**SELENIUM NOTES - HG**

|  |  |
| --- | --- |
| Content | Link |
| Actions | [Actions](#Actions) |
| **Advantages of Selenium** | [**Selenium Advantages**](#AdvantagesOfSelenium) |
| Alert  **Alert alert = driver.switchTo().alert();**  alert.accept() & alert.dismiss() | [**Alert**](#Alert) |
| Apache POI  DataDriven Testing with Excel | [Apache POI](#ApachePOIExcel)  [DataDriven Testing with Excel](#ApachePOIExcel) |
| API vs Webservices | [API vs Webservices](#APIvsWebservices) |
| API- End Point | [API- End Point](#APIEndPiontVerifyValueResponseBody) |
| API- Type of Authentications | [API- Type of Authentications](#APITtpeOfAuthentications) |
| API- Verify Value in your Response Body | [API- Verify Value in your Response Body](#APIEndPiontVerifyValueResponseBody) |
| AWS – Amazon Web Services | [AWS – Amazon Web Services](#AWSandBasePage) |
| Base Page / Parent Class | [Base Page / Parent Class](#AWSandBasePage) |
| Cross Browser Test | [**Cross Browser Test**](#CrossBrowserTest) |
| Cucumber configuration.properties | [Link](#CucuHookStepDefCconfprop) |
| Data Driven Framework | [**Data Driven Framework**](#DataDrivenFramework) |
| **Disadvantages of Selenium** | [**Selenium Disadvantages**](#DisadvantagesOfSelenium) |
| driver.close() vs driver.quit() | [driver.close() vs driver.quit()](#driver_close_vs_driver_quit) |
| Drop Down & Methods | [**Drop Down Methods**](#SeleniumDropDownMethods) |
| **Element Locators** | [**Element Locators**](#ElementLocators) |
| ETL Testing | ETL Testing |
| **Exceptions in Selenium** | [**Exceptions in Selenium**](#SeleniumExceptions) |
| **Explicit Wait Expected Conditions** | [**Explicit Wait Expected Conditions**](#ExplicitWaitExpectedCondt) |
| Feature File | [Cucumber Feature File](#CucuFeatureFile) |
| Framework Selenium Framework | [Selenium Framework](#SeleniumFramework) |
| Hook Class | [Link](#CucuHookStepDefCconfprop) |
| iFrame | [**iFrame**](#iFrame) |
| **Implicit Wait vs Explicit Wait** | [**Implicit Wait vs Explicit Wait**](#ImplicitExplicitWait) |
| **Limitations Selenium** | [**Selenium Limitations**](#LimitationsOfSelenium) |
| Maven Project | [Maven Project](#MavenPro) |
| Moving nth Child element using XPath | [Moving nth Child element using XPath](#Moving_nth_ChildElementUsingXPath) |
| POM XML | [POM XML](#POM_XML) |
| Rest and RESTful | [Rest and RESTful](#RESTfulWebServQuestions) |
| RestAssured Library and API/WebServ.. | [RestAssured Library and API/WebServ..](#RestAssuredLibrary) |
| RESTful -Protocol used for RESTful  URI -What is URI | [Protocol used for RESTful](#RESTfulWebServQuestions)  [What is URI](#RESTfulWebServQuestions) |
| Runner Class | [Cucumber Runner Class](#CucuRunnerClas) |
| Selenium Flavors | [Flavors of Selenium](#flavorsOfSelenium) |
| Selenium is | [Selenium is](#SeleniumIs) |
| Serialization / De-Serialization | [Serialization / De-Serialization](#SerializationandDe_Serialization) |
| SOAP vs RESTful | [SOAP vs RESTful](#RESTfulWebServQuestions) |
| SQL - Constrains | [SQL - Constrains](#SQLReview1) |
| SQL - DML vs DDL  (Data Manuplation and Data Definition Language) | [SQL - DML vs DDL (Data Manuplation and Data Definition Language)](#SQLReview2) |
| SQL - PL/SQL Functions and Triggers in Oracle | [SQL - PL/SQL Functions and Triggers in Oracle](#SQLReview2) |
| SQL – Below average Salary Employees | [SQL – Below average Salary Employees](#SQLReview4Examples) |
| SQL – Connecting Database | [SQL – Connecting Database](#SQL_ConnectDatabase) |
| SQL – Data Types | [SQL – Data Types](#SQLReview1) |
| SQL – Duplicate Names Finding | [SQL – Duplicate Names Finding](#SQLReview4Examples) |
| SQL – JOIN Tables | [SQL – JOIN Tables](#SQLReview3Join) |
| SQL – Maximum Salaries | [SQL – Maximum Salaries](#SQLReview4Examples) |
| SQL – Second Highest Salary | [SQL – Second Highest Salary](#SQLReview4Examples) |
| SQL – Top 3 High Paid Employees | [SQL – Top 3 High Paid Employees](#SQLReview4Examples) |
| SQL – UNION | [SQL – UNION](#SQLReview3Join) |
| SQL- RDBMS, Database Schema | [SQL- RDBMS, Database Schema](#SQLReview1) |
| Step Definition Class | [Link](#CucuHookStepDefCconfprop) |
| TestNG & TestNG vs JUnit | [TestNG & TestNG vs JUnit](#TestNG) |
| Versions | [Versions](#Versions) |
| **XPath** | [**XPath**](#XPath) |
| XPath Moving Parent element using XPath | [Moving Parent element using XPath](#MovingParentElementUsingXPath) |
| **XPath Absolute and Relative** | [**XPath Absolute and Relative**](#AbsoluteandRelativeXPath) |
| XPath Handling Dynamic Elements  Contains(), Starts-with(), Text(), AND, OR | **[XPath Handling Dynamic Elements](#HandlingDynamicElements)**  [Contains(),Starts-with(), Text(), AND, OR](#HandlingDynamicElements) |
| **XPath Syntax** | [**XPath Syntax**](#XPathSyntax) |
| XPath vs CSS | [XPath vs CSS](#XPath_vs_CSS) |
| Background in Cucumber | Background is run before each of your scenarios but after any of your @Before hook |
| Desired Capabilities in Selenium | Desired Capabilities in Selenium |
| Entrance, Exit Criteria | [Entrance, Exit Criteria](#EnteranceAndExitCriteria) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Selenium**

