Web GUI testing

Validation and Verification
University of Rennes 1

Erwan Bousse (erwan.bousse@irisa.fr)

Last update September 22, 2015

This lab session is about testing the GUI of web applications. We will use the Selenium tools to do some blackbox testing of a simple web application.

1 Preparing the test environment

The application to test is called *MDMS*. It is a very simple web application used for software engineering research experiments, which can display a set of articles written using *Markdown*[1] syntax.

To avoid installing all the required packages on your system, and also to easily isolate the application and reset its context, we will rely on *Docker*[2] both for the database and the web server and application.

1.1 Docker installation

1. To install the latest version of docker on Ubuntu 14.04, follow the following instructions (from `http://olivier.barais.fr/blog/posts/2014.08.29/Operation_portable_M2_ISTIC.html`):

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 \
    --recv-keys 36A1D7869245C8950F966E92D8576A8BA88D21E9
$ sudo sh -c "echo deb https://get.docker/ubuntu docker main \
    > /etc/apt/sources.list.d/docker.list"
$ sudo apt-get update
$ sudo apt-get install lxc-docker
```

2. While it is not mandatory, you can add yourself to the `docker` group in order to be able to use docker without `sudo`. This can be done the following way:

```
$ sudo usermod -a -G docker YOUR_LOGIN_NAME
```

[1] `https://en.wikipedia.org/wiki/Markdown`
[2] `https://www.docker.com/`

3. Finally, with the provided Java class we will interact with Docker HTTP API, thus we have to edit the options given to the docker daemon to enable this API. We have to add the option "`-H 127.0.0.1:5555 -H unix:///var/run/docker.sock`". The way to do this will depend on the used operating system; here are two examples:

   Fedora: In the file `/etc/sysconfig/docker`, append the option to the `OPTIONS` variable.

   Ubuntu: In the file `/etc/default/docker`, append the option to the `DOCKER_OPTS` variable (you might have to uncomment or create the variable).

   Don't forget to restart the docker service after doing this! If you are not sure of how to do that, you can simply restart your computer. Using Ubuntu, you can do the following:

   ```
   $ sudo service docker restart
   ```

1.2 Docker usage for MDMS

We won't go very far with Docker, we will simply see the commands that you may need to get the images, create containers and remove containers.

Start Docker service Before using Docker, you must start the corresponding system service. With Ubuntu 14.04, the service is (I think) automatically started after install and at each boot. To start it manually:

```
$ sudo service docker start
```

Download MDMS images We need a *redis* image to have a working database, and the MDMS image for the web server and app. To obtain them:

```
$ docker pull maxleiko/mdms
$ docker pull redis
```

Create and start MDMS containers Using the images, we can instantiate one container named `mdms-redis` with a redis database, and one container named `mdms` with the web application. However we must make sure to connect them so that the web server can use the database, and we must make sure to forward a port (8080) of the host system to the web application container. Use these commands:

```
$ docker run --name mdms-redis -d redis
$ docker run --name mdms --link mdms-redis:mdms-redis -p 8080:8080 \
     --rm -it maxleiko/mdms
```

Deleting MDMS containers You can use this command to destroy all containers whose name contains the word `mdms`.

```
docker rm -f $(docker ps -a | grep mdms | grep -o '^.\{12\}')
```

Deleting MDMS images You can use this command to delete the two images used in the lab session (`redis` and `maxleiko/mdms`). **WARNING: don't do that during the lab session! This is just for cleaning up!**

