

**TEST AUTOMATION**

PRACTICAL GUIDE for SPEAKER

UI ◦ SERVICE ◦ PERFORMANCE   
TEST CASES

Vasilis Petrou ◦ 2019

Contents

[**1. Analysis of the Application Under Test (AUT)** 4](#_Toc26979939)

[**2. UI Testing with Java and Selenium WebDriver** 5](#_Toc26979940)

[**2.1 Basic Implementation** 5](#_Toc26979941)

[2.1.1 Create New Contact 10](#_Toc26979942)

[2.1.2 Create New Contact Only Mandatory Fields 13](#_Toc26979943)

[2.1.4 Search and View Contact 14](#_Toc26979944)

[2.1.4 Search and Edit Contact 15](#_Toc26979945)

[2.1.5 Delete Contacts 18](#_Toc26979946)

[**2.2 Packaging & Refactoring** 19](#_Toc26979947)

[2.2.1 Packaging and Reporting 19](#_Toc26979948)

[2.2.2 Using PAGE OBJECT MODEL 23](#_Toc26979949)

[2.2.3 Page Factory Class with Lazy Instantiation 28](#_Toc26979950)

[2.2.4 Test Data Custom Provider 29](#_Toc26979951)

[2.2.5 Update BaseTest super class 29](#_Toc26979952)

[2.2.6 Update Test Cases with POM 31](#_Toc26979953)

[**3. Web Service Testing with SoapUI** 34](#_Toc26979954)

[**3.1 Basic Implementation** 34](#_Toc26979955)

[**3.2 Packaging & Refactoring** 40](#_Toc26979956)

[**4. Performance Testing with JMeter** 47](#_Toc26979957)

[**4.1 Basic Implementation** 47](#_Toc26979958)

[**4.2 Packaging & Refactoring** 49](#_Toc26979959)

# **1. Analysis of the Application Under Test (AUT)**

**GIT REPO**: <https://github.com/Intrasoft-International/STATION-8>

**GIT Folder**: demo-app

**Content**: CRUD Application to create/read/update/delete contacts.

**Technologies**: Java, Spring Boot, Angular 6 with PrimeNG and Material Components.

**Scope:** Test Automation Practice

**Installation**:

1. Download & Install Docker
2. Open CMD and run commands from demo-app folder:

docker-compose build

docker-compose down

1. To stop demo-app just click CTRL + C or run command:

docker-compose down

**Web Application URL:** <http://localhost:7001/>

**Swagger URL (API Documentation):** <http://localhost:7001/swagger-ui.html>

# **2. UI Testing with Java and Selenium WebDriver**

## **2.1 Basic Implementation**

1. **Download IntelliJ IDEA CE** <https://www.jetbrains.com/idea/download/>
2. **Download JDK 8** <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
3. Describe the Scenarios to be implemented
   1. Create Contact with all the fields
   2. Create Contact with only mandatory fields
   3. Search and View Contact
   4. Search and Edit Contact
   5. Delete Contacts
4. Create a new folder e.g. C:\demo-test\GuiTests
5. Open IntelliJ > Create New Project > Maven > 1.8 Java

GroupId > com.intrasoft.codinghive

ArtifactId > demo-ui-test

Version > 1.0.0

1. Add in pom.xml below dependencies:

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.4.0</version>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.9.10</version>

<scope>compile</scope>

</dependency>

</dependencies>

1. Delete “test” package and keep only “main”. The reason that we use “main” package is to create a standalone test runner.
2. Download “chromedriver” from <http://chromedriver.chromium.org/downloads> and copy it in “resources” folder.
3. Create a new series of packages under src/main/java

/com/intrasoft/codinghive/test\_cases/TS\_CH\_UI

1. Create a new test case class under src/main/java/com/intrasoft/codinghive/test\_cases/TS\_CH\_UI named TC\_CH\_UI\_001\_CreateNewContact
2. Create a simple command to use the TestNG

@Test

public void TC\_LA\_UI\_001\_Test\_Steps() {

Assert.assertEquals(10, 10);

}

1. Rename the test class and start implementing test cases under a package “test\_cases” within src/main/java/com/intrasoft/stsc/test\_cases/TS\_CH\_UI :

* TC\_CH\_UI\_001\_CreateNewContact
* TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory
* TC\_CH\_UI\_003\_SearchAndViewContact
* TC\_CH\_UI\_004\_SearchAndEditContact
* TC\_CH\_UI\_005\_DeleteContacts

1. Create “test\_suites” folder under “resources” package and an XML file inside it “TS\_CH\_UI.xml” with content:

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >  
<suite name="TS\_CH\_UI" verbose="1">  
 <test name="TC\_CH\_UI\_001\_CreateNewContact">  
 <classes>  
 <class name="com.intrasoft.codinghive.test\_cases.TS\_CH\_UI.TC\_CH\_UI\_001\_CreateNewContact"></class>  
 </classes>  
 </test>  
 <test name="TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory">  
 <classes>  
 <class name="com.intrasoft.codinghive.test\_cases.TS\_CH\_UI.TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory"></class>  
 </classes>  
 </test>  
 <test name="TC\_CH\_UI\_003\_SearchAndViewContact">  
 <classes>  
 <class name="com.intrasoft.codinghive.test\_cases.TS\_CH\_UI.TC\_CH\_UI\_003\_SearchAndViewContact"></class>  
 </classes>  
 </test>  
 <test name="TC\_CH\_UI\_004\_SearchAndEditContact">  
 <classes>  
 <class name="com.intrasoft.codinghive.test\_cases.TS\_CH\_UI.TC\_CH\_UI\_004\_SearchAndEditContact"></class>  
 </classes>  
 </test>  
 <test name="TC\_CH\_UI\_005\_DeleteContacts">  
 <classes>  
 <class name="com.intrasoft.codinghive.test\_cases.TS\_CH\_UI.TC\_CH\_UI\_005\_DeleteContacts"></class>  
 </classes>  
 </test>  
</suite>

**Test Cases**

In the beginning of each test case we must open browser and navigate to the application under test.

At the end of each test case, browser must be closed.

So, for the 1st Test Case we will put below content:

//OPEN BROWSER AND APPLICATION URL  
System.*setProperty*("webdriver.chrome.driver", "C:/demo-test/GuiTests/src/main/resources/chromedriver.exe");  
  
//create a new chrome driver  
WebDriver driver = new ChromeDriver();  
  
// navigate to web application  
driver.get("http://localhost:7001");  
  
// Verify that H3 element contains text "Contact List"  
Assert.*assertEquals*(driver.findElement(By.*xpath*("//h3")).getText(), "Contact List");  
  
//CLOSE BROWSER  
driver.quit();

As the 2 parts are repeated before and after each test case, we can use annotation methods @BeforeTest and @AfterTest

In order to avoid this repetition we will create a super class and each test case class will extend this super class.

WebDriver driver;  
  
@BeforeTest  
public void setup(){  
 //OPEN BROWSER AND APPLICATION URL  
 System.*setProperty*("webdriver.chrome.driver", "D:\\Projects\\coding-hive\\3.gui-test-v1\\src\\main\\resources\\chromedriver.exe");  
  
 //create a new chrome driver  
 driver = new ChromeDriver();  
  
 // navigate to web application  
 driver.get("http://localhost:7001");  
}  
  
@Test  
public void test(){  
 // Verify that H3 element contains text "Contact List"  
 Assert.*assertEquals*(driver.findElement(By.*xpath*("//h3")).getText(), "Contact List");  
}  
  
@AfterTest  
public void tearDown(){  
 //CLOSE BROWSER  
 driver.quit();  
}

As BeforeTest and AfterTest will be used by each test case, there is no reason to repeat it every time. So we will create a new package with name test\_utils under /src/main/java/com/Intrasoft/codinghive

In test\_utils package we will create a new class named BaseTest

We will copy paste the BeforeTest and AfterTest methods

public class BaseTest {  
  
 public static WebDriver driver;

public static WebDriverWait wait;  
  
 @BeforeTest  
 public void setUp() {  
 //OPEN BROWSER AND APPLICATION URL  
 System.*setProperty*("webdriver.chrome.driver", "D:\\Projects\\coding-hive\\3.gui-test-v1\\src\\main\\resources\\chromedriver.exe");  
  
 //create a new chrome driver  
 driver = new ChromeDriver();  
  
 // navigate to web application  
 driver.get("http://localhost:7001");

// define a global wait variable (e.g. 10 seconds until test fail)

wait = new WebDriverWait(driver, 10);

//maximize window  
 //driver.manage().window().maximize();  
 }  
  
 @AfterTest  
 public void tearDown() {  
 driver.quit();  
 }

And we will just extend the BaseTest to the Test Case removing the BeforeTest and AfterTest methods.



To avoid hardcoded paths we will create a method in BaseTest to get the current path of the resources folder (relative path). We must consider one thing. Ideally, when we will package the UI Test Automation Projects, tests will be executed via a JAR file. So the path of resources folder will be different.

public static String getResourcesPath() {  
 String filePathString = *getAbsolutePath*() + "/src/main/resources";  
 File f = new File(filePathString);  
 if (!f.exists())  
 filePathString = *getAbsolutePath*();  
 return filePathString;  
}  
  
private static String getAbsolutePath() {  
 String absPath = Paths.*get*(".").toAbsolutePath().normalize().toString();  
 return absPath.replace("\\", "/");  
}

For that reason, we check if there is indeed src/main/resources folder or not to dynamically return the resources path.