What is Selenium?

Selenium is a suite of tools for automated web testing.

What are the flavors of Selenium?

* **Selenium IDE *(Selenium 1.0 / 2004)****:* *is a Firefox plugin that works for recording and play back.*
* **Selenium RC *(Selenium 2.0 / 2011):*** *is a test tool used to work with JavaScript to automate web applications.*
* **Selenium WebDriver** *(Selenium 3.0 / 2016): is a web automation framework and allows us to execute tests in different web browsers.*
* **Selenium Grid**, allows tests to run in parallel across multiple machines.

**Advantages** **of Selenium**

* Open source no licensing cost.
* Supports multiple languages like Java, Ruby, Python etc..
* Supports multi browser testing.
* It supports different operating systems/platforms like windows, mac, linux…
* Has large free resources and helping community.

**Disadvantages** **of Selenium**

* Supports only web based applications, DOESN'T support windows based apps.
* Needs third party tools for report generating.

**Limitations** **of Selenium**

* Unable to test desk top applications
* We CAN'T test web services using Selenium
* It needs external libraries and tools for performing tasks like testing frameworks (JUnit, TestNG), reading from external files (Apache POI for excel files)
* CAN'T automate Captcha.
* DOESN'T support file upload facility.

|  |  |  |  |
| --- | --- | --- | --- |
| **Implicit wait** | **VS** | | **Explicit wait** |
| Implicit wait is a type of wait. (Which waits specific time while locating an element before throwing "NoSuchElementException")  As by default Selenium tries to find element immediately without wait. So, the implicit wait is useful. This wait is applied to all elements of the current driver instance | | **Explicit wait is a type of wait as well. It is applied to a particular web element until the expected condition specified is met.** | |
| If the condition is met before the time out, it will continue to next step, however if the condition is NOT met within the time out will throw "NoSuchelemetException" | | If the condition is met before the time out, it will continue to next step, however if the condition is NOT met within the time out will throw "TimeOutException" | |
| driver.manage()timeouts().implicitlyWait(5,TimeUnit.*SECONDS*) | | **WebDriverWait wait = new WebDriver Wait(driver, 5);**  **wait.until(ExpectedConditions.visibilityOf(***element***);** | |

