is considered loaded, and a success message is printed. If loading fails within the timeout, an error message is logged.

```
private static void handleLoginSignUp() {
        try {

clickButton(By.cssSelector("button[color='darkOnyx'][pattern='filled']
"));
            enterText(By.id("multiform"), "8197742770");
            System.out.println("Entered phone/email.");
            clickButton(By.cssSelector("button[type='submit']"));
            System.out.println("Continue button clicked.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

The handleLoginSignUp method automates the login or signup process. It clicks a specific button on the webpage, enters a phone number or email into a designated input field, and submits the form by clicking the "Continue" button. Success messages are logged after each action, and any exceptions encountered during the process are handled and printed.

```
private static void performSignUpWithManualOTP() {
        try {
            System.out.println("Enter the OTP manually:");
            Scanner scanner = new Scanner(System.in);
            String otp = scanner.nextLine();
            enterText(By.cssSelector("input[type='number']"), otp);
            System.out.println("Entered OTP.");
            clickButton(By.xpath("//button[text()='Verify &
Continue']"));
            System.out.println("Verify & Continue button clicked.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

The performSignUpWithManualOTP method handles the manual OTP input during the signup process. It prompts the user to enter the OTP via the console, then enters the provided OTP into the relevant input field. Afterward, it clicks the "Verify & Continue" button to proceed with the signup. Success messages are displayed after each action, and exceptions are caught and printed if they occur.

```
private static void addItemToCartAndCaptureScreenshot() {
        try {
            WebElement addToCartButton = new WebDriverWait(driver,
Duration.ofSeconds(20))

.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("b
utton[color='rossoCorsa'][pattern='outline']")));

            scrollIntoView(addToCartButton);
            addToCartButton.click();
            System.out.println("Item added to cart.");

            scrollToTop();

            clickCartIcon();

            Thread.sleep(2000);
```

```
            System.out.println("Cart is fully loaded. Taking
screenshot...");

            captureScreenshot("cart_screenshot.png");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

The addItemToCartAndCaptureScreenshot method automates the process of adding an item to the cart. It waits for the "Add to Cart" button to be visible, scrolls to it, and clicks it. After the item is added, the method scrolls to the top of the page, clicks the cart icon to view the cart, and waits briefly to ensure the cart is fully loaded. Finally, it captures a screenshot of the cart and saves it with the specified file name. Any errors during the process are caught and printed.

```
public static void clickCartIcon() {
        try {
            WebElement svgContainer = new WebDriverWait(driver,
Duration.ofSeconds(10)).until(

ExpectedConditions.presenceOfElementLocated(By.cssSelector("g[mask='ur
l(#basket_svg__a)']"))
            );

            ((JavascriptExecutor)
driver).executeScript("arguments[0].dispatchEvent(new
MouseEvent('click', {bubbles: true}));", svgContainer);
            System.out.println("Cart icon clicked successfully.");
        } catch (Exception e) {
            System.err.println("Failed to click the cart icon.");
            e.printStackTrace();
        }
    }
```

The clickCartIcon method automates the process of clicking the cart icon on the webpage. It waits for the cart icon element to be present on the page using a WebDriverWait. Once the icon is located, a JavascriptExecutor is used to simulate a click event on it. A success message is printed once the icon is clicked, and any failure is caught and logged with an error message.

```
private static void enterText(By locator, String text) {
        try {
            WebElement element = new WebDriverWait(driver,
Duration.ofSeconds(15))

.until(ExpectedConditions.visibilityOfElementLocated(locator));
            element.sendKeys(text);
        } catch (Exception e) {
            System.err.println("Error entering text: " + text);
            e.printStackTrace();
        }
    }
```

The enterText method automates entering text into an input field on the webpage. It waits for the element, identified by the provided locator, to become visible using a WebDriverWait. Once the element is visible, the method enters the specified text into the input field. If an error occurs during this process, an error message is printed with the problematic text and the exception details.

