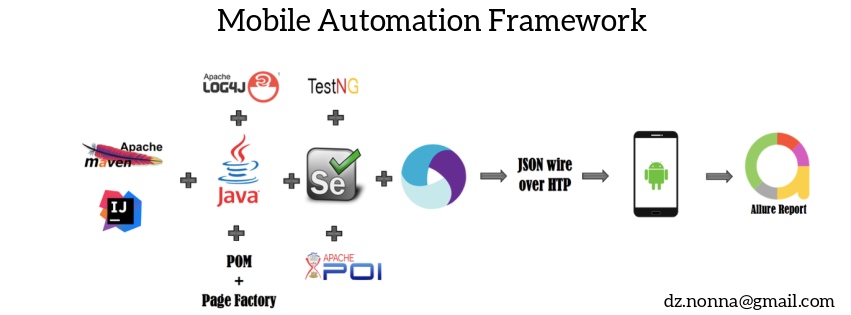
**Mobile QA Automation Framework**

**PPart 1. Framework architecture**

In this tutorial we will learn how to built Automation Mobile Testing Framework from scratch.



**About framework:**

In this framework we will use Java. Even though Selenium supports multiple languages, I preferred Java because it is a platform independent language, and it is Object Oriented.

It is a data driven framework. I chose this type of framework because it allows us to execute the same test case with multiple sets of data. Some test data will be kept in "DataProvider.xlsx" excel file, and some in @DataProvider method. Also, there is a properties file, which will store information that remains static, such as browser specific information, web page URL, OS type, and test type.

We will use Page Object Model and Page Factory pattern to separate element locators and tests.

As a testing framework we will use TestNG for Assertions, Grouping and parallel execution. Also, we will implement TestNG listeners to capture screenshots in case of test failure, and will store them in separate folder.  
As a project management tool we will use Maven, execution and dependency purpose. Integrating with the TestNG dependency in POM.xml file and running this POM file using Jenkins.  
For the reporting purpose, we will use Allure report. It generates beautiful HTML reports. Also, we will implement the log4j for lodging purpose. This will help us to detect the location and type of error at any point of a test case failure.

***Step 1.Create Maven project***

In a terminal (\*uix or Mac) or cmd(Windows), navigate to the folder you want to create the Java project. Type this command :

mvn archetype:generate

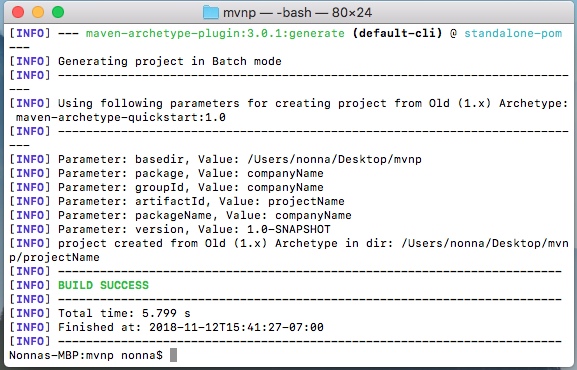
"-DgroupId=companyName"

"-DartifactId=projectName"

"-DarchetypeArtifactId=maven-archetype-quickstart"

"-DinteractiveMode=false"

You will see the next:



***Step 2. Import completed project to IntelliJ IDEA***

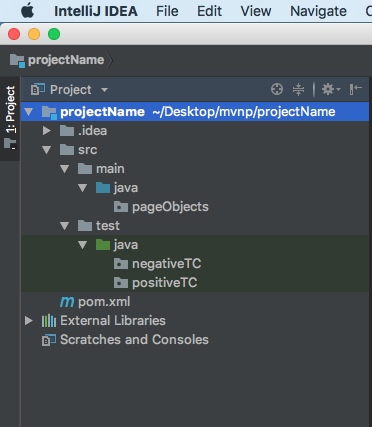
To import the project open IntelliJ IDEAandproceed to the steps shown below.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