So, BeforeTest method is updated accordingly calling method getResourcesPath()

//OPEN BROWSER AND APPLICATION URL  
System.*setProperty*("webdriver.chrome.driver", *getResourcesPath*() + "/chromedriver.exe");

There are 2 more methods that are commonly used, so we will include them in BaseTest method:

*/\*\*  
 \* This is method is used for dynamic waits until the page is loaded  
 \*/*public void waitForLoad() {  
 ExpectedCondition<Boolean> expectation = new  
 ExpectedCondition<Boolean>() {  
 public Boolean apply(WebDriver driver) {  
 return ((JavascriptExecutor) driver).executeScript("return document.readyState").toString().equals("complete");  
 }  
 };  
 try {  
 Thread.*sleep*(500);  
 WebDriverWait wait = new WebDriverWait(driver, 30);  
 wait.until(expectation);  
 } catch (Throwable error) {  
 Assert.*fail*("Timeout waiting for Page Load Request to complete.");  
 }  
}

*/\*\*  
 \* This is a method that return if an element exists or not  
 \*  
 \** ***@param*** *by is the element locator  
 \** ***@return*** *true when an element exists, false for the opposite  
 \*/*public Boolean isElementPresent(By by) {  
 if (driver.findElements(by).size() > 0)  
 return true;  
 else  
 return false;  
}

Let’s start now the implementation of the Test Cases

### 2.1.1 Create New Contact

public class TC\_CH\_UI\_001\_CreateNewContact extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_001\_CreateNewContact() {  
 // select from menu Button > Add New Contact  
 driver.findElement(By.*id*("newContact")).click();  
  
 // wait for page to load  
 waitForLoad();  
  
 //common input fields  
 driver.findElement(By.*name*("name")).sendKeys("Contact 001");  
 driver.findElement(By.*name*("address")).sendKeys("Address 001");  
 driver.findElement(By.*name*("phone")).sendKeys("2101111111");  
 driver.findElement(By.*name*("email")).sendKeys("email001@intrasoft-intl.com");  
  
 // DROPDOWN  
 // not working for p-dropdown  
 //new Select(driver.findElement(By.name("city"))).selectByVisibleText("Athens");  
  
 // not working for p-dropdown  
 //driver.findElement(By.name("city")).click();  
 //new Select(driver.findElement(By.name("city"))).selectByVisibleText("Athens");  
 driver.findElement(By.*name*("city")).click();  
 wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*xpath*("//li[contains(@class,'ui-dropdown-item')]")));  
 driver.findElement(By.*xpath*("//li[contains(@class,'ui-dropdown-item') and contains(.,'Athens')]")).click();  
  
 // CHECKBOX  
 //label[text()='Disabled']/preceding-sibling::input[@type='checkbox']  
  
 // common click() is not working for angular primeng checkbox  
 //driver.findElement(By.xpath("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']")).click();  
  
 // with ACTIONS is not working. no error but no clicking too  
 //WebElement chbox = driver.findElement(By.xpath("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']"));  
 //Actions action = new Actions(driver);  
 //action.click(chbox).perform();  
  
 // with JAVASCRIPT EXECUTOR works properly  
  
// System.out.println("VALUE BEFORE CLICKING:" + driver  
// .findElement(By.xpath("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']"))  
// .getAttribute("value"));  
// WebElement chbox = driver.findElement(By.xpath("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']"));  
// ((JavascriptExecutor) driver).executeScript("arguments[0].click();", chbox);  
// System.out.println("VALUE AFTER CLICKING:" + driver  
// .findElement(By.xpath("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']"))  
// .getAttribute("value"));  
  
// if (driver.findElement(By.xpath("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']")).isSelected()) {  
// System.out.println("SELECTED");  
// } else {  
// System.out.println("NOT SELECTED");  
// }  
// ((JavascriptExecutor) driver).executeScript("arguments[0].click();", driver.findElement(By.xpath("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']")));  
// if (driver.findElement(By.xpath("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']")).isSelected()) {  
// System.out.println("SELECTED");  
// } else {  
// System.out.println("NOT SELECTED");  
// }  
  
 // WHEN the desired option is to ENABLE the contact  
 // IF the checkbox is unchecked  
 // no action  
 // ELSE  
 // check the checkbox  
 // WHEN the desired option is to DISABLE the contact  
 // IF the checkbox is unchecked  
 // check the checkbox  
 // ELSE  
 // no action  
  
 WebElement chbox;  
 String status = "enabled"; // enabled or disabled  
 if (status.equalsIgnoreCase("enabled")) {  
 chbox = driver.findElement(By.*xpath*("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']"));  
 if (chbox.isSelected()) {  
 ((JavascriptExecutor) driver).executeScript("arguments[0].click();", chbox);  
 }  
 } else {  
 chbox = driver.findElement(By.*xpath*("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']"));  
 if (!chbox.isSelected()) {  
 ((JavascriptExecutor) driver).executeScript("arguments[0].click();", chbox);  
 }  
 }  
  
 // RADIOBUTTON  
// //System.out.println("Selected Gender:" + driver.findElement(By.xpath("//p-radiobutton[@value='f']")).isSelected());  
//  
// if (isElementPresent(By.xpath("//p-radiobutton[@value='f']//\*[contains(@class,'ui-state-active')]"))){  
// System.out.println("Selected Gender: female");  
// } else {  
// if (isElementPresent(By.xpath("//p-radiobutton[@value='m']//\*[contains(@class,'ui-state-active')]"))) {  
// System.out.println("Selected Gender: male");  
// }  
// }  
//  
// //driver.findElement(By.xpath("//p-radiobutton[@value='m']")).click();  
// driver.findElement(By.xpath("//p-radiobutton[@value='m']//\*[contains(@class,'ui-radiobutton-box')]")).click();  
//  
// if (isElementPresent(By.xpath("//p-radiobutton[@value='f']//\*[contains(@class,'ui-state-active')]"))){  
// System.out.println("Selected Gender: female");  
// } else {  
// if (isElementPresent(By.xpath("//p-radiobutton[@value='m']//\*[contains(@class,'ui-state-active')]"))) {  
// System.out.println("Selected Gender: male");  
// }  
// }  
//  
// WHEN the desired option for gender is FEMALE  
// IF the selected radiobutton is FEMALE  
// THEN no action  
// ELSE  
// IF the selected rariobutton is MALE  
// THEN select FEMALE  
  
// WHEN the desired option for gender is MALE  
// IF the selected radiobutton is FEMALE  
// THEN select MALE  
// ELSE  
// IF the selected rariobutton is MALE  
// THEN no action  
  
 WebElement radiobutton;  
 String gender = "female"; // male or female  
 if(gender.equalsIgnoreCase("female")) {  
 if (isElementPresent(By.*xpath*("//p-radiobutton[@value='m']//\*[contains(@class,'ui-state-active')]"))){  
 driver.findElement(By.*xpath*("//p-radiobutton[@value='f']//\*[contains(@class,'ui-radiobutton-box')]")).click();  
 }  
 } else {  
 if (isElementPresent(By.*xpath*("//p-radiobutton[@value='f']//\*[contains(@class,'ui-state-active')]"))){  
 driver.findElement(By.*xpath*("//p-radiobutton[@value='m']//\*[contains(@class,'ui-radiobutton-box')]")).click();  
 }  
 }  
  
 // BUTTON  
 driver.findElement(By.*xpath*("//button[contains(text(),'Save')]")).click();  
  
 //wait for page to load after successful creation of contact  
 waitForLoad();  
  
 //verify auto re-direction to Contact List Page  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//h3[contains(.,'Contact List')]")));  
  
 //verify that table contains contact 'Contact 001'  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//table[contains(.,'Contact 001')]")));  
  
 }

### 2.1.2 Create New Contact Only Mandatory Fields

public class TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory() {  
 // select from menu Button > Add New Contact  
 driver.findElement(By.*id*("newContact")).click();  
  
 // wait for page to load  
 waitForLoad();  
  
 //only mandatory fields  
 driver.findElement(By.*name*("name")).sendKeys("Contact 002");  
 driver.findElement(By.*name*("email")).sendKeys("email002@intrasoft-intl.com");  
  
 driver.findElement(By.*name*("city")).click();  
 wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*xpath*("//li[contains(@class,'ui-dropdown-item')]")));  
 driver.findElement(By.*xpath*("//li[contains(@class,'ui-dropdown-item') and contains(.,'Thessaloniki')]")).click();  
  
 driver.findElement(By.*xpath*("//button[contains(text(),'Save')]")).click();  
  
 //wait for page to load after successful creation of contact  
 waitForLoad();  
  
 //verify auto re-direction to Contact List Page  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//h3[contains(.,'Contact List')]")));  
  
 //verify that table contains contact 'Contact 002'  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//table[contains(.,'Contact 002')]")));  
  
 }

### 2.1.4 Search and View Contact

public class TC\_CH\_UI\_003\_SearchAndViewContact extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_003\_SearchAndViewContact() {  
 // Global Filter filtering name  
 driver.findElement(By.*xpath*("//input[@placeholder='Global Filter']")).sendKeys("Contact 001");  
 waitForLoad();  
  
 //click on Details  
 driver.findElement(By.*xpath*("//tbody[contains(.,'Contact 001')]//a[text()='Details']")).click();  
 waitForLoad();  
  
 //Verify TITLE  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//h3[contains(.,'Contact 001')]")));  
 //alternatively  
 Assert.*assertTrue*(driver.findElement(By.*xpath*("//h3")).getText().equals("Contact 001"));  
  
 //Verify address  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dd[contains(.,'Address 001')]")));  
 // verify including label to verify that values are following the correct labels  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Address')]/following-sibling::dd[contains(.,'Address 001')]")));  
  