```
private static void clickButton(By locator) {
        try {
            WebElement button = new WebDriverWait(driver,
Duration.ofSeconds(15))

.until(ExpectedConditions.elementToBeClickable(locator));
            scrollIntoView(button);
            button.click();
            System.out.println("Button clicked successfully.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

The clickButton method automates clicking a button on the webpage. It waits for the button, identified by the provided locator, to be clickable using a WebDriverWait. Once the button is ready, it scrolls the button into view and clicks it. A success message is printed upon successful click, and any exceptions encountered are caught and printed.

```
public static void scrollIntoView(WebElement element) {
        try {
            ((org.openqa.selenium.JavascriptExecutor) driver)
                .executeScript("arguments[0].scrollIntoView({
behavior: 'smooth', block: 'center' });", element);
            Thread.sleep(500);
            System.out.println("Scrolled into the element's view.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

The scrollIntoView method ensures that a specific web element is brought into the visible area of the browser window. It uses JavascriptExecutor to scroll the element into view with a smooth scrolling effect. After scrolling, the method waits briefly (500 milliseconds) to ensure the element is fully in view before printing a confirmation message. Any exceptions encountered during this process are caught and printed.

```
public static void scrollToTop() {
        try {
            ((org.openqa.selenium.JavascriptExecutor)
driver).executeScript("window.scrollTo(0, 0);");
            System.out.println("Scrolled back to the top.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

The scrollToTop method scrolls the webpage back to the top using JavascriptExecutor. It executes a script that moves the viewport to the coordinates (0, 0), which corresponds to the top-left corner of the page. After successfully scrolling, a message is printed to indicate that the action was completed. Any exceptions that occur are caught and printed.

```
public static void captureScreenshot(String fileName) {
        try {
            TakesScreenshot ts = (TakesScreenshot) driver;
            File srcFile = ts.getScreenshotAs(OutputType.FILE);
            File destFile = new File("screenshots/" + fileName);
            destFile.getParentFile().mkdirs();
            Files.copy(srcFile, destFile);
```

```
            System.out.println("Screenshot saved to: " +
destFile.getAbsolutePath());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

The captureScreenshot method captures a screenshot of the current state of the webpage. It uses TakesScreenshot to take the screenshot and stores it in a temporary file. The screenshot is then copied to a specified directory (screenshots/) with the provided filename. If the directory doesn't exist, it is created. A message is printed with the full path of the saved screenshot. If an IOException occurs during this process, the exception is caught and printed.

```
private static void cleanup() {
        if (driver != null) {
            driver.quit();
            System.out.println("Driver closed.");
        }
    }
```

The cleanup method ensures proper cleanup after the test execution. If the driver is not null, it calls driver.quit() to close the browser and release any resources associated with the WebDriver session. After the browser is closed, a message is printed to confirm that the driver has been successfully closed.

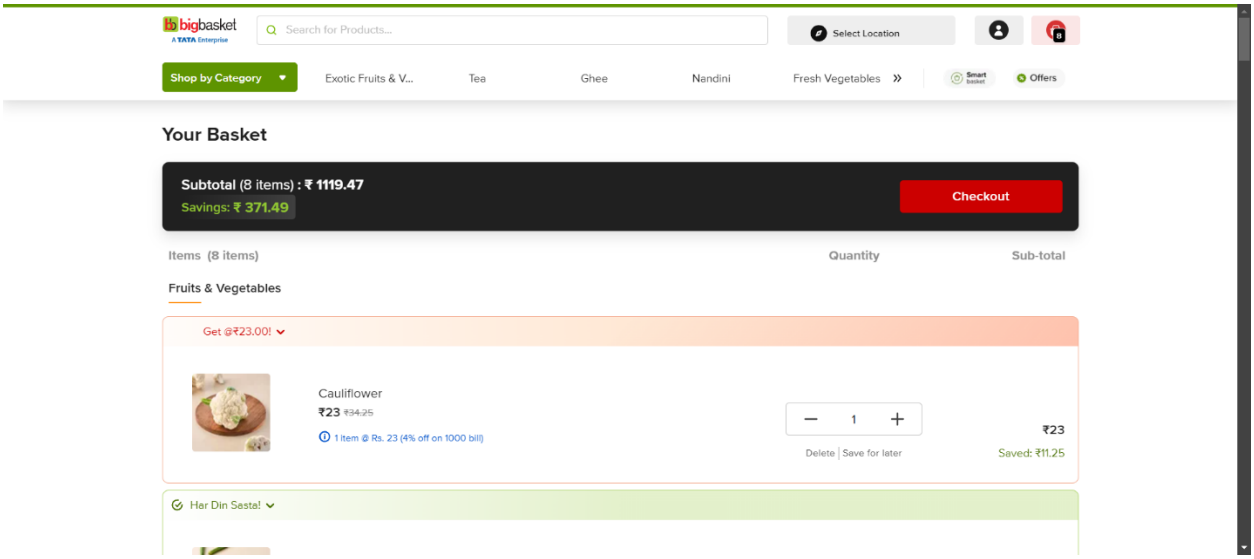
CHAPTER 5: RESULT