```
docker rmi $(docker images | grep '^maxleiko/mdms\|^redis' \
    | grep -o '[[:alnum:]]\{12\}')
```

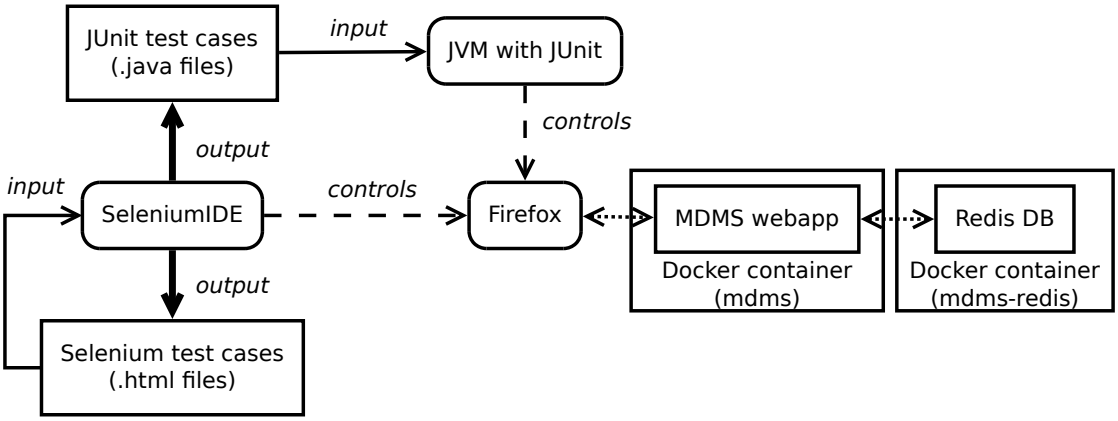

Figure 1: Workflow with Selenium IDE, JUnit and Firefox.

2 Selenium presentation

Selenium[3] is a suite of tools to test web applications. These tools are made by OpenQA, and are distributed under the Apache 2.0 open source license. A test tool for a GUI web application is defined by:

- The ability to simulate the action of a user with support actions carried out with the keyboard and mouse (click a field entry , select it from a dropdown list, etc.).
- Using this simulation ability, the ability to record/write test cases, which are based on use cases
- The ability to perform these simulations in an automated way.

Selenium is composed of three elements :

- Selenium WebDriver: an API to either interact directly with a web brower or to interact with a Selenium Server.
- Selenium IDE : Firefox extension that can record and run tests and test suites. Tests can then be exported in three different formats:
  - Selenium test cases (`.html`), which can be executed by Selenium IDE
  - JUnit test cases with local WebDriver bindings (Java tests that directly interact with Firefox, see Figure **??**)
  - JUnit test cases with remote WebDriver bindings (Java tests that interact with a Selenium Server, in charge of interacting with web browsers.)
- Selenium Server: can interact with browsers to run the tests. By deploying such a server on a remote system, it avoids the test writer to handle the (sometimes) many required browser instances, and helps to automate the process. We won't be using this component.

[3] `http://docs.seleniumhq.org/`

2.1 Installation

To install Selenium IDE, go to this page `http://docs.seleniumhq.org/download/` and download and install the latest Selenium IDE Firefox extension.

2.2 Usage

For our tests, we won't use a Selenium Server, and will only use Selenium IDE and Selenium WebDriver API, which will directly interact with a local Firefox. Here is basically how you can proceed:

1. Launch Firefox with Selenium IDE installed.
2. Open the Selenium IDE window by doing Tools Menu > Selenium IDE, or by clicking on the Selenium IDE button.
3. To start recording, press the record button (red circle).
4. In your Firefox window, go to the page to test.
5. Run the different stages of your scenario. Don't forget assertions! (Selenium provides a contextual menu available on the whole web page to do that)
6. Once your script finished, stop the recording by clicking on button again.
7. *(not mandatory)* Save your test (in Selenium HTML format).
8. Launch your test to see if it works as planned.
9. If necessary, refine your test. For instance:
    - Add command "pause" if Selenium is faster than your application or detects the wrong end of page loading
    - Add the Selenium commands that Selenium IDE failed to record (for certain menus / forms with javascript)
10. Export your test into a JUnit test class: File> Export in Test ... > JUnit4 (WebDriver).
11. The tests can then be imported into a Java project, and run with JUnit.

Remark on html elements IDs You will notice that when recording, selenium stores the IDs of the html elements you interact with. This works most of the time, yet sometimes IDs of the page are not deterministic and vary from a situation to another. To cope with this issue, **you can change the way Selenium finds elements by switching from *id* to *xpath:position* in the "Target" drop-down box below the "Command" box.**

3 Apache JMeter presentation

Apache JMeter is a load testing tool for analyzing and measuring the performance of a variety of services, with a focus on web applications. The tool provides a GUI to both design tests, execute them, and visualize results.

3.1 Installation

You can go to this address and download the latest binaries: `https://jmeter.apache.org/download_jmeter.cgi`. Then extract it anywhere, and start it with `bin/jmeter.sh`.

3.2 Usage

For this lab session we will use JMeter to create very simple HTTP accesses tests to check both response code and response time in defined conditions.

1. Start JMeter GUI. You will use the *Test Plan* and you can ignore the *WorkBench*.