 //Verify city  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'City')]/following-sibling::dd[contains(.,'Athens')]")));  
  
 //Verify phone  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Phone')]/following-sibling::dd[contains(.,'2101111111')]")));  
  
 //Verify email  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Email')]/following-sibling::dd[contains(.,'email001@intrasoft-intl.com')]")));  
  
 //Verify gender  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Gender')]/following-sibling::dd[contains(.,'Female')]")));  
  
 //Verify status  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Status')]/following-sibling::dd[contains(.,'enabled')]")));  
  
 //verify buttons  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//a[text()='Edit']")));  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//a[text()='Back']")));  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//button[contains(.,'Delete')]")));  
  
 // go Back  
 driver.findElement(By.*xpath*("//a[text()='Back']")).click();  
 waitForLoad();  
  
 // verify go Back works properly  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//h3[contains(.,'Contact List')]")));  
 }  
  
}

### 2.1.4 Search and Edit Contact

public class TC\_CH\_UI\_004\_SearchAndEditContact extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_004\_SearchAndEditContact() {  
 // Global Filter filtering name  
 driver.findElement(By.*xpath*("//input[@placeholder='Global Filter']")).sendKeys("Contact 001");  
 waitForLoad();  
  
 //click on Details  
 driver.findElement(By.*xpath*("//tbody[contains(.,'Contact 001')]//a[text()='Details']")).click();  
 waitForLoad();  
  
 // click EDIT  
 driver.findElement(By.*xpath*("//a[text()='Edit']")).click();  
 waitForLoad();  
  
 // verify that EDIT CONTACT page opens  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//h3[contains(.,'Edit Contact')]")));  
  
 //UPDATE ALL FIELDS (clear and insert new value)  
 driver.findElement(By.*name*("name")).clear();  
 driver.findElement(By.*name*("name")).sendKeys("Contact 001a");  
  
 driver.findElement(By.*name*("address")).clear();  
 driver.findElement(By.*name*("address")).sendKeys("Address 001a");  
  
 driver.findElement(By.*name*("phone")).clear();  
 driver.findElement(By.*name*("phone")).sendKeys("2101111111a");  
  
 driver.findElement(By.*name*("email")).clear();  
 driver.findElement(By.*name*("email")).sendKeys("email001a@intrasoft-intl.com");  
  
 driver.findElement(By.*name*("city")).click();  
 wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*xpath*("//li[contains(@class,'ui-dropdown-item')]")));  
 driver.findElement(By.*xpath*("//li[contains(@class,'ui-dropdown-item') and contains(.,'Thessaloniki')]")).click();  
  
 // CHECKBOX  
 WebElement chbox;  
 String status = "disabled"; // enabled or disabled  
 if (status.equalsIgnoreCase("enabled")) {  
 chbox = driver.findElement(By.*xpath*("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']"));  
 if (chbox.isSelected()) {  
 ((JavascriptExecutor) driver).executeScript("arguments[0].click();", chbox);  
 }  
 } else {  
 chbox = driver.findElement(By.*xpath*("//label[text()='Disabled']/preceding-sibling::input[@type='checkbox']"));  
 if (!chbox.isSelected()) {  
 ((JavascriptExecutor) driver).executeScript("arguments[0].click();", chbox);  
 }  
 }  
  
 // RADIOBUTTON  
 String gender = "male"; // male or female  
 if(gender.equalsIgnoreCase("female")) {  
 if (isElementPresent(By.*xpath*("//p-radiobutton[@value='m']//\*[contains(@class,'ui-state-active')]"))){  
 driver.findElement(By.*xpath*("//p-radiobutton[@value='f']//\*[contains(@class,'ui-radiobutton-box')]")).click();  
 }  
 } else {  
 if (isElementPresent(By.*xpath*("//p-radiobutton[@value='f']//\*[contains(@class,'ui-state-active')]"))){  
 driver.findElement(By.*xpath*("//p-radiobutton[@value='m']//\*[contains(@class,'ui-radiobutton-box')]")).click();  
 }  
 }  
  
 // BUTTON  
 driver.findElement(By.*xpath*("//button[contains(text(),'Update')]")).click();  
 waitForLoad();  
  
 // VERIFY CHANGES  
 //Verify TITLE  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//h3[contains(.,'Contact 001a')]")));  
  
 //Verify address  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Address')]/following-sibling::dd[contains(.,'Address 001a')]")));  
  
 //Verify city  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'City')]/following-sibling::dd[contains(.,'Thessaloniki')]")));  
  
 //Verify phone  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Phone')]/following-sibling::dd[contains(.,'2101111111a')]")));  
  
 //Verify email  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Email')]/following-sibling::dd[contains(.,'email001a@intrasoft-intl.com')]")));  
  
 //Verify gender  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Gender')]/following-sibling::dd[contains(.,'Male')]")));  
  
 //Verify status  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Status')]/following-sibling::dd[contains(.,'disabled')]")));  
  
 }  
  
}

### 2.1.5 Delete Contacts

public class TC\_CH\_UI\_005\_DeleteContacts extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_005\_DeleteContacts() {  
 // Global Filter filtering name  
 driver.findElement(By.*xpath*("//input[@placeholder='Global Filter']")).sendKeys("Contact 001a");  
 waitForLoad();  
  
 //click on Details  
 driver.findElement(By.*xpath*("//tbody[contains(.,'Contact 001a')]//a[text()='Details']")).click();  
 waitForLoad();  
  
 // Delete Contact  
 driver.findElement(By.*xpath*("//button[contains(.,'Delete')]")).click();  
 driver.switchTo().alert().accept();  
 driver.switchTo().defaultContent();  
 waitForLoad();  
  
 // Repeat Delete Action for rest Contacts  
 driver.findElement(By.*xpath*("//input[@placeholder='Global Filter']")).sendKeys("Contact 002");  
 waitForLoad();  
 driver.findElement(By.*xpath*("//tbody[contains(.,'Contact 002')]//a[text()='Details']")).click();  
 waitForLoad();  
 driver.findElement(By.*xpath*("//button[contains(.,'Delete')]")).click();  
 driver.switchTo().alert().accept();  
 driver.switchTo().defaultContent();  
 waitForLoad();  
  
 // VERIFY THAT TABLE IS EMPTY AFTER DELETING ALL CONTACTS  
 int numberOfRows = driver.findElements(By.*xpath*("//table//tbody[@class='ui-table-tbody']//tr")).size();  
 Assert.*assertEquals*(numberOfRows, 0);  
 }

## **2.2 Packaging & Refactoring**

### 2.2.1 Packaging and Reporting

To be able to deliver the Test Automation Project and the tests to be executed externally (not via IntelliJ) we will create a Main method that will execute the TS\_CH\_UI XML test suite (all the test cases).

We will also integrate our tests with ReportNG to have a report with results

1. Update TS\_CH\_UI with a listener. Put it before the test elements

<listeners>  
 <listener class-name="org.uncommons.reportng.HTMLReporter"/>  
 <listener class-name="org.uncommons.reportng.JUnitXMLReporter"/>  
</listeners>

1. Update pom.xml with below content

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.intrasoft.codinghive</groupId>  
 <artifactId>demo-test</artifactId>  
 <version>1.0.0</version>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <!--Maven plugins-->  
 <maven-compiler-plugin.version>3.8.1</maven-compiler-plugin.version>  
 <maven-resources-plugin.version>2.6</maven-resources-plugin.version>  
 <maven-clean-plugin.version>3.0.0</maven-clean-plugin.version>  
 <!--Testing-->  
 <selenium.version>3.14.0</selenium.version>  
 <testng.version>6.9.10</testng.version>  
 <reportng.version>1.1.4</reportng.version>  
 <velocity.version>1.4</velocity.version>  
 <juicy.version>3.0</juicy.version>  
 </properties>  
  
 <repositories>  
 <repository>  
 <id>jcenter</id>  
 <url>https://dl.bintray.com/cbeust/maven/</url>  
 </repository>  
 </repositories>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>${maven-compiler-plugin.version}</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 <plugin>  
 <artifactId>maven-assembly-plugin</artifactId>  
 <executions>  
 <execution>  
 <phase>package</phase>  
 <goals>  
 <goal>single</goal>  
 </goals>  
 </execution>  
 </executions>  
 <configuration>  
 <archive>  
 <manifest>  
 <mainClass>com.intrasoft.codinghive.Main</mainClass>  
 </manifest>  
 </archive>  
 <descriptorRefs>  
 <descriptorRef>jar-with-dependencies</descriptorRef>  
 </descriptorRefs>  
 <appendAssemblyId>false</appendAssemblyId>  
 </configuration>  
 </plugin>  
 <plugin>  
 <artifactId>maven-resources-plugin</artifactId>  
 <version>${maven-resources-plugin.version}</version>  
 <executions>  
 <execution>  
 <id>copy-resources</id>  
 <phase>validate</phase>  
 <goals>  
 <goal>copy-resources</goal>  
 </goals>  
 <configuration>  
 <outputDirectory>${project.build.directory}</outputDirectory>  
 <resources>  
 <resource>  
 <directory>src/main/resources</directory>  
 </resource>  
 </resources>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 <plugin>  
 <artifactId>maven-clean-plugin</artifactId>  
 <version>${maven-clean-plugin.version}</version>  
 <executions>  
 <execution>  
 <id>auto-clean</id>  
 <phase>install</phase>  
 <goals>  
 <goal>clean</goal>  
 </goals>  
 <configuration>  
 <excludeDefaultDirectories>true</excludeDefaultDirectories>  
 <filesets>  
 <fileset>  
 <directory>${basedir}/target/archive-tmp</directory>  
 </fileset>  
 <fileset>  
 <directory>${basedir}/target/classes</directory>  
 </fileset>  
 <fileset>  
 <directory>${basedir}/target/generated-sources</directory>  
 </fileset>  
 <fileset>  
 <directory>${basedir}/target/maven-archiver</directory>  
 </fileset>  
 <fileset>  
 <directory>${basedir}/target/surefire</directory>  
 </fileset>  
 <fileset>  
 <directory>${basedir}/target/maven-status</directory>  
 </fileset>  
 </filesets>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
 </build>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.seleniumhq.selenium</groupId>  
 <artifactId>selenium-java</artifactId>  
 <version>${selenium.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.testng</groupId>  
 <artifactId>testng</artifactId>  
 <version>${testng.version}</version>  
 <scope>compile</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.uncommons</groupId>  
 <artifactId>reportng</artifactId>  
 <version>${reportng.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>velocity</groupId>  
 <artifactId>velocity-dep</artifactId>  
 <version>${velocity.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>com.google.inject</groupId>  
 <artifactId>guice</artifactId>  
 <version>${juicy.version}</version>  
 </dependency>  
 </dependencies>  
  