```
Driver setup complete.
Site launched successfully: Online Grocery Shopping and Online Supermarket in India - bigbasket
Page loaded completely.
Scrolled into the element's view.
Button clicked successfully.
Entered phone/email.
Scrolled into the element's view.
Button clicked successfully.
Continue button clicked.
Enter the OTP manually:
353217
Entered OTP.
Scrolled into the element's view.
Button clicked successfully.
Verify & Continue button clicked.
Scrolled into the element's view.
Item added to cart.
Scrolled back to the top.
Cart icon clicked successfully.
Cart is fully loaded. Taking screenshot...
Screenshot saved to: E:\1\Code\Java_EclipseIDE\bb\screenshots\cart_screenshot.png
Driver closed.
```

The output shows the sequence of actions performed by your automated BigBasket website interaction:

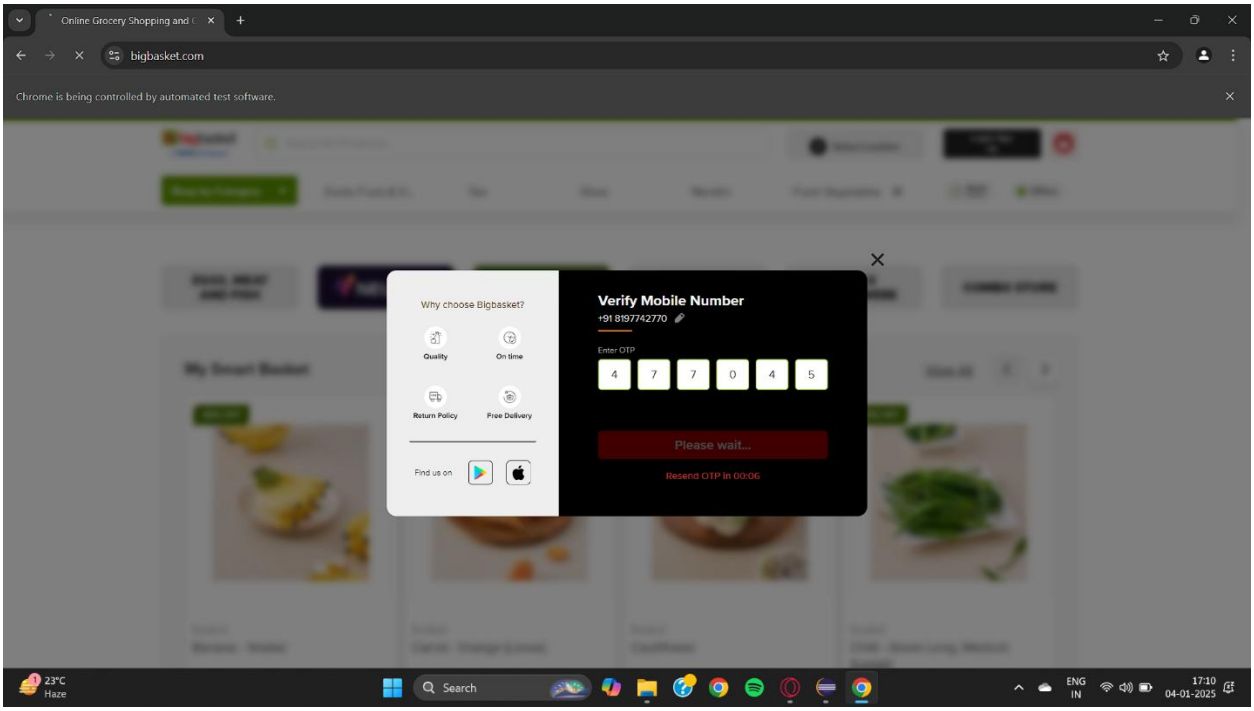
1. **Driver Setup:** The WebDriver is successfully initialized, and the browser window is maximized.
2. **Site Launch:** The BigBasket website is successfully launched, and the page title confirms it is the correct site.
3. **Page Load:** The page is fully loaded after waiting for all elements to be ready.
4. **Login/Signup Process:**
    - The phone/email input is filled, and the "Continue" button is clicked.
    - The OTP is manually entered by the user, and the "Verify & Continue" button is clicked.
5. **Add Item to Cart:**
    - The "Add to Cart" button is located, scrolled into view, and clicked.
6. **Cart Interaction:**
    - The cart icon is clicked successfully.
    - The cart page is fully loaded, and a screenshot of the cart is captured.
7. **Cleanup:** The WebDriver session is closed, and the browser is shut down.

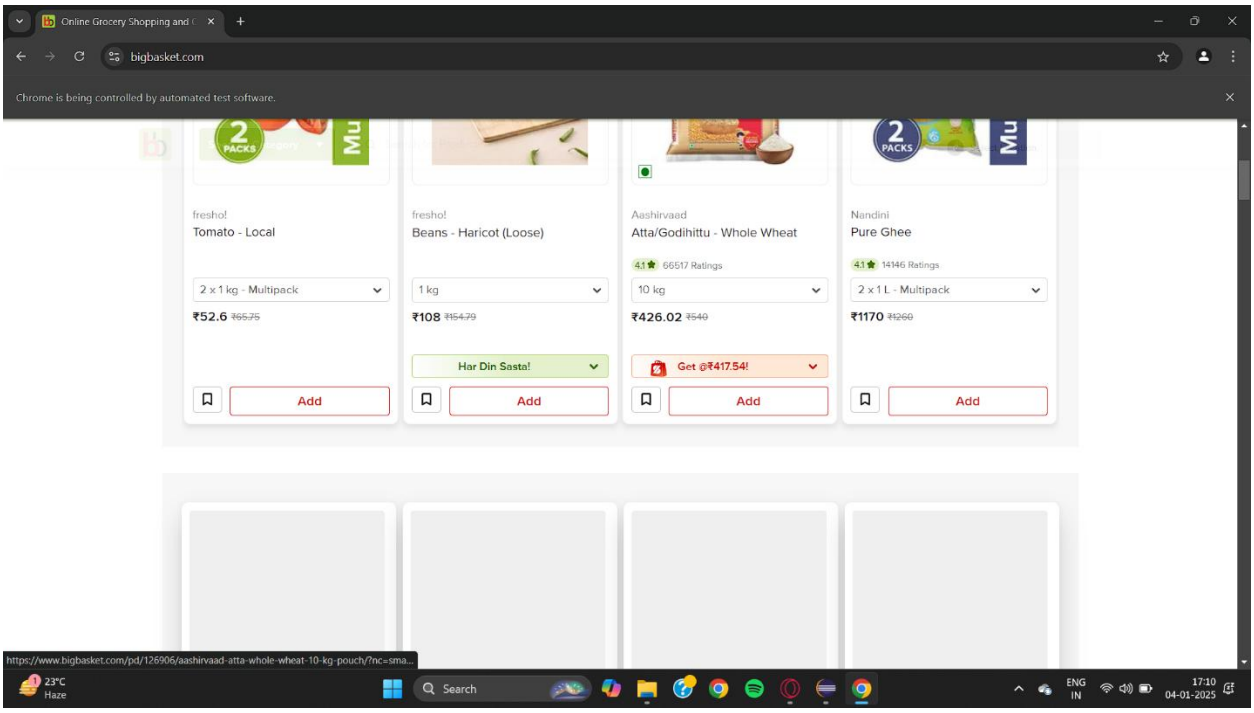
Finally, the screenshot of the cart is saved in the specified directory (E:\1\Code\Java_EclipseIDE\bb\screenshots\cart_screenshot.png).



