I. Test plan

Introduction

This documentation / test plan is created to follow and explain the details and justify the decisions during the development and execution. The main application of the this testing exercise will be a webshop, which is meant to be a practice website for test automation. The goal of this test plan is to choose some existing functionalities and make sure that they are working as expected.

Test items

The main functionalities of our applications are:

- Registration
- User management
  - Order history
  - Credit slips
  - Addresses
  - Personal information
  - Wishlist
- Sign in
- Searching for an article
- Main menu bar with quick browsing
- Browsing and filtering the articles
- Article details
- Purchasing
  - Adding / removing items to cart
  - Review / change address
  - Review / change shipping
  - Review / change payment method
  - Confirm order
- Marketing (teasers, popular, best seller categories)
- Footer with legal and additional informations, and links

Features to be tested

The list of features to be tested:

- Registration
- Sign in
- Searching for an article

- Purchasing
    - Adding items to cart
    - Review address
    - Review shipping
    - Review / change payment method
    - Confirm order

Features not to be tested

- User management
    - Order history
    - Credit slips
    - Addresses
    - Personal information
    - Wishlist
- Main menu bar with quick browsing
- Browsing and filtering the articles
- Article details
- Purchasing
    - Removing items to cart
    - Change address
    - Change shipping
- Marketing (teasers, popular, best seller categories)
- Footer with legal and additional informations, and links

Approach

Looking at our current situation, we have a webshop with a fair amount of features, but we have restricted time and capacity to complete the testing.
In situations like this we have to ensure, that at least our core functionalities are working as expected. This means we have to identify our main features, which can make up an MVP, and cover them with test cases, in our case automated, black box, functional UI tests.
Based on the above mentioned criterias and restrictions, we'll create a smoke testing suite.

Items pass/fail criteria

Test cases:

- The prepared smoke test suite (BasicTests.feature) should be GREEN/PASSED.

Features:

- All test cases PASSED and NO P1, P2 defects are found / open.

Test deliverables

- Test plan
- Test cases
    - Search_for_a_product_should_work
    - User_registration_and_sign_in_should_work
    - Purchasing_one_article_should_work
- Test result

Test environment

Product:
- Deployed, working web application

Testing tools:
- Cucumber functional test cases, written in Java
- IDE for test case execution

II. Jenkins integration

Why Jenkins and Selenium?

- Jenkins is the leading open-source continuous integration tool.
- Running Selenium tests in Jenkins allows you to run your tests every time your software changes and deploy the software to a new environment when the tests pass.
- Jenkins can schedule your tests to run at specific time.
- You can save the execution history and Test Reports.

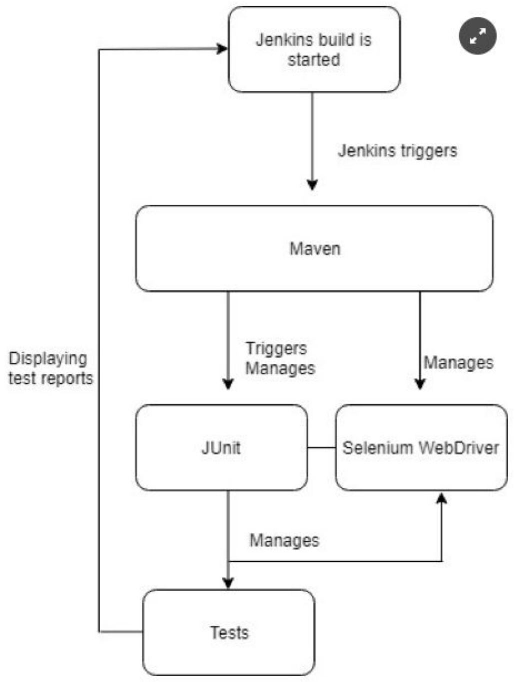
What is and why Maven?

Maven is a powerful project / build management tool, based on the concept of a POM (Project Object Model) that includes project information and configuration information for Maven.

- Maven is used to define project structure, dependencies, build, and test management.
- Using pom.xml(Maven) you can configure dependencies needed for building testing and running code.
- Maven automatically downloads the necessary files from the repository while building the project.

Integration and pipeline

Here is how the tools interact:



The easiest way to develop, manage, run and integrate a Java + Selenium + Cucumber project is with using Maven. The process is simple and there are numerous how-to guides.

Given we have an already setup Jenkins server and a working Java project with Maven. First we have to define our custom Maven test profile eg. smoke test suite. Second we just have to create a new Maven project on our Jenkins server, and configure the Git repository and branch, the build trigger, the pom.xml location and our Maven goals and options (profile).
Last but not least, we can define thresholds, also we have the possibility of notifying specified users regarding the results of a build via e-mail, save and populate the test results or the feature of archiving different artifacts generated at the end of a build.

A simple pipeline would look like this:

- Build
- Deploy (test env)
- Execute
- Report
- Deploy (beta env)