</project>

1. Create a Main class in path src/main/java/com/Intrasoft/codinghive with below content

public class Main {  
  
 public static void main(String[] args) {  
 System.*out*.println("Test Suite: TS\_CH\_UI");  
 TestListenerAdapter tla = new TestListenerAdapter();  
 TestNG testng = new TestNG();  
 List<String> suites = Lists.*newArrayList*();  
  
 suites.add("test\_suites/TS\_CH\_UI.xml");  
 String timeStamp = *getTimestamp*();  
 new File("reports/TS\_CH\_UI\_" + timeStamp).mkdirs();  
 testng.setOutputDirectory("reports/TS\_CH\_UI\_" + timeStamp);  
 testng.setTestSuites(suites);  
 testng.addListener(tla);  
 testng.run();  
 }  
  
 private static String getTimestamp() {  
 String timeStamp = new SimpleDateFormat("yyyy-MM-dd\_HH-mm-ss").format(new Date());  
 return timeStamp;  
 }  
}

1. In resources folder insert a SHELL SCRIPT with name **launch-test.sh** and below content

java -jar \*.jar

### 2.2.2 Using PAGE OBJECT MODEL

**The idea is**

**Every PAGE is a CLASS with its methods. We just call methods from each test case, and when something changes, the TCs are not affected. Only the Class (object) that relates with this page.**

**New Package > web\_pages**

* + **Contact**
  + **ContactList**
  + **Menu**

public class Menu extends BaseTest {  
  
 @FindBy(id = "newContact")  
 WebElement addNewContactBtn;  
 @FindBy(id = "listOfContacts")  
 WebElement contactListBtn;  
  
 public Menu () {  
 initPageFactory();  
 }  
  
 public Menu clickNewContact(){  
 addNewContactBtn.click();  
 waitForLoad();  
 return this;  
 }  
  
 public Menu clickContactList(){  
 contactListBtn.click();  
 waitForLoad();  
 return this;  
 }  
  
}

public class ContactList extends BaseTest {  
  
 @FindBy(xpath = "//input[@placeholder='Global Filter']")  
 WebElement globalFilter;  
  
 private final static String *NAME* = "Name";  
 private final static String *FILTER* = "Filter";  
  
 public ContactList() {  
 initPageFactory();  
 }  
  
 public ContactList selectContact(Properties testData) {  
 insertGlobalFilter(testData);  
 clickDetails(testData);  
 return this;  
 }  
  
 public ContactList verifyContactListPageOpens() {  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//h3[contains(.,'Contact List')]")));  
 return this;  
 }  
  
 public ContactList verifyContactListContent(Properties testData) {  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//table[contains(.,'" + testData.getProperty(*NAME*) + "')]")));  
 return this;  
 }  
  
 public ContactList verifyEmptyTable() {  
 int numberOfRows = *driver*.findElements(By.*xpath*("//table//tbody[@class='ui-table-tbody']//tr")).size();  
 Assert.*assertEquals*(numberOfRows, 0);  
 return this;  
 }  
  
 private void insertGlobalFilter(Properties testData) {  
 globalFilter.clear();  
 globalFilter.sendKeys(testData.getProperty(*FILTER*));  
 }  
  
 private void clickDetails(Properties testData) {  
 *driver*.findElement(By.*xpath*("//tbody[contains(.,'" + testData.getProperty(*FILTER*) + "')]//a[text()='Details']")).click();  
 waitForLoad();  
 }  
  
}

public class Contact extends BaseTest {  
  
 @FindBy(name = "name")  
 WebElement name;  
 @FindBy(name = "address")  
 WebElement address;  
 @FindBy(name = "city")  
 WebElement city;  
 @FindBy(name = "phone")  
 WebElement phone;  
 @FindBy(name = "email")  
 WebElement email;  
 @FindBy(xpath = "//button[contains(text(),'Save')]")  
 WebElement saveBtn;  
 @FindBy(xpath = "//button[contains(.,'Delete')]")  
 WebElement deleteBtn;  
 @FindBy(xpath = "//a[text()='Back']")  
 WebElement backBtn;  
 @FindBy(xpath = "//a[text()='Edit']")  
 WebElement editBtn;  
 @FindBy(xpath = "//button[contains(text(),'Update')]")  
 WebElement updateBtn;  
  
 private final static String *NAME* = "Name";  
 private final static String *ADDRESS* = "Address";  
 private final static String *CITY* = "City";  
 private final static String *PHONE* = "Phone";  
 private final static String *EMAIL* = "Email";  
 private final static String *GENDER* = "Gender";  
 private final static String *STATUS* = "Status";  
  
 public Contact() {  
 initPageFactory();  
 }  
  
 public Contact inputData(Properties testData) {  
 input(name, testData.getProperty(*NAME*));  
 input(address, testData.getProperty(*ADDRESS*));  
 dropdown(city, testData.getProperty(*CITY*));  
 input(phone, testData.getProperty(*PHONE*));  
 input(email, testData.getProperty(*EMAIL*));  
 checkbox("Disabled", testData.getProperty(*STATUS*));  
 radiobutton(testData.getProperty(*GENDER*));  
 return this;  
 }  
  
 public Contact save() {  
 saveBtn.click();  
 waitForLoad();  
 return this;  
 }  
  
 public Contact delete() {  
 deleteBtn.click();  
 *driver*.switchTo().alert().accept();  
 *driver*.switchTo().defaultContent();  
 waitForLoad();  
 return this;  
 }  
  
 public Contact back() {  
 backBtn.click();  
 waitForLoad();  
 return this;  
 }  
  
 public Contact edit() {  
 editBtn.click();  
 waitForLoad();  
 return this;  
 }  
  
 public Contact update() {  
 updateBtn.click();  
 waitForLoad();  
 return this;  
 }  
  
 public Contact verifyViewMode(Properties testData) {  
 //Verify TITLE  
 Assert.*assertTrue*(*driver*.findElement(By.*xpath*("//h3")).getText().equals(testData.getProperty(*NAME*)));  
  
 //Verify address  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Address')]/following-sibling::dd[contains(.,'" + testData.getProperty(*ADDRESS*) + "')]")));  
  
 //Verify city  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'City')]/following-sibling::dd[contains(.,'" + testData.getProperty(*CITY*) + "')]")));  
  
 //Verify phone  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Phone')]/following-sibling::dd[contains(.,'" + testData.getProperty(*PHONE*) + "')]")));  
  
 //Verify email  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Email')]/following-sibling::dd[contains(.,'" + testData.getProperty(*EMAIL*) + "')]")));  
  
 //Verify gender  
 if (testData.getProperty(*GENDER*).equalsIgnoreCase("female"))  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Gender')]/following-sibling::dd[contains(.,'Female')]")));  
 else  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Gender')]/following-sibling::dd[contains(.,'Male')]")));  
  
 //Verify status  
 if (testData.getProperty(*STATUS*).equalsIgnoreCase("enabled"))  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//dt[contains(.,'Status')]/following-sibling::dd[contains(.,'enabled')]")));  
 else  
 Assert.*assertFalse*(isElementPresent(By.*xpath*("//dt[contains(.,'Status')]/following-sibling::dd[contains(.,'enabled')]")));  
  
 //verify buttons  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//a[text()='Edit']")));  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//a[text()='Back']")));  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//button[contains(.,'Delete')]")));  
  
 return this;  
 }  
  
 public Contact verifyEditMode() {  
 Assert.*assertTrue*(isElementPresent(By.*xpath*("//h3[contains(.,'Edit Contact')]")));  
 return this;  
 }  
  
 private void input(WebElement input, String value) {  
 if (value != null && !value.isEmpty()) {  
 input.clear();  
 input.sendKeys(value);  
 }  
 }  
  
 private void dropdown(WebElement dropdown, String value) {  
 if (value != null && !value.isEmpty()) {  
 dropdown.click();  
 wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*xpath*("//li[contains(@class,'ui-dropdown-item')]")));  
 *driver*.findElement(By.*xpath*("//li[contains(@class,'ui-dropdown-item') and contains(.,'" + value + "')]")).click();  
 }  
 }  
  
 private void checkbox(String checkboxLabel, String value) {  
 if (value != null && !value.isEmpty()) {  
 WebElement chbox;  
 if (value.equalsIgnoreCase("enabled")) {  
 chbox = *driver*.findElement(By.*xpath*("//label[text()='" + checkboxLabel + "']/preceding-sibling::input[@type='checkbox']"));  
 if (chbox.isSelected()) {  
 ((JavascriptExecutor) *driver*).executeScript("arguments[0].click();", chbox);  
 }  
 } else {  
 chbox = *driver*.findElement(By.*xpath*("//label[text()='" + checkboxLabel + "']/preceding-sibling::input[@type='checkbox']"));  
 if (!chbox.isSelected()) {  
 ((JavascriptExecutor) *driver*).executeScript("arguments[0].click();", chbox);  
 }  
 }  
 }  
 }  
  
 private void radiobutton(String value) {  
 if (value != null && !value.isEmpty()) {  
 if (value.equalsIgnoreCase("female")) {  
 if (isElementPresent(By.*xpath*("//p-radiobutton[@value='m']//\*[contains(@class,'ui-state-active')]"))) {  
 *driver*.findElement(By.*xpath*("//p-radiobutton[@value='f']//\*[contains(@class,'ui-radiobutton-box')]")).click();  
 }  
 } else {  
 if (isElementPresent(By.*xpath*("//p-radiobutton[@value='f']//\*[contains(@class,'ui-state-active')]"))) {  
 *driver*.findElement(By.*xpath*("//p-radiobutton[@value='m']//\*[contains(@class,'ui-radiobutton-box')]")).click();  
 }  
 }  
 }  
 }  
  
