

Test Automation

Selenium Test Automation

The sample is a Maven project. The project has the following dependencies implemented: Selenium WebDriver (framework for test automation), TestNG (framework for testing), Allure (framework for report generation), Html Elements (framework for description of pages block structure)

1. Project Configuration

The required dependencies should be added to the pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>${selenium.version}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>${testng.version}</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>ru.yandex.qatools.htmlelements</groupId>
    <artifactId>htmlelements-java</artifactId>
    <version>${htmlelement.version}</version>
  </dependency>

  <dependency>
    <groupId>ru.yandex.qatools.allure</groupId>
    <artifactId>allure-testng-adaptor</artifactId>
    <version>${allure.version}</version>
  </dependency>
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${aspectj.version}</version>
  </dependency>
</dependencies>
```

Also, the Maven Surefire plugin should be configured. Specify the test classes required for execution. For report generating, plug Allure:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.19.1</version>
  <configuration>
    <properties>
      <property>
        <name>delegateCommandSystemProperties</name>
```

```
        <value>true</value>
    </property>
    <property>
        <name>haltOnfailure</name>
        <value>>false</value>
    </property>
    <property>
        <name>usedefaultlisteners</name>
        <value>>false</value>
    </property>
</properties>
<systemProperties>
    <property>
        <name>allure.results.directory</name>
        <value>${project.build.directory}/allure-results</value>
    </property>
</systemProperties>
<suiteXmlFiles>
    <suiteXmlFile>${project.build.testOutputDirectory}/testng.xml</suiteXmlFile>
</suiteXmlFiles>
<argLine>-
javaagent:${settings.localRepository}/org/aspectj/aspectjweaver/${aspectj.version}/aspectjweaver-
${aspectj.version}.jar</argLine>
</configuration>
</plugin>
```

A test suite is described in testng.xml:

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Simple test suite" verbose="1" >

    <test name="Doodles Archive Page Functionality" >
        <classes>
            <class name="com.qatestlab.automation.demodoodle.tests.DoodlesArchivePageTests" />
        </classes>
    </test>

</suite>
```

2. Test Scripts Design. Project Structure

The pattern PageFactory is used in the project. It enables to specify the work with the website under test by using pages classes the elements of which will be automatically initialized.

Also, do not forget to annotate pages methods using @Step. The annotation specifies to Allure framework what methods calls should be interpreted as scenario steps and displayed in reports.

Description of Google Main Page:

```
import com.qatestlab.automation.support.pages.BasePage;
import org.openqa.selenium.support.FindAll;
import org.openqa.selenium.support.FindBy;
import ru.yandex.qatools.allure.annotations.Step;
import ru.yandex.qatools.htmlElements.element.Button;

public class MainPage extends BasePage {

    @FindAll({
        @FindBy(css = "input[jsaction='sf.lck']"),
```

```
        @FindBy(css = "input[onclick*=doodles]")
    })
    Button iFeelLuckyButton;

    @Step("Open main page")
    public MainPage open() {
        open("https://google.com");
        return this;
    }

    @Step
    public void clickIFeelLuckyButton() {
        iFeelLuckyButton.click();
    }
}
}
```

Description of Doodles Archive Page:

```
import com.qatestlab.automation.demodoodle.pages.blocks.DoodleWebElement;
import com.qatestlab.automation.support.pages.BasePage;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import ru.yandex.qatools.allure.annotations.Step;

import java.util.List;

public class DoodlesArchivePage extends BasePage {

    @FindBy(id = "loading")
    WebElement loader;

    List<DoodleWebElement> doodleCards;

    @Step("Open doodles page")
    public DoodlesArchivePage open() {
        open("https://google.com/doodles");
        return this;
    }

    @Step
    public DoodlesArchivePage loadNewArchiveDoodles() {
        scrollPageDown();
        driverWait.waitForElementInvisibility(loader);
        return this;
    }

    @Step
    public int getArchiveDoodlesAmount() {
        return doodleCards.size();
    }
}
}
```

Based on the sample provided above, to describe the Archive Page, the block `DoodleWebElement` is used. It is responsible for specification of one block on the Doodle Page. Blocks description and use of basic element, e.g., buttons or fields, are conducted using the framework `Html Elements`. The framework enables to specify the repeated blocks as separate classes and use them in page descriptions.

Description of our element `DoodleWebElement`:

```
import org.openqa.selenium.support.FindBy;
import ru.yandex.qatools.allure.annotations.Step;
import ru.yandex.qatools.htmlelements.annotations.Name;
import ru.yandex.qatools.htmlelements.element.HtmlElement;
import ru.yandex.qatools.htmlelements.element.TextBlock;