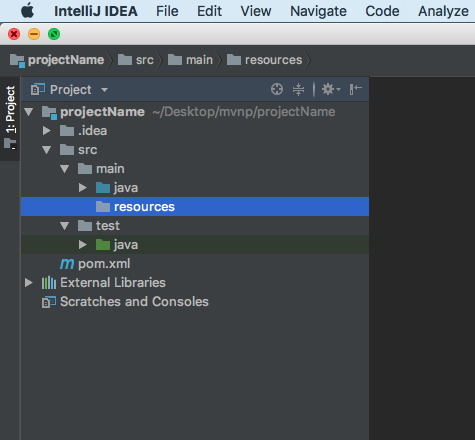
**Step 3. Apply Page Object Model**

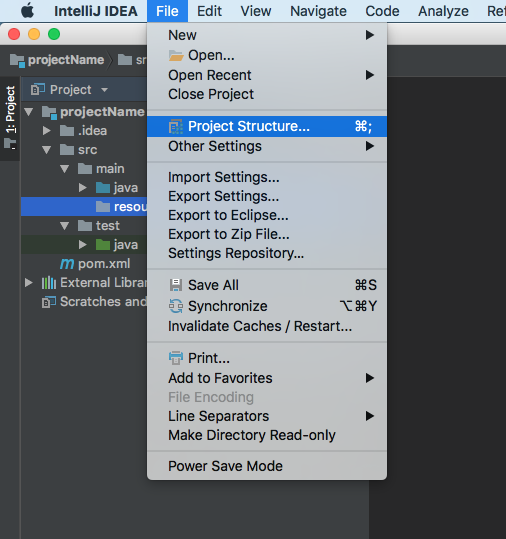
In this framework we will use POM and Page Factory pattern to separate element locators and tests. Each page has a separate class, and that class holds the functionality and members of that page. We will store all elements and locators in src-main-java-pageObjects and Test Cases in src-test-java-testCases. For convenience we will store Positive TC and Negative TC in the separate folders.

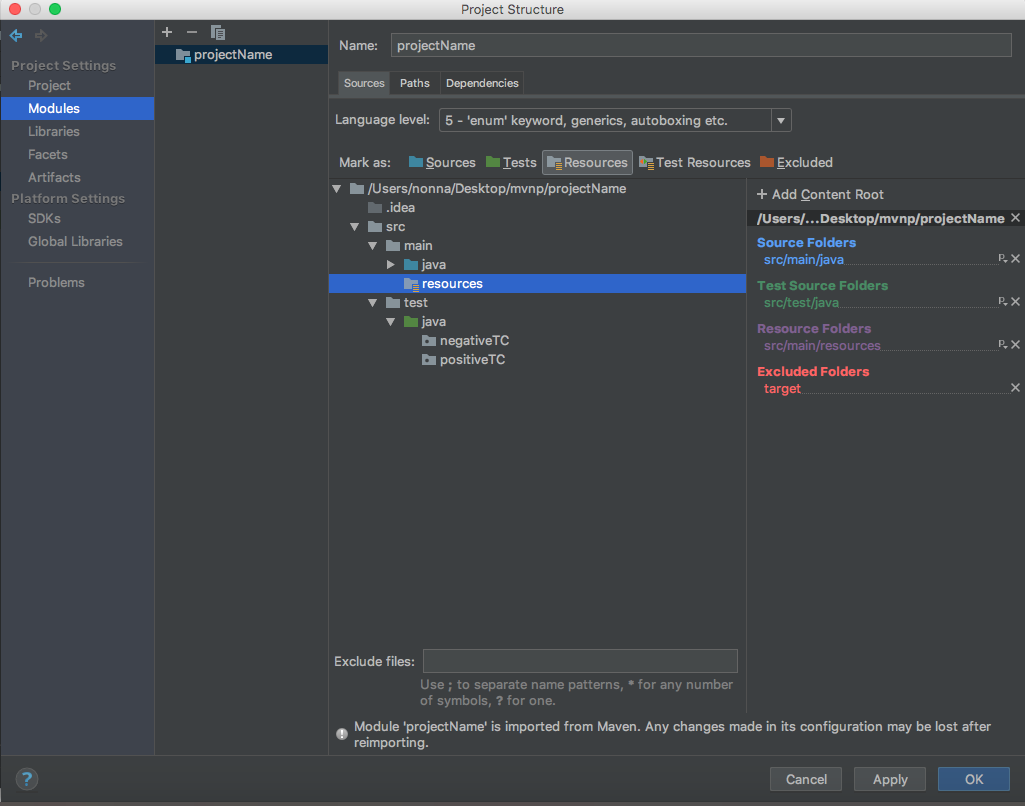
For that, we need to create new packages(File-New-Package), as shown below.