  
}

### 2.2.3 Page Factory Class with Lazy Instantiation

public class Page extends BaseTest {  
  
 private Menu menu;  
 private Contact contact;  
 private ContactList contactList;  
  
 public Menu menu() {  
 synchronized (this) {  
 if (menu == null) {  
 menu = new Menu();  
 }  
 }  
 return menu;  
 }  
  
 public Contact contact() {  
 synchronized (this) {  
 if (contact == null) {  
 contact = new Contact();  
 }  
 }  
 return contact;  
 }  
  
 public ContactList contactList() {  
 synchronized (this) {  
 if (contactList == null) {  
 contactList = new ContactList();  
 }  
 }  
 return contactList;  
 }  
}

### 2.2.4 Test Data Custom Provider

**Under resources create a folder test\_data and a file test\_data.properties**

**For each test case use a prefix the Test Case ID, as suffix a number that refers to the step (if needed) and as a third part the variable that will be mapped with the page objects.**

TC\_CH\_UI\_001\_CreateNewContact.Name=Contact 001  
TC\_CH\_UI\_001\_CreateNewContact.Address=Address 001  
TC\_CH\_UI\_001\_CreateNewContact.City=Athens  
TC\_CH\_UI\_001\_CreateNewContact.Phone=2101111111  
TC\_CH\_UI\_001\_CreateNewContact.Email=email001@intrasoft-intl.com  
TC\_CH\_UI\_001\_CreateNewContact.Status=enabled  
TC\_CH\_UI\_001\_CreateNewContact.Gender=male  
  
TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory.Name=Contact 002  
TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory.City=Thessaloniki  
TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory.Email=email002@intrasoft-intl.com  
  
TC\_CH\_UI\_003\_SearchAndViewContact.1.Filter=Contact 001  
TC\_CH\_UI\_003\_SearchAndViewContact.2.Name=Contact 001  
TC\_CH\_UI\_003\_SearchAndViewContact.2.Address=Address 001  
TC\_CH\_UI\_003\_SearchAndViewContact.2.City=Athens  
TC\_CH\_UI\_003\_SearchAndViewContact.2.Phone=2101111111  
TC\_CH\_UI\_003\_SearchAndViewContact.2.Email=email001@intrasoft-intl.com  
TC\_CH\_UI\_003\_SearchAndViewContact.2.Status=enabled  
TC\_CH\_UI\_003\_SearchAndViewContact.2.Gender=male  
  
TC\_CH\_UI\_004\_SearchAndEditContact.1.Filter=Contact 001  
TC\_CH\_UI\_004\_SearchAndEditContact.2.Name=Contact 001a  
TC\_CH\_UI\_004\_SearchAndEditContact.2.Address=Address 001a  
TC\_CH\_UI\_004\_SearchAndEditContact.2.City=Thessaloniki  
TC\_CH\_UI\_004\_SearchAndEditContact.2.Phone=21022222222  
TC\_CH\_UI\_004\_SearchAndEditContact.2.Email=email001a@intrasoft-intl.com  
TC\_CH\_UI\_004\_SearchAndEditContact.2.Status=disabled  
TC\_CH\_UI\_004\_SearchAndEditContact.2.Gender=female  
  
TC\_CH\_UI\_005\_DeleteContacts.1.Filter=Contact 001  
TC\_CH\_UI\_005\_DeleteContacts.2.Filter=Contact 002

### 2.2.5 Update BaseTest super class

Update BeforeTest and add another static literal Page.

public static WebDriver *driver*;  
public static WebDriverWait *wait*;  
public static Page *page*;  
  
@BeforeTest  
public void setUp() {  
 //OPEN BROWSER AND APPLICATION URL  
 System.*setProperty*("webdriver.chrome.driver", *getResourcesPath*() + "/chromedriver.exe");  
  
 //create a new chrome driver  
 *driver* = new ChromeDriver();  
  
 // navigate to web application  
 *driver*.get("http://localhost:7001");  
  
 // define a global wait variable (e.g. 10 seconds until test fail)  
 *wait* = new WebDriverWait(*driver*, 10);  
  
 *page* = new Page();  
 //maximize window  
 //driver.manage().window().maximize();  
}

Add methods.

*/\*\*  
 \* This method instantiates the Page Object Elements  
 \*/*protected void initPageFactory() {  
 PageFactory.*initElements*(*driver*, this);  
}  
  
*/\*\*  
 \* This method returns a list of properties with prefix the test case ID (no step as suffix)  
 \** ***@return*** *\*/*public Properties getTestData() {  
 return getProperties(  
 loadProperties(*getResourcesPath*() + "/test\_data/test\_data.properties"), getClassName(), 0);  
}  
  
*/\*\*  
 \* This method returns a list of properties with prefix the test case ID and with a suffix the test step number  
 \** ***@return*** *\*/*public Properties getTestData(int step) {  
 return getProperties(  
 loadProperties(*getResourcesPath*() + "/test\_data/test\_data.properties"), getClassName(), step);  
}  
  
*/\*\*  
 \* This method returns the Class Name of the Test Case (Test Case ID)  
 \** ***@return*** *\*/*private String getClassName() {  
 return this.getClass().getSimpleName();  
}  
  
*/\*\*  
 \* This method is used by method 'getTestData' and is used to load properties (test data) from external property files  
 \** ***@param*** *testDataFile is the path of the property test data file  
 \** ***@return*** *\*/*private Properties loadProperties(String testDataFile) {  
 Properties prop = new Properties();  
 try {  
 InputStream inputStream = new FileInputStream(testDataFile);  
 Reader reader = new InputStreamReader(inputStream, "UTF-8");  
 prop.load(reader);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return prop;  
}  
  
*/\*\*  
 \* This method is used from the 'getTestData' method and returns specific subset based on prefix (test case id) and suffix (test step number)  
 \** ***@param*** *params properties as were returned by method "loadProperties"  
 \** ***@param*** *prefix test case id prefix to get subset of properties  
 \** ***@param*** *step test step number suffix to get subset of properties  
 \** ***@return*** *\*/*private Properties getProperties(Properties params, String prefix, int step) {  
 Properties result = new Properties();  
  
 Enumeration<?> names = params.propertyNames();  
  
 if (step > 0) {  
 prefix += "." + step;  
 }  
 while (names.hasMoreElements()) {  
 String name = (String) names.nextElement();  
  
 if (name.indexOf(prefix) == 0) {  
 result.put(name.substring(prefix.length() + 1), params.get(name));  
 }  
 }  
  
 return result;  
}

### 2.2.6 Update Test Cases with POM

public class TC\_CH\_UI\_001\_CreateNewContact extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_001\_CreateNewContact() {  
 *page*.menu()  
 .clickNewContact();  
  
 *page*.contact()  
 .inputData(getTestData())  
 .save();  
  
 *page*.contactList()  
 .verifyContactListPageOpens()  
 .verifyContactListContent(getTestData());  
 }  
  
}

public class TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_002\_CreateNewContactOnlyMandatory() {  
 *page*.menu()  
 .clickNewContact();  
  
 *page*.contact()  
 .inputData(getTestData())  
 .save();  
  
 *page*.contactList()  
 .verifyContactListPageOpens()  
 .verifyContactListContent(getTestData());  
  
 }  
  
}

public class TC\_CH\_UI\_003\_SearchAndViewContact extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_003\_SearchAndViewContact() {  
 *page*.contactList()  
 .selectContact(getTestData(1));  
  
 *page*.contact()  
 .verifyViewMode(getTestData(2))  
 .back();  
  
 *page*.contactList().verifyContactListPageOpens();  
 }  
  
}

public class TC\_CH\_UI\_004\_SearchAndEditContact extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_004\_SearchAndEditContact() {  
 *page*.contactList()  
 .selectContact(getTestData(1));  
  
 *page*.contact()  
 .edit()  
 .verifyEditMode()  
 .inputData(getTestData(2))  
 .update()  
 .verifyViewMode(getTestData(2));  
 }  
  
}

public class TC\_CH\_UI\_005\_DeleteContacts extends BaseTest {  
  
 @Test  
 public void TC\_CH\_UI\_005\_DeleteContacts() {  
 *page*.contactList()  
 .selectContact(getTestData(1));  
 *page*.contact()  
 .delete();  
  
 *page*.contactList()  
 .selectContact(getTestData(2));  
 *page*.contact()  
 .delete();  
  