@Name("Doodle card block")
@FindBy(css = "#archive-list .doodle-thumb")
public class DoodleWebElement extends HtmlElement {

    @FindBy(className = "title")
    private TextBlock titleLabel;

    @Step
    public String getTitle() {
        return titleLabel.getText();
    }
}
```

During pages description, the basic class BasePage is used. It includes the core functionality for all the pages and is responsible for proper initialisation of pages samples (initialisation of web elements):

```
import com.qatestlab.automation.demodoodle.tests.DoodlesArchivePageTests;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.support.PageFactory;
import ru.yandex.qatools.allure.annotations.Step;
import ru.yandex.qatools.htmlelements.loader.decorator.HtmlElementDecorator;
import ru.yandex.qatools.htmlelements.loader.decorator.HtmlElementLocatorFactory;

public abstract class BasePage {
    private WebDriver driver;
    protected WebDriverWaitLogged driverWait;
    private JavascriptExecutorLogged jsExecutor;

    public BasePage() {
        driver = DoodlesArchivePageTests.driver;
        driverWait = new WebDriverWaitLogged(driver);
        jsExecutor = new JavascriptExecutorLogged(driver);

        PageFactory.initElements(new HtmlElementDecorator(new HtmlElementLocatorFactory(driver)),
            this);
    }

    @Step
    protected void open(String url) {
        driver.navigate().to(url);
    }

    @Step
    protected void scrollPageDown() {
        jsExecutor.executeScript("window.scrollTo(0, document.documentElement.scrollHeight)");
    }
}
```

After describing all required pages, it is time to develop test scripts for interaction with our pages and to check the behavior of website under test:

```
import com.qatestlab.automation.demodoodle.pages.DoodlesArchivePage;
import com.qatestlab.automation.demodoodle.pages.MainPage;
import com.qatestlab.automation.support.DriverFactory;
import com.qatestlab.automation.support.logging.AllureTestListener;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Listeners;
import org.testng.annotations.Test;

import java.io.File;
import java.util.concurrent.TimeUnit;

@Listeners({AllureTestListener.class})
public class DoodlesArchivePageTests {
    public static WebDriver driver;

    @BeforeClass
    public void setUp() {
        System.setProperty(
            "webdriver.chrome.driver",
            new
File(DriverFactory.class.getResource("/chromedriver.exe").getFile()).getPath());
        driver = new ChromeDriver();

        driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);
        driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);

        driver.manage().window().maximize();
    }

    @AfterClass
    public void tearDown() {
        if (driver != null) {
            driver.quit();
        }
    }

    @Test
    void navigateToDoodlePageTest() {
        new MainPage()
            .open()
            .clickIFeelLuckyButton();
        Assert.assertTrue(
            driver.getCurrentUrl().endsWith("/doodles"),
            "Doodles page is not opened.");
    }