The screenshot captures the cart page of the BigBasket website after an item has been successfully added to the cart. The page is fully loaded, showing the added item, along with any other relevant cart details. This image serves as a visual confirmation that the item has been added to the cart and the cart page has been displayed properly, reflecting the interaction performed by the automated script. The screenshot is saved in the directory E:\1\Code\Java_EclipseIDE\bb\screenshots\cart_screenshot.png.

Screenshots





CHAPTER 6: CONCLUSION AND FUTURE SCOPE

Conclusion

This BigBasket automation project successfully showcases the power of Selenium WebDriver in automating key tasks such as website navigation, login, OTP verification, adding items to the cart, and capturing screenshots. The script ensures that interactions with the website are smooth by waiting for elements to load and handling dynamic content effectively.

The use of explicit waits ensures that actions are only performed when the page is ready, reducing the chance of errors. By capturing a screenshot of the cart, the script also provides a visual checkpoint for validation.

In conclusion, this automation project demonstrates how Selenium can be used to automate repetitive tasks efficiently, and it serves as a foundation for more complex automation scenarios in the future.

Future Scope

This automation script can be extended further by automating additional functionalities, such as:

1. **Checkout Process**: Automating the entire checkout flow, including entering payment details and completing the purchase.
2. **Error Handling**: Enhancing error handling for various scenarios, such as invalid inputs or failed logins.