 *page*.contactList()  
 .verifyEmptyTable();  
 }  
  
}

# **3. Web Service Testing with SoapUI**

## **3.1 Basic Implementation**

1. **Download SoapUI Open Source** <https://www.soapui.org/downloads/soapui.html>
2. **Download also GIT Bash for Windows** <https://gitforwindows.org/>
3. Open SoapUI
4. Describe **GUI Environment of SoapUI**
5. Describe the Scenarios to be implemented
   1. Create Contact
   2. View Contact
   3. Update Contact
   4. Delete Contact
   5. Get a List of all Contacts
6. Select from the menu

**File** > **Create Empty Project**

1. Rename Project’s name

**Project 1** > **ContactREST**

1. Before you start to implement the test-cases **SAVE** the SoapUI Project.

So right click on **ContactREST** and select **Save Project** e.g. **demo-app-soapui-project-v1**

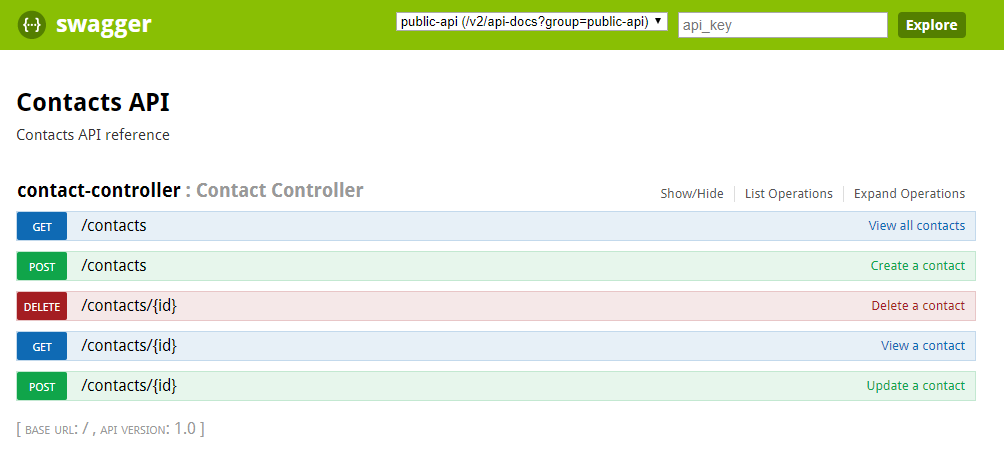
1. And before you continue, **load** the **REST** **ENDPOINTS** to your project.

So right click on **ContactREST** and select **New REST Service from URI**

Do this for all the 5 different REST Endpoints.

So, visit SWAGGER (API DOC of REST Web Services)

<http://localhost:7001/swagger-ui.html>



Click option **List Operations**

Get Contacts GET <http://localhost:7001/contacts>

Create Contact POST <http://localhost:7001/contacts>

Delete Contact DELETE [http://localhost:7001/contacts/{contactID}](http://localhost:7001/contacts/%7bcontactID%7d)

View Contact GET [http://localhost:7001/contacts/{contactID}](http://localhost:7001/contacts/%7bcontactID%7d)

Update Contact POST [http://localhost:7001/contacts/{contactID}](http://localhost:7001/contacts/%7bcontactID%7d)

The 2 POST operations require JSON Body with the Contact.

For the UPDATE Operation we must include in JSON the ID (not only as parameter)

1. Right click on **ContactREST** and select **New TestSuite**
2. Name the suite **TS\_CH\_SRV**
3. Right click on **TS\_CH\_SRV** and select **New TestCase**. Repeat it for each test case
4. Name the test cases:

**TC\_CH\_SRV\_01**

**TC\_CH\_SRV\_02**

**TC\_CH\_SRV\_03**

**TC\_CH\_SRV\_04**

**TC\_CH\_SRV\_05**

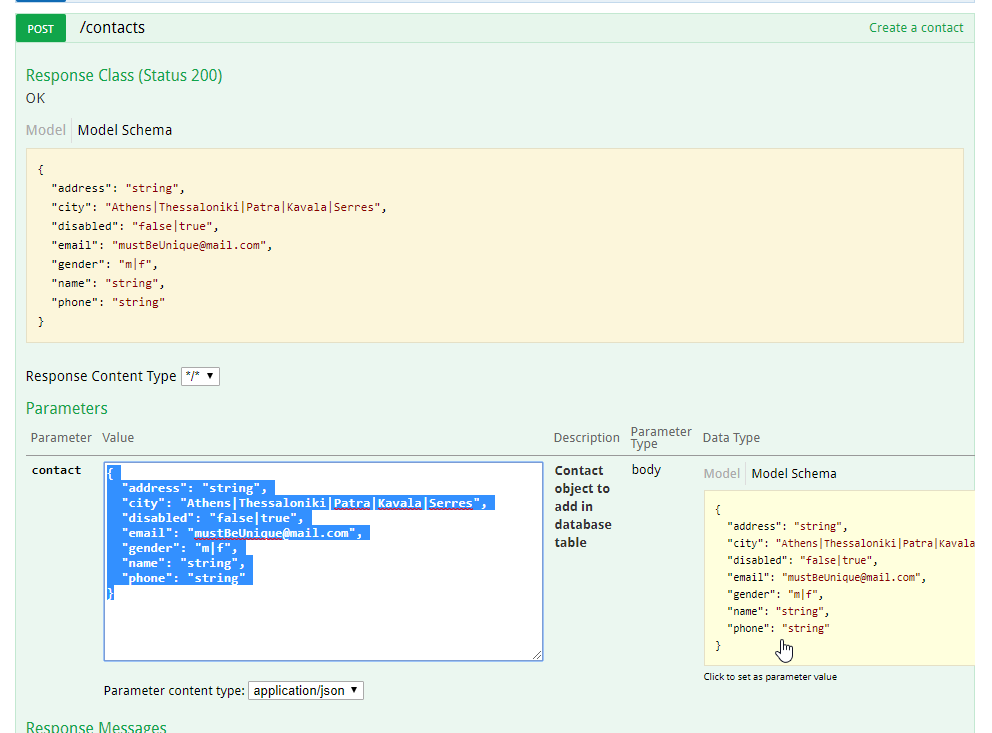
1. Expand the test case **TC\_CH\_SRV\_01** and right click on item **Test Steps** and select

**Add Step** > **REST Request**

1. Specify Name for the new test step: **CreateContact**
2. Select REST method to invoke for request

<http://localhost:7001> -> /contacts -> Create Contact

1. Select **Media Type: application/json**
2. Visit **Swagger** and from the part which help us to get the JSON body, generate one and leave the appropriate values



1. **Run** **from** **Swagger** the REST Request
2. **Display** **in** the **UI** the new contact.
3. **Delete** **from** **UI** the contact
4. **Open** **SoapUI** and **CreateContact** test step and **insert** **JSON** **Body** based on the swagger API DOC

{

"address": "Address 1",

"city": "Athens",

"disabled": "false",

"email": "mail01@mail.com",

"gender": "m",

"name": "Name 1",

"phone": "2101111111"

}

1. Run the test case (run button) and check the area with the response clicking tab JSON



1. As a final step for the test case we must put some assertions (verifications). So, we click on TAB Assertions and button +

Compliance, Status and Standards > Valid HTTP Status Codes > 200

Property Content > Contains > [mail02@mail.com](mailto:mail02@mail.com)

After creating each assertion we can rename their default description to more human readable content, by clicking on assertion and clicking button F2

E.g. the contains can become “Contains New Email”



1. **Display** also **in** the **UI** the new contact.
2. **Delete** **from** **UI** the contact
3. Proceed with the new TC **View Contact.** What we need for this TC is some pre-requisite data. It is significant to have its own contact to view in order to have independency between TCs.
4. First, we create a new test case **TC\_CH\_SRV\_02**
5. Right clicking on step **CreateContact** of test case **TC\_CH\_SRV\_02** we select option **Clone TestStep**

and as **Target Test Case** we select **TC\_CH\_SRV\_02**

1. We update the **JSON Body** with a new **unique** **Email**

{

"address": "Address 2",

"city": "Thessaloniki",

"disabled": "false",

"email": "mail02@mail.com",

"gender": "f",

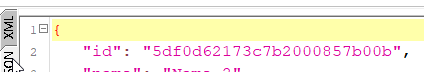
"name": "Name 2",

"phone": "2102222222"

}

1. Execute the test step.

The response area contains the ID which is needed from the VIEW operation.