**Step 4. Create resources folder**Also we need resources folder, where we will keep all configuration, data provider and XML based files. For example the logging library log4j stores log4j.xml in this folder by default. Create new directory — “resources” in main folder (File-New-Directory). Then go to Project Structure and mark this directory as shown below.







**Step 5.Configure dependencies in POM.xml**

Go to Maven repository site — [https://mvnrepository.com/artifact/io.appium/java-client](https://vk.com/away.php?to=https%3A%2F%2Fmvnrepository.com%2Fartifact%2Fio.appium%2Fjava-client&cc_key=)

Add dependencies:

— Add Selenium (Selenium Java) dependency

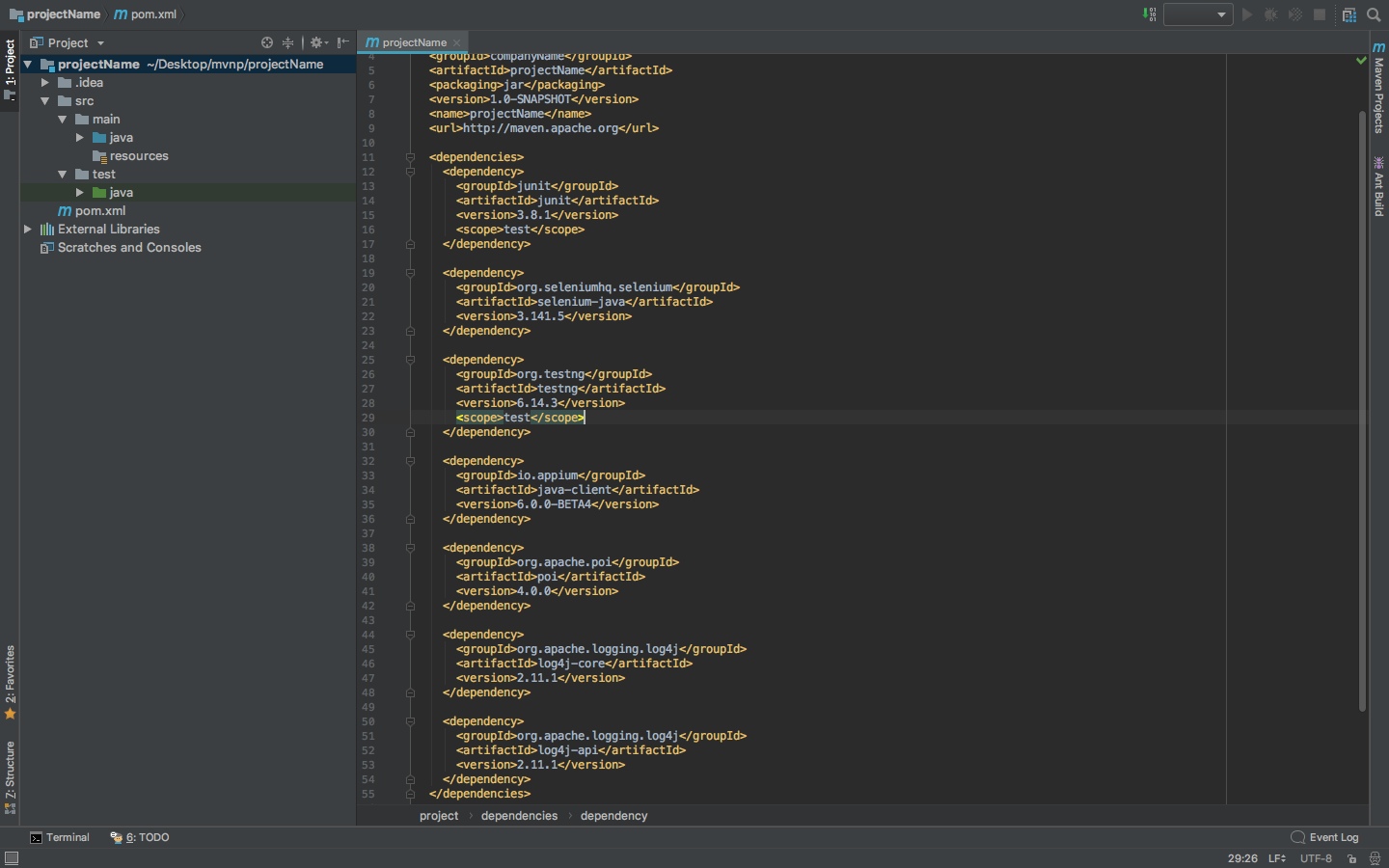
— Add TestNG dependency

— Add Appium (Java client for Appium Mobile Webdriver) dependency