3. **Multiple Browsers**: Making the script cross-browser compatible by testing it on different browsers like Firefox and Edge.
4. **Data-Driven Testing**: Integrating data-driven testing, where different products and user credentials are used to test the script with multiple inputs.
5. **Performance Testing**: Analyzing the load time and performance of the website under various conditions.

These improvements will make the automation more robust, versatile, and applicable to a wider range of testing scenarios.

BIBLIOGRAPHY

Selenium Documentation:
Selenium WebDriver is the core tool used for automating browser interactions. The official documentation provides extensive details on how to use the library and best practices.
URL: https://www.selenium.dev/documentation/

Baker, N. (2018). Selenium WebDriver Recipes in Java. Packt Publishing.
This book provides practical examples and recipes to work with Selenium WebDriver using Java, including automating form submissions, handling dynamic content, and more.

Hunt, R., & Thomas, D. (2019). The Pragmatic Programmer: Your Journey to Mastery. Addison-Wesley Professional.
While not focused on Selenium specifically, this book offers valuable advice for developers automating tasks and writing maintainable, efficient code.

Gonzalez, M. (2017). Mastering Selenium WebDriver. Packt Publishing.
This book goes deep into Selenium WebDriver's capabilities and guides developers through advanced topics like handling complex user interactions and working with JavaScript-heavy websites.

Stack Overflow Contributors (2021). Selenium WebDriver Error Handling. Stack Overflow.
A valuable community resource with a range of practical solutions for handling common Selenium WebDriver issues, including element visibility, synchronization, and exception handling.
URL: https://stackoverflow.com/questions/tagged/selenium-webdriver