1. What we need now is a way to **grab** the **value** **of** **id** from the response. For that reason we will use a **GROOVY SCRIPT** step. Right click on TC\_CH\_SRV\_02 and select

Add Step > Groovy Script

1. Specify name **GetNewID**
2. Below content gets the response from the step **CreateContact** and **grabs** the **value** **of** the **ID** element.

import com.eviware.soapui.support.GroovyUtils;

import com.eviware.soapui.support.XmlHolder;

import groovy.json.JsonSlurper;

import groovy.util.\*;

def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context);

// Get target testStep and testCase names of this step

def currentTSName = testRunner.testCase.testSuite.getName();

def currentTCName = testRunner.testCase.getName();

// Define target step

def targetStep = "CreateContact";

//Get the response

def project = testRunner.testCase.testSuite.project;

def tcase = project.testSuites[currentTSName].testCases[currentTCName];

def tstep = tcase.getTestStepByName(targetStep);

def response = tstep.getPropertyValue("response");

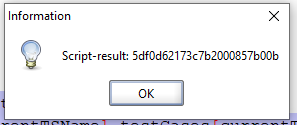
def slurperResponse = new JsonSlurper().parseText(response);

//Get the value of a specific node (e.g.conceptId)

def contactId = slurperResponse.id.toString();

return contactId;

1. **Running** above script, a **popup** **window** must be displayed with the **value** **of** the **ID**



1. Now we need a REST Request step to view the contact we previously created. right click on item **Test Steps** of **TC\_CH\_SRV\_02** test case and select

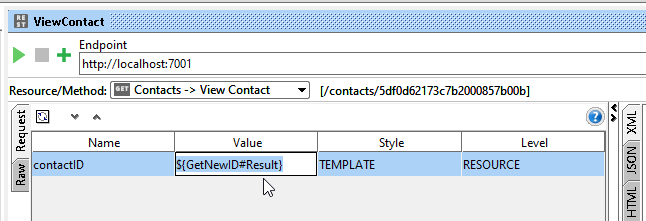
**Add Step** > **REST Request**

1. Specify name **ViewContact**
2. Select **REST method to invoke for request**

<http://localhost:7001> -> /contacts/{contactID} -> View Contact

1. In the Parameters section we must define the value (DYNAMICALLY) as returned by step GetNewID. To do this we can put as value the string:

${GetNewID#Result}



1. Running the step, we will see in the RESPONSE AREA and JSON tab a JSON string with the new Contact.



1. As a final step for the test case we must put some assertions (verifications). So, we click on TAB Assertions and button +

Compliance, Status and Standards > Valid HTTP Status Codes > 200

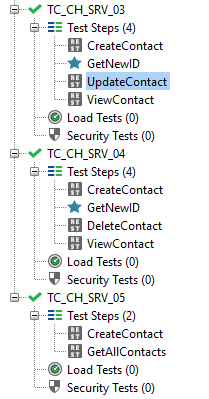
Property Content > Contains > [mail02@mail.com](mailto:mail02@mail.com)

After creating each assertion we can rename their default description to more human readable content, by clicking on assertion and clicking button F2

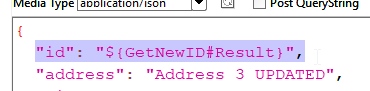
E.g. the contains can become “Contains New Email”



1. Repeat with the same logic the next 3 test cases for the UPDATE/DELETE/GET ALL CONTACTS



The only thing that needs extra notice is that for the Update Contact, the ID is also required in JSON Body (Bad documentation in swagger).



## **3.2 Packaging & Refactoring**

**After implementing** the service test cases there are **2 basic challenges** we must consider:

* The **Packaging**:
  + An organized way to deliver the tests so they can be easily executed from command line and not opening SoapUI.
* The **Refactoring**:
  + Are the test cases correctly implemented **so they can be re-executed** again and again without failures? Do they contain **methods to cleanup/load** all the needed data before each re-execution?
  + Do they contain **external PROPERTY files** with global variables like host/port to be handled externally and not within each test step?

1. First, we will create a **different** **set** of **folders**, so we can integrate the SoapUI tests with maven.

**[srv-test-v2]**

**[src]**

**[test]**

**[resources]**

**[soapui]**

1. We will copy-paste in /src/test/soapui/ the 1st version of SoapUI project we created previously with name

**demo-app-soapui-project-v2**

1. We will create a new file in the same level of **src** folder with name **pom.xml** with below content

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.intrasoft.stsc</groupId>

<artifactId>demo-app-srv-test</artifactId>

<version>1.0.0</version>

<packaging>pom</packaging>

<name>DEMO APP</name>

<properties>

<testSuite></testSuite>

<testCase></testCase>

</properties>

<url>http://maven.apache.org</url>

<pluginRepositories>

<pluginRepository>

<id>SmartBearPluginRepository</id>

<url>http://smartbearsoftware.com/repository/maven2/</url>

</pluginRepository>

</pluginRepositories>

<build>

<plugins>

<plugin>

<groupId>com.smartbear.soapui</groupId>

<artifactId>soapui-maven-plugin</artifactId>

<version>5.5.0</version>

<executions>

<execution>

<phase>test</phase>

<goals>

<goal>test</goal>

</goals>

<configuration>

<projectFile>./src/test/soapui/demo-app-soapui-project-v2.xml</projectFile>

<junitReport>true</junitReport>

<outputFolder>${project.basedir}/target/surefire-reports</outputFolder>

<testSuite>${testSuite}</testSuite>

<testCase>${testCase}</testCase>

</configuration>

</execution>

</executions>

<dependencies>

<dependency>

<groupId>com.jgoodies</groupId>

<artifactId>forms</artifactId>

<version>1.0.7</version>

</dependency>

</dependencies>

</plugin>

</plugins>

</build>

</project>

1. Additionally, in the same folder with the pom.xml, we will create a **SHELL SCRIPT** that runs the tests and generates a report (Integration with surefire report). So, we create a new file with name **launch-test.sh** and below content

# define a variable for the report's folder name using timestamp as suffix

export DIR="DemoAppWS\_TestReport\_"$(date +"%Y%m%d%H%M") ;

# run the tests via maven with additional arguments to generate report

mvn clean install surefire-report:report-only site -DgenerateReports=false;

# generate test-reports folder and a subfolder with the name as defined in DIR variable above

mkdir -p "test-reports/$DIR" ;

# when running with maven a target folder is generated automatically. copy its content to the test-report folder

cp -a target/site/. test-reports/$DIR/ ;

cp -a target/surefire-reports/. test-reports/$DIR/ ;

# generate a logs subfolder and copy there all the log files that are generated by soapui

mkdir -p "test-reports/$DIR/logs";

mv \*.log test-reports/$DIR/logs ;

mv soapui.log.\* test-reports/$DIR/logs

# remove target folder

rm -rf target && rm \*.log && rm soapui.log.\* ;

1. **To run the test and generate a report**, just right click on the folder with the pom.xml and launch-test.sh and select from the menu, option **GIT Bash here**
2. After the execution, a **Report** is generated in path:

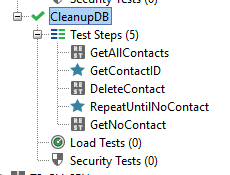
**test-reports/DemoAppWS\_TestReport\_<timestamp>/surefire-report.html**

1. Probably the **report** will **contain** **failures**. These failures relate with the fact that we are **not** **removing** the **contacts** that our **tests** **created** during the first execution. So, we need to create a **method** **that** **removes** all of them, and this method should be **executed** **prior** the execution of the **tests**.
2. So, we open SoapUI, we create a new **TestSuite** with name **BeforeTest** and we create a new Test Case with name **CleanupDB**.

The way, we implement a cleanup relates with what options we have. **There is no right or wrong**. We can run JDBC requests, REST/Soap Requests etc.

In our case we will work with the available REST Services.

**Steps: Get All Contacts, Delete one by one until no contacts.**



1. Step **GetContactID**, gets from **GetAllContacts** response all the **IDs**. **If** the **size** = **0** then it **redirects** to the **last** **step** which verifies that there is no contact

import com.eviware.soapui.support.GroovyUtils;

import com.eviware.soapui.support.XmlHolder;

import groovy.json.JsonSlurper;

import groovy.util.\*;

def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context);

// Get target testStep and testCase names of this step

def currentTSName = testRunner.testCase.testSuite.getName();

def currentTCName = testRunner.testCase.getName();

// Define target step

def targetStep = "GetAllContacts";

//Get the response

def project = testRunner.testCase.testSuite.project;

def tcase = project.testSuites[currentTSName].testCases[currentTCName];

def tstep = tcase.getTestStepByName(targetStep);

def response = tstep.getPropertyValue("response");

def slurperResponse = new JsonSlurper().parseText(response);

//Get the value of a specific node (e.g.conceptId)

def contactId = slurperResponse.id;

if(contactId.size() > 0) {

return contactId[0]

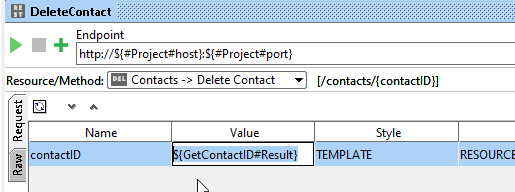
} else {

testRunner.gotoStepByName("GetNoContact")

return null

}

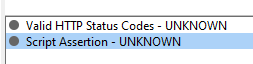
1. REST Request step **DeleteContact**, deletes each **contactID** as returned by groovy script step **GetContactID** (it always grabs the first in the list)



1. Then a step is needed to redirect back to GetAllContacts to re-check the list of contacts. The groovy script step **RepeatUntilNoContact** contains below code:

testRunner.gotoStepByName("GetAllContacts")

1. A final step **GetNoContact** is added just to confirm that the list of contacts is indeed empty. The REST Request Step will contain 2 assertions:



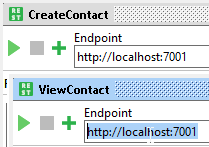
Script assertion is a more reliable way to achieve smarter checks. So, to **check** that there is **no** **contact** when Get Contacts REST operation is called, we put below code:

import groovy.json.JsonSlurper;

def responseMessage = messageExchange.response.contentAsString;

assert(responseMessage=="[]"): "Response Message was expected to be null. Response Message: " + responseMessage

Except the Cleanup of DB we have to improve another thing. This has to do with some data that are repeated in many test cases. The **ENDPOINTS**.



When we deliver a test automation project, it is very possible when the customer tries to execute the test in his/her site, the endpoints to be different. It is not a good practice to tell him/her, update all the endpoints of all the test steps by hand.