— Add Apache POI dependency

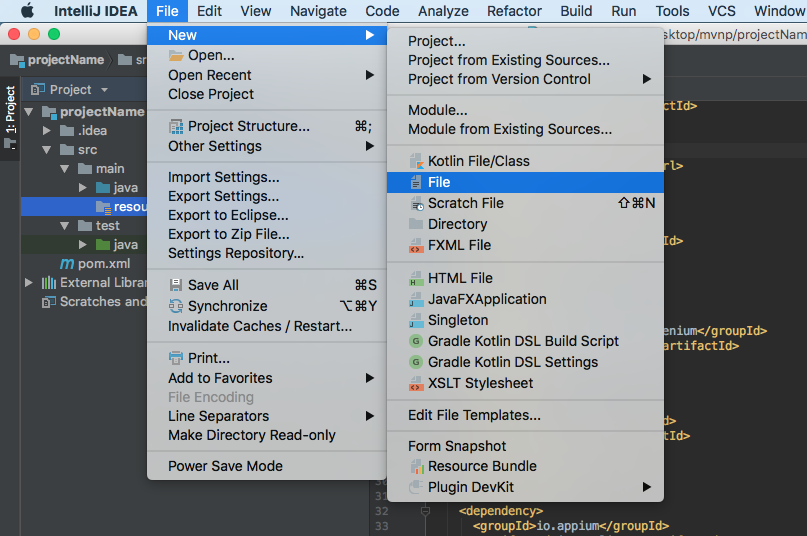
— Add Log4j API and Core dependencies

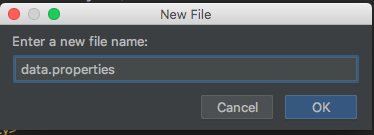
After all adds your POM.xml file should look as shown below.

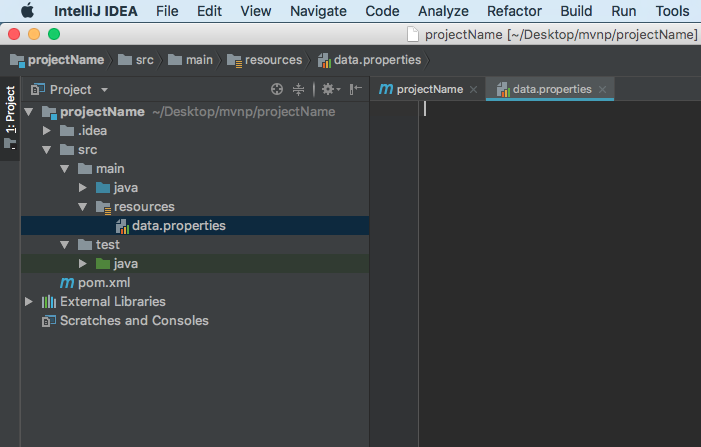


**Step 6. Create properties file.**

The next step is create properties file, which will store information that remains static, such as browser specific information, web page URL, OS type, and test type.For that we should create new file “data.properties” in resources folder. Proceed to the next steps.







That's all about Framework architecture. In the next part we will see, how to create Page Object classes and Test Cases. Also we will learn couple patterns witch will help us make our code more reusable.

*Author: Nonna Dzhabieva*