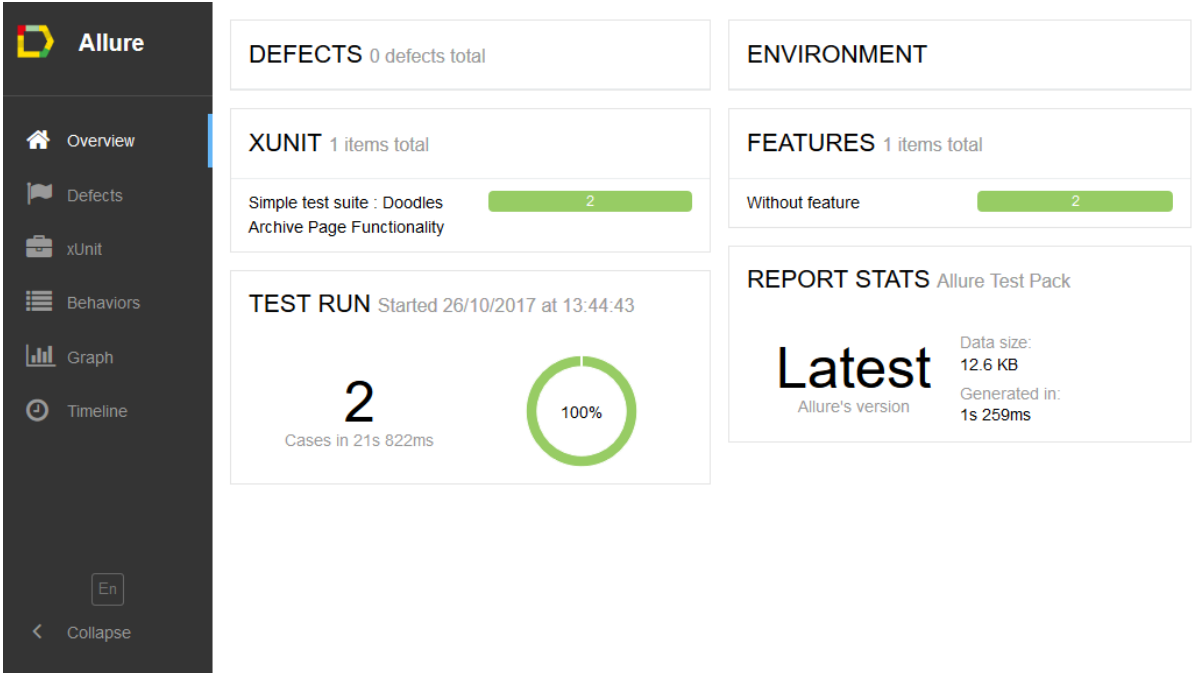
    @Test
    void loadMoreArchiveDoodlesTest() {
        DoodlesArchivePage page = new DoodlesArchivePage().open();
        int doodlesBefore = page.getArchiveDoodlesAmount();
        page.loadNewArchiveDoodles();
        Assert.assertTrue(
            doodlesBefore < page.getArchiveDoodlesAmount(),
            "Doodles are not loaded after scrolling.");
    }
}
```

To describe test classes, TestNG annotations are used. Also, the listener of Allure framework is connected to have the required report generated after test execution.

3. Test Execution and Result Generation

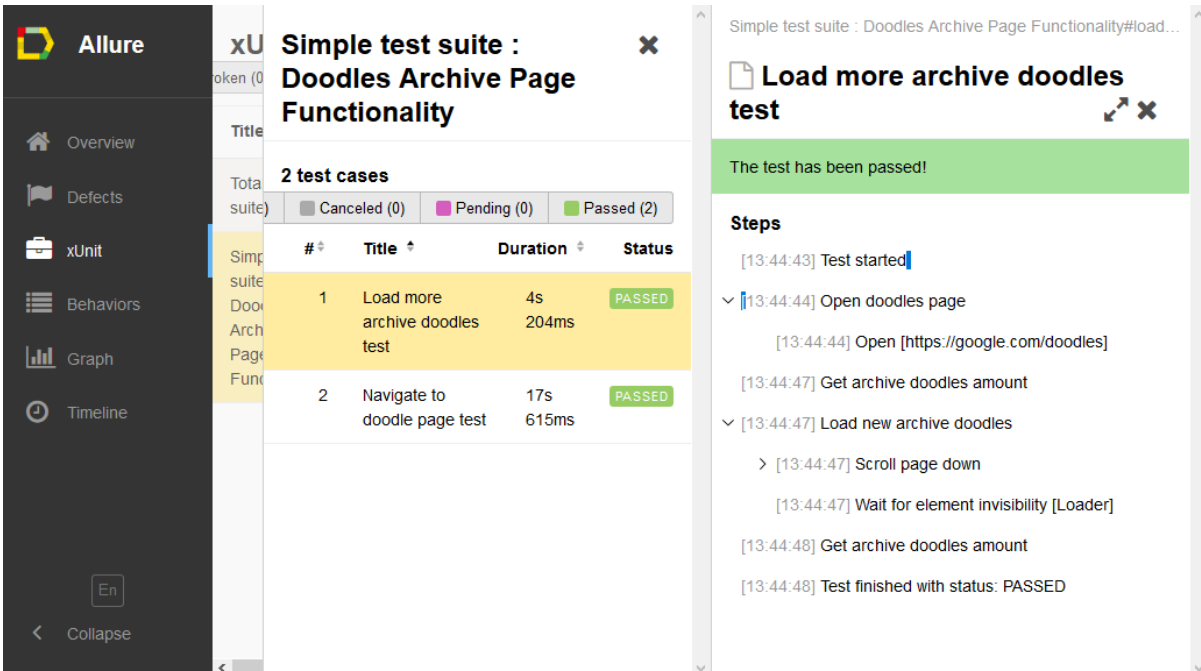
Tests can be executed by calling a test phase in Maven. And the tests will be called in such a way as we have configured at the first stage.

After test execution, a user has Allure reports available. They can contain a lot of additional information added to a list of performed steps. For example, in case of error, the option of screenshot and saved web page attachment is available.



The Allure dashboard overview shows the following summary:

- DEFECTS**: 0 defects total
- ENVIRONMENT**: (empty)
- XUNIT**: 1 items total
 - Simple test suite : Doodles Archive Page Functionality: 2 (100%)
- FEATURES**: 1 items total
 - Without feature: 2 (100%)
- TEST RUN**: Started 26/10/2017 at 13:44:43
 - 2 Cases in 21s 822ms
 - 100% completion
- REPORT STATS**: Allure Test Pack
 - Latest Allure's version
 - Data size: 12.6 KB
 - Generated in: 1s 259ms



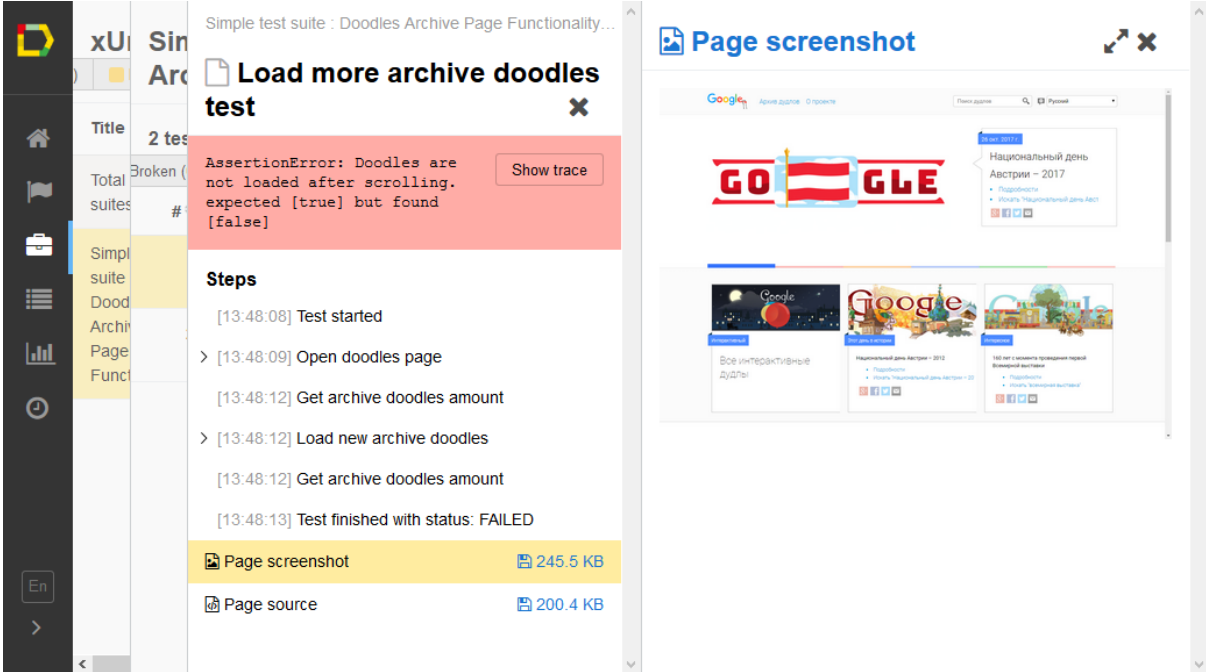
The Allure test case details for "Simple test suite : Doodles Archive Page Functionality" show the following:

- 2 test cases**: Canceled (0), Pending (0), Passed (2)
- Test Case 1**: Load more archive doodles test (4s 204ms) - PASSED
- Test Case 2**: Navigate to doodle page test (17s 615ms) - PASSED

The test steps for the first case are:

- [13:44:43] Test started
- [13:44:44] Open doodles page
 - [13:44:44] Open [https://google.com/doodles]
- [13:44:47] Get archive doodles amount
- [13:44:47] Load new archive doodles
 - [13:44:47] Scroll page down
 - [13:44:47] Wait for element invisibility [Loader]
- [13:44:48] Get archive doodles amount
- [13:44:48] Test finished with status: PASSED

The example of test report with detected issue (the error of checking page functionality, scrolling does not work):



The image shows a test report interface on the left and a page screenshot on the right. The test report, titled "Simple test suite : Doodles Archive Page Functionality...", displays a failed test case "Load more archive doodles test". An error message is shown in a red box: "AssertionError: Doodles are not loaded after scrolling. expected [true] but found [false]". Below the error, a "Steps" section lists the test execution process: "[13:48:08] Test started", "[13:48:09] Open doodles page", "[13:48:12] Get archive doodles amount", "[13:48:12] Load new archive doodles", "[13:48:12] Get archive doodles amount", and "[13:48:13] Test finished with status: FAILED". At the bottom of the report, there are links for "Page screenshot" (245.5 KB) and "Page source" (200.4 KB). The page screenshot on the right shows the Google homepage with a "GOOGLE" logo featuring the Austrian flag and a sidebar with news about the National Day of Austria in 2017.