The solution is to use **CUSTOM PROPERTIES** and an **EXTERNAL PROPERTY FILE**

1. Create a txt file in path **/src/test/resources** with name **application.properties**
2. Edit **application.properties** and insert below content

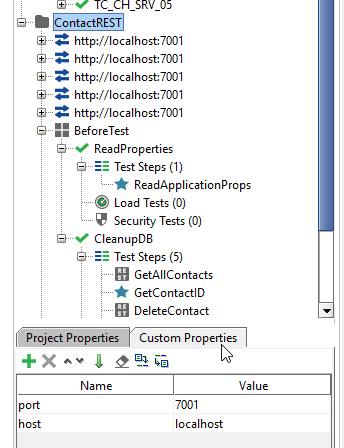
host=localhost

port=7001

1. Open SoapUI and project **demo-app-soapui-project-v2** and update all the ENDPOINTS of the already existing REST Request steps with below content:

http://${#Project#host}:${#Project#port}

1. With the above content we actually get the values from the **Custom Property** **area** of the **SoapUI** **Project**



1. What we need now is a groovy that reads the external property file **application.properties** and loads its content to the **Custom Properties** of SoapUI automatically.

Create another Test Case in **BeforeTest** test suite with name **ReadProperties** and create a Groovy Script test step with name **ReadApplicationProps** (names are indicative)

1. The content of the groovy script step **ReadApplicationProps** is displayed below:

// SET CUSTOM PROPERTIES

for( prop in testRunner.testCase.testSuite.project.properties) {

testRunner.testCase.testSuite.project.removeProperty( prop.key )

}

def projectPath = context.expand('${projectDir}');

Properties properties = new Properties()

File propertiesFile = new File(projectPath+"//../resources//application.properties")

propertiesFile.withInputStream { properties.load(it) }

Set<String> keys = properties.stringPropertyNames();

for (String key : keys) {

testRunner.testCase.testSuite.project.setPropertyValue(key, properties.getProperty(key))

}

In the same folder with the pom.xml, we will create another **SHELL SCRIPT** that runs only the TEST CASE CleanupDB and generates a report (Integration with surefire report). This script could be used from UI Tests or Performance Tests too, to clean-up the contacts. So, we create a new file with name **cleanup.sh** and below content:

# define a variable for the report's folder name using timestamp as suffix

export DIR="DemoAppCleanupDB\_TestReport\_"$(date +"%Y%m%d%H%M") ;

mvn clean test -DtestCase="CleanupDB" surefire-report:report-only site -DgenerateReports=false;

# generate test-reports folder and a subfolder with the name as defined in DIR variable above

mkdir -p "test-reports/$DIR" ;

# when running with maven a target folder is generated automatically. copy its content to the test-report folder

cp -a target/site/. test-reports/$DIR/ ;

cp -a target/surefire-reports/. test-reports/$DIR/ ;

# generate a logs subfolder and copy there all the log files that are generated by soapui

mkdir -p "test-reports/$DIR/logs";

mv \*.log test-reports/$DIR/logs ;

mv soapui.log.\* test-reports/$DIR/logs

# remove target folder

rm -rf target && rm \*.log && rm soapui.log.\* ;

# **4. Performance Testing with JMeter**

## **4.1 Basic Implementation**

1. **Download** **JMeter** <https://jmeter.apache.org/download_jmeter.cgi>

Place it in a folder as it is a **standalone** **application** (no setup wizard)

1. **Open** **JMeter** (navigate to **bin** folder and double click on **jmeter.bat**)
2. Describe **GUI Environment of JMeter**
3. Describe the **Scenario** to be implemented

100 users that reach gradually the system in 10 seconds, request this list of contacts for 30 seconds.

1. How this can be mapped with JMeters variable:

100 users, 10 ramp-up period, 30 seconds duration

1. First, we need our contact list to contain at list one user (not to be empty). So, our test will handle the creation of prerequisite data.

Open JMeter > rename project to **DemoAppPerfTests**

To be able to send REST requests, right click on **DemoAppPerfTests**:

Add > **Config Element** > **HTTP Header Manager**

Content-Type application/json

Accept application/json

Add > **Threads(Users)** > **setUp Thread Group** > rename it to **Setup**

Number of Threads: 1

Ramp-up period: 1

Loop Count: 1 (no infinitive)

Add > **Sampler** > **HTTP Request** > rename it to **Create Contact**

Protocol > http

Server Name or IP > localhost

Port Number > 7001

Method > POST

Path > /contacts

Body Data:

{

"address": "Address 1",

"city": "Athens",

"disabled": "false",

"email": "mail01@mail.com",

"gender": "m",

"name": "Name 1",

"phone": "2101111111"

}

1. To see the test logs/results, right click on **DemoAppPerfTests**:

Add > **Listener** > **View Results Tree**

Add > **Listener** > **Summary Report**

1. Run the test clicking the button 
2. Check the View Results Tree.   
   When a new contact is created Response data, contains its content in JSON format.
3. Check from the UI that contact is created
4. Before each execution we can clean up the logs, clicking the button 
5. To create the performance test, right click on **DemoAppPerfTests**:

Add > **Threads(Users)** > **Thread Group** > rename it to **Performance Test**

Add > **Sampler** > **HTTP Request** > rename it to **Create Contact**

Protocol > http

Server Name or IP > localhost

Port Number > 7001

Method > GET

Path > /contacts

1. Update **Performance Test** Thread Group with below arguments:

Number of Threads: 100  
Ramp-up period: 10  
Loop Count: 0 (infinitive YES)  
Duration: 30

## **4.2 Packaging & Refactoring**

After implementing the performance test cases there are 2 basic challenges we must consider:

* The **Packaging**:
  + An organized way to deliver the tests so they can be easily executed from command line and not opening JMeter.
* The **Refactoring**:
  + Are the test cases correctly implemented **so they can be re-executed** again and again without failures? Do they contain **methods to cleanup/load** all the needed data before each re-execution?
  + Do they contain **external PROPERTY files** with global variables like host/port to be handled externally and not within each test step?

1. First thing to improve is the use of **User Defined Variables** that can be handled externally (out of JMeter)
2. To set **User Defined Variables**, right click on **DemoAppPerfTests** and add below content:

protocol ${\_\_P(protocol,http)}  
host ${\_\_P(host,localhost)}  
ports ${\_\_P(ports,7001)}

1. To set default protocol/host/ports for all the HTTP requests (to avoid the repetition), right click on **Setup** and **Performance Test**

Add > Config Element > HTTP Request Defaults

Protocol [http] ${protocol}

Server Name or IP ${host}

Port Number ${ports}

1. Remove from steps **Create Contact** and **Search Contact** the values of protocol/server name or ip/port number
2. Close JMeter, create folders **src/test/jmeter** and transfer there the **jmx** file
3. In the folder with the **jmx** file, create a folder **src/test/resources**
4. Copy-Paste from **JMeter bin** folder to **src/test/resources** folder, file **reportgenerator.properties** and folder **report-template**
5. In the folder with jmx file, create a pom.xml file and insert appropriate content (copy from repo)
6. To **run** the **tests** just run command

**mvn clean verify**

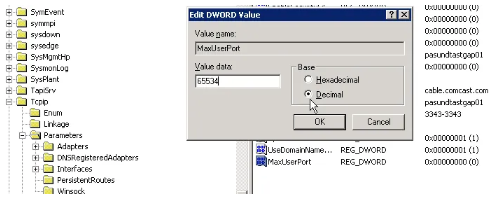
1. To **generate** the **report** just run command

**mvn pre-site**

Report is generated in path: **/target/jmeter/results/dashboard/**

If we get an exception **java.net.BindException: Address already in use: connect**   
this issue has to do with the Windows OS and registry must be updated as described below:

* Start Registry Editor (regedit)
* Locate the following subkey in the registry, and then click Parameters: HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
* Right mouse clicked on Parameters.
  + Create a new DWORD value with name MaxUserPort
* Right mouse click on MaxUserPort .
  + then enter 65534 for the Value Data field.
  + Under the radio button selection for Base, select the Decimal radio button.
* Restart the machine.



1. To have more reliable results in our tests we must load the DB with more than 1 contacts. To avoid the repetition of step “Create Contact” in Setup Thread we will loop the step 30 times, to create 30 different contacts with dynamic content to achieve uniqueness.

Right click on **Setup** Thread

Add > **Pre Processors** > **BeanShell PreProcessor** > Rename it to **Generate Unique Email** and Drag n drop it before **Create Contact**

1. Insert script content of **Generate Unique Email** step with below content

import java.text.SimpleDateFormat;

sdf = new SimpleDateFormat("yyyyMMddHHmmssSSS");

timestamp = sdf.format(new Date());

vars.put("uniqueEmail", timestamp + "@mail.com");

vars.put("uniqueNumber", timestamp.substring(timestamp.length() - 5));

1. To achive 30 times execution, click on **Setup** Thread and update below variable

**Loop Count 30**

1. To use the **UNIQUE EMAIL** that is generated by the step **Generate Unique Email** click on step **Create Contact** and update the value of **email, name,phone & address** variables

{

"address": "Address ${uniqueNumber}",

"city": "Athens",

"disabled": "false",

"email": "${uniqueEmail}",

"gender": "m",

"name": "Name ${uniqueNumber}",

"phone": "21011${uniqueNumber}"

}

1. Try to run only the **Setup** Thread (comment out the **Performance Test** Thread) to approve the creation of the prerequisite data (at least 30 contacts)

To **avoid another implementation** to **clean up** the test data via JMeter, in the Test Design Specification Document we can point to the section of **SoapUI Cleanup** step for those who need to cleanup data before the execution of the performance tests. Solution exists in Service Test Project. So why don’t we take the advantage to re-use it for the other types of test (Performance, UI).