|  |
| --- |
| [[https://myetudes.org/etudes-melete-tool/images/printer.png](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385332) Send to Printer](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385332) | [Close Window](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385332) |
|  | |
| 21. Testing  21.1. What do we test?  *Copyright (c) 2014, Rula Khayrallah*  It is essential to test a web application thoroughly before releasing it.  It is even better to **write** **the tests before writing the code**.  That is what **Test Driven Development (TDD)** is all about:  we write the test based on how we would like the code to behave, then we write the code that makes the test pass.  This forces us to really understand our requirements before we start writing the code.  For a web application, there are several testing aspects that need to be considered.  Some of these aspects, such as functional testing or usability testing, are common to most software development endeavors.  Others, such as compatibility testing, load testing and security testing, address challenges inherent to web applications.  **Functional Testing:**  Does our application do what it is supposed to do?  Does it behave correctly?  Does it produce the expected outcome?  There are several tools available to create test cases and automate functional testing.  We’ll take a closer look at Selenium in the next section.  **Usability Testing:**  Is our application easy to navigate?  Is it obvious to the user which actions are available?  Does the user interface match the target audience's needs?  **Compatibility Testing**  Does our application run on all browsers, or on all 'modern' browsers, or on a subset thereof?  Does it run on all devices including phones and tablets or is the functionality dependent on the device?  **Security Testing:**  Does our application protect the user's information?  Is it vulnerable to any kind of hackers' attacks?  **Performance Testing:**  What response time does the user experience under normal conditions?  Are there any bottlenecks in our application that we need to address?  We’ll look more closely at performance issues in web applications in an upcoming section.  **Load Testing:**  Does our application perform well regardless of how many users it is serving?  Remember the issues surrounding the launch of the health care web site healthcare.gov…  21.2. Selenium  *Copyright (c) 2014, Rula Khayrallah*  Selenium is a **browser automation tool** that is commonly used for automating the testing of web applications.  It includes two major components:  Selenium IDE and Selenium WebDriver.  **Selenium IDE** is a Firefox plugin.  It lets us record our interactions with the browser as a script. The script is recorded in Selenese, a special language for Selenium. We can then edit the script or play it back.  Selenese provides commands for performing actions in a browser (such as clicking a button or typing something), as well as commands for retrieving data from the resulting pages.  Scripts generated with the Selenium IDE can also be exported to Selenium WebDriver.  **Selenium WebDriver** accepts commands, sends them to a browser and retrieves the results.  The commands may be written in Selenese or in any of the supported languages (through special language drivers). The core languages supported at this time are Java, C#, Ruby, Python and JavaScript (Node).  **Selenium IDE Demo**  To install the Selenium IDE, follow the detailed instructions at:[http://docs.seleniumhq.org/docs/02\_selenium\_ide.jsp#](http://docs.seleniumhq.org/docs/02_selenium_ide.jsp)  After you restart Firefox and open the IDE, read through the IDE features (described on that page) to gain some familiarity with the tool.  In the following demo, we’ll just record some interactions with our guessing game app (guess.html) and demonstrate how we can check the results.  We’ll first open Firefox and the Selenium IDE and make sure that the red recording button is activated.  In the browser address bar, we’ll just enter the file: scheme address of the word guessing game guess.html ([file:///.../..../guess.html](file:///\\...\....\guess.html)).  The exact address will depend on your configuration.  Then we’ll start interacting with the app.  For demonstration purposes, we’ll enter the following letters, in order, pressing the GUESS button after each letter: j, a, v, s, c, r, i, p then q.  At this point we’ll move our mouse over the dashed display javascrip- and right click.  The following options are shown:  We’ll select 'Show All Available Commands' first and then ‘verify Text id=display javascrip-‘ as shown below.  This is how we specify to Selenium to check the page against a known result.  We are simply checking that all the correct letters that we guessed are displayed in the HTML element whose id is display and the last character is still a dash: verify Text id=display javascrip-  The next thing to check is that the letter q has appeared under the Wrong Letters. We can do that again by right clicking on the corresponding element, selecting 'Show All Available Commands' first and then ‘verify Text id=wrong q’ as shown below.    At this point we can stop recording, save the test case and of course replay it many many times.  We can also edit the script generated.  The one thing to note here is that there are **three main options to check results:  an assert option, a verify option and a waitFor option**.   When an assert fails, the test is aborted. When a verify fails, the test will continue execution, logging the failure.  waitFor commands wait for some condition to become true (which can be useful for testing Ajax applications). They will succeed immediately if the condition is already true. They will fail and halt the test if the condition does not become true within a certain timeout setting.  Let’s demonstrate the use of waitFor with our ajaxdemo.html example.  Just remember to serve that page with our in house  node server.  Here again we start recording in Selenium then in the Firefox browser, we type:[http://localhost:8080/html/ajaxdemo.html](http://localhost:8080/ajaxdemo.html)  After the page has loaded, we right click on the heading Ajax Demo, select 'Show All Available Commands’ first then select the command: ‘waitForText css=h2 Ajax Demo’.  We are basically telling Selenium to wait for ‘Ajax Demo’ to appear in the h2 header.  We then press on the CLICK button.  The line ‘This is the additional data requested from the server.’ is supposed to appear under the button.  We can then right click on the line and select the command: ‘waitForText css=span This is the additional data requested from the server.’  So here we are basically telling Selenium to wait for the Ajax response from the server.  At this point we can stop recording, save the test case and of course replay it over and over…  21.3. Performance Tuning and Latency  *Copyright (c) 2014, Rula Khayrallah*  Several metrics may be used to assess the performance of a web application.  **Latency**is defined as the time between making a request and beginning to see a result.   It may also be defined as the time between making a request and the completion of the response.  In both cases we are concerned about the response time that the user sitting in front of the browser, experiences.  How long does it take to load the web page?  That response time is determined by the following components:   * How long does it take to generate the web page (on the server)? * How long does it take for the browser to parse it? * How long does it take to download all the components (scripts, stylesheets, images)? * How long does it take for the browser to render it?   For most applications, the response time is dominated by the time needed to download all the components. As web applications have become increasingly complex, the size of the associated components has considerably increased. Depending on where the user is located, and depending on their network access, it can take a significant amount of time to fetch all the components.  Many tools are available to measure the performance of a web application from the client side.  The tricky part is being able to replicate a typical user’s environment.  **Firebug includes a Net panel** that may be extremely helpful in identifying latency related performance issues.  To see how that works, we first open Firebug, enable the Net panel by clicking on the Net tab and selecting Enable.  We can then type a url address in the address bar, and see detailed information and a timeline corresponding to all the requests that had to be sent to get the initial web page to load.  The screenshot below shows the timeline corresponding to loading the page at<http://foothill.edu/index.php>  We can also select to view a subset of the requests such as JavaScript only or images only.  Note that the timeline will depend on whether or not some of these resources have been cached earlier.  21.4. Load Testing and Throughput  *Copyright (c) 2014, Rula Khayrallah*  Load testing is primarily focused on back end performance. Its objective is to determine the **maximum throughput** that a web application can sustain.  **Throughput is defined as the number of requests (or transactions) that an application can handle per unit of time.**  Several load testing tools are available today.  They typically generate a heavy concurrent load, verify responses, and produce a variety of reports.  To determine the maximum throughput, the number of requests needs to be increased until the throughput levels off or starts to drop.  That usually happens because of a bottleneck in the application (commonly found in the persistence layer.) |  |