2. Set the language to English (menu *Options*).
3. Right click on *Test Plan* and create a *Thread Group*, with different parameters:
   - *Number of threads/users*: the number of users to simulate
   - *Ramp-up period*: time to reach the full number of threads chosen
   - *Loop count*: number of accesses per user
4. In the *Thread Group*, create the following elements:
   (a) Create a *HTTP Request default* with a server name (e.g. `google.fr`) and port (e.g. 443). This sets the default server for the whole thread group.
   (b) Create requests of type *HTTP Request*. This tells our thread group to perform HTTP requests to the default server, to some specific path (e.g. `/mail`). You don't have to put a server here, since the default server is used.
   (c) Create a listener of type *Graph Results*. This allows you to visualize some metrics over time, such as *throughput* (the number of requests handled per minute).
   (d) Create assertions, such as a *Duration Assertion* to check is the response time is satisfying or a *Response Assertion* to check if the response code of the web server is the one expected.
   (e) Create a listener of type *Assertion Results*, to visualize whether the assertions are passed or not. You can choose to only display errors.

To run the tests:

- To run the tests, click on the green arrow, or press Ctrl+R
- To clear the listeners (ie the results), click on the little brooms, or press Ctrl+E.
- Note that on the top right corner you can see the number of threads currently created. While the little square is green, it means that the testing session is not finished.

4 What you must do

4.1 Functional testing

The goal of this part is to write blackbox/functional tests for the MDMS web application, using Selenium. You will use Selenium IDE to create the tests, and you will export the tests to run them with JUnit. Note that since we are doing blackbox testing, you do not have to patch the tested program.

Programming 1 *Using Selenium IDE, record/write JUnit tests for the following requirements. Don't hesitate to re-create the containers in order to reset the database. There is only one user available whose credentials are* `admin/admin`.

1. *Appearance*
   (a) *The title of the welcome page is "MdMS - Diversify"*
   (b) *When logged-in, the user name is visible in the top-left corner.*
2. *User rights*
   (a) *You can log-in with some existing user and password.*
   (b) *You cannot log-in with the wrong password.*
   (c) *You cannot log-in with a user name that doesn't exist.*
   (d) *Someone not logged in cannot edit an article.*
3. *Content editing/viewing (as a logged in user only)*
   (a) *You can write an article*
      - *An article requires a title.*
      - *An article requires some content.*
      - *When a title is created with markdown syntax* `=====`*, a* `<h>` *title is displayed in the markdown preview.*
      - *When created, an article is visible on the main page (note: you may have to use xpath to point to the right element).*
   (b) *You can erase an article.*

Programming 2 *Export your tests in JUnit classes (WebDriver). Do what you need to integrate them within the provided Maven project, by extending the* `AbstractMDMSTest` *abstract class with each of your test classes.* `AbstractMDMSTest` *contains a set-up method that will recreate new containers at each test, and a tear-down method to destroy the used containers. This allows us to be 100% sure to have a clean initial context in each test!*

4.2 Performance testing

In this part, you must design performance/non-functional tests for MDMS. You will use JMeter to perform simple load testing.

Programming 3 *We want to see if the website can survive the following conditions:*

- *500 users;*
- *during 5 seconds;*
- *Each doing 10 accesses.*

The expected properties are:

1. *The server always returns a "200 OK" code.*
2. *The response time is less than 30 milliseconds.*

5 What to produce

You have to produce:

- A Maven project in tar.gz or zip format. It must contain:
  - The source code of all your commented tests
  - The generated test report (with javadoc)
- A JMeter test suite.
- A tiny report explaining if the JMeter tests were passed or not.