Automated System Testing with Selenium2

Demonstrate how to use Selenium to test a modern interactive (JavaScript driven) web application, and some of the problems involved with automated GUI testing[1].

This is a StudyPoint exercise and will end up as part of an exam-question as sketched below:

Demonstrate your solution to the exercise "Automated System Testing with Selenium2".

You should (as a minimum):

- Discuss Pros and Cons with manual versus automated tests
- Explain about the Test Pyramid and whether this exercise supports the ideas in the Test Pyramid
- Discuss some of the problems with automated GUI tests and what makes such tests "vulnerable"
- Demonstrate details in how to create a Selenium Test using the code for the exercise
- Explain shortly about the DOM, and how you have read/manipulated DOM-elements in your test
- Explain how (and why it was necessary) you have solved "waiting" problems in your test

---

Getting started

This exercise requires the system under test to be installed locally. This is the system you must test using Selenium:

- Install (if not already done) *node.js* (the LTS version): https://nodejs.org/en/download/
- Clone this project somewhere on your system: https://github.com/Lars-m/seleniumExCompiled.git
  This is a simple "modern" Single Page Application using a REST/JSON-server as backend.
- **Setup the project** In the root of the project type: `npm install`
- **Start the backend:** Open a (new) terminal (in the root of the cloned project) and type: `npm run backend` (Leave this open for the rest of the exercise)
- **Start the client**: Open a terminal (in the root of this project) and type: `npm run client`
- Verify that everything is OK by opening a browser with this URL: http://localhost:3000

The Test Scenario

This is the scenario you must automate (note: order matters in the following[2]):

1. Verify that data is loaded, and the DOM is constructed (Five rows in the table)
2. Write 2002 in the filter text and verify that we only see two rows
3. Clear the text in the filter text and verify that we have the original five rows
4. Click the sort “button” for Year, and verify that the top row contains the car with id 938 and the last row the car with id = 940.
5. Press the edit button for the car with the id 938. Change the Description to "Cool car", and save changes. Verify that the row for car with id 938 now contains "Cool car" in the Description column
6. Click the new “Car Button”, and click the “Save Car” button. Verify that we have an error message with the text “All fields are required” and we still only have five rows in the all cars table.
7. Click the new Car Button, and add the following values for a new car
   a. Year: 2008
   b. Registered: 2002-5-5
   c. Make: Kia
   d. Model: Rio
   e. Description: As new
   f. Price: 31000

   Click “Save car”, and verify that the new car was added to the table with all the other cars.

---

[1] You can freely choose technology stack as long as you use Selenium and similar web app & tests in your solution.

[2] You can reset data to the original five cars via this URL in your browser using: localhost:3000/reset

The Tasks

Implement a x-unit test scenario, using Selenium on your chosen platform (Java/JUnit, C#/NUnit, JavaScript/Mocha etc.)

Hints:

You can, but do not have to, use this getting started project to see Selenium combined with JUnit:
https://github.com/Lars-m/SeleniumExerciseGettingStarted

1) The provided project will not work with the HTMLUnit headless browser, which does not like "pure" JavaScript projects, bundled into a single file.

2) See slides for hints related to **DOM-manipulation**

3)

Most/many test frameworks do not guarantee the order in which tests are executed. With JUnit 4.x you can set the order using this annotation on test class:

```
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
```

Now just name your test methods like `test-1`, `test-2` ….

Testing a real system (which is what the provided system simulate) leads to a number of problems with the following being just a few:

4) It must be setup to use a test database and (obviously) not the production server.

You can assume that, this is what you have with the provided system

5) How do we ensure that test data are consistent, if our test changes data (on a production-like server)?

- You can write your tests, so that they always set data back to where we started (undo each operation)
- In a real system, using a real database, run a script up against the database before your tests (or something similar to that)
  The system under test is written by a developer meant to be tested by developers ;-), so it implements a feature to simulate the behaviour suggested above.

  This url (`http:localhost:3000/reset`) will reset the database to the original five cars.

  With Java, we can use `RestAssured` to very easily perform a programmatic get request.

  Add this entry to your POM-file:

```
<dependency>
   <groupId>com.jayway.restassured</groupId>
   <artifactId>rest-assured</artifactId>
   <version>2.9.0</version>
</dependency>
```

  Make the reset-request like this in your test fixture(s).

```
com.jayway.restassured.RestAssured.given().get("http://localhost:3000/reset");
```

6) How do we get the id's or similar, and how do we ensure they won't change in the future?

Your answer to this question will be relevant when you now turn back to the initial questions stated in the exam-part

How to hand in:

Push your solution to github, including an image with test results in the root.
Publish your git-url in the hand-in document on Moodle.