**Expected Conditions** **can be used in Explicit Wait:** **.visibilityOf(element)**

**.elementTobeClickable(element)**

**.alertIsPresent()**

**Exceptions** **in Selenium:**

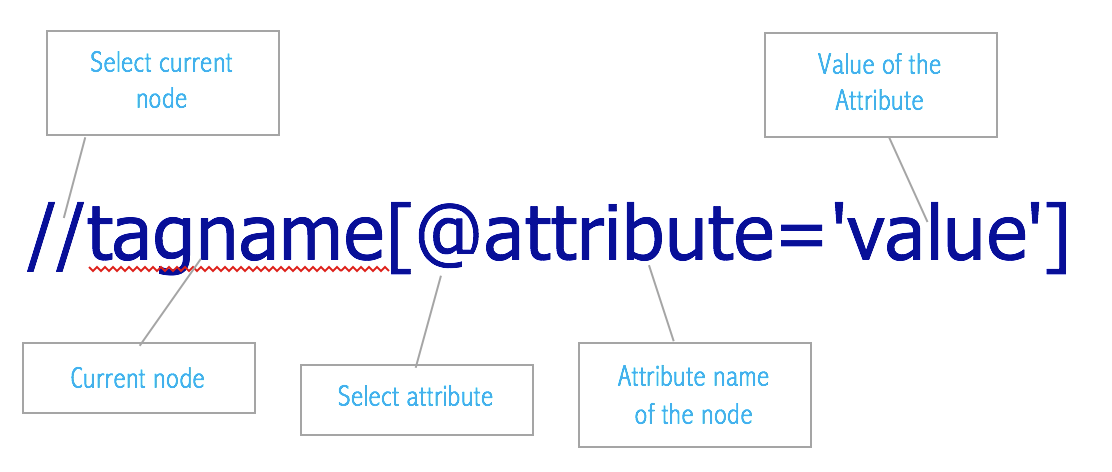
* **NoSuch Element Exception**
* **Element NotVisible Exception**
* **NoAlert Present Exception**
* **TimeOut Exception**
* **Webdriver Exception**

**Element** **Locators:**

* **id**
* **Name**
* **ClassName**
* **tagName**
* **linkText**
* **partialLinkText**
* **xpath**
* **cssSelector**

**XPath****:** XPath is used to find the location of any element on a web page using HTML standards.

|  |  |  |  |
| --- | --- | --- | --- |
| **Absolute** **XPath 🡪 /** | **VS** | | **Relative XPath 🡪 //** |
| Starts with single slash " / " starting from root element and all the way to the element. | | Starts with double slash " // " starts selection matching anywhere in the document. | |

Syntax for XPath

Handling Dynamic Elements*There are some methods I use:*

|  |  |
| --- | --- |
| Contains()  It is used to identify an element, when we are familiar with some part of the attributes value of an element | //\*[contains(@attribute\_name, 'attribute\_value'] |
| Starts-with()  It is used to identify an element, when we are familiar with the attributes value (starting with the specified text) of an element. | //\*[starts-with(@attribute\_name,'attribute\_value')] |
| Text()  It is used to locate an element based on the text available on a webpage | //\*[text()='New look for sign-in coming soon'] |
| AND | //\*[@attribute\_name1='attribute\_value1' and @attribute\_name2='attribute\_value2] |
| OR | //\*[@attribute\_name1='attribute\_value1' or @attribute\_name2='attribute\_value2] |

*[For More Methods click here](https://www.softwaretestingmaterial.com/dynamic-xpath-in-selenium/" \l "ANDXPATH)*

Moving Parent element using XPath

* Using "**..**" (double dot) expression in XPath we can move to parent element.

Moving nth Child element using XPath

*There are two ways:*

* Using square brackets "[ ]" with index position 🡪 **div[2]** will find the second div element
* Using **position()** method 🡪 **div[position()=2]** will find you the second div element

|  |  |  |  |
| --- | --- | --- | --- |
| **X****Path** | **VS** | | **CSS Selector** |
| XPath CAN search element back and forward | | CSS works only in forward direction | |
| XPath CAN work with text | | CSS CAN'T work with text | |
| XPath has more combination and CAN search by index | | CSS CAN'T search by index. | |
| Slower | | Faster | |

|  |  |  |  |
| --- | --- | --- | --- |
| **driver.close()** | **VS** | | **driver.quit()** |
| Closes current browser | | close all the browser instances | |

What is Selenium **Framework?**

Selenium framework is a **code structure** that helps to make code maintenance easy, code readability and code re-usage.

Frameworks: Data Driven

Hybrid Driven

**Data** **Driven Framework**

**Data Driven Testing Framework** helps the user separate the test script logic and the test data from each other (means we can store data externally). Sometimes we frequently need to test the same feature or function of an application multiple times with different sets of data. In that case it's critical that the test data not be hard-coded in the script itself (this happens with a Linear or Modular-based testing framework)

**Data-driven test framework** will allow us to store and pass the input/ output **parameters** to test scripts from an **external** **data** **source**, such as **Excel** **sheets**, **Text Files**, CSV files, **SQL Tables**, or ODBC repositories.

The **test scripts** are **connected** to the **external data source** and told to read and populate the necessary data when needed.

Data is conventionally stored in “Key-Value” pairs. So, the key can be used to access used in within the test code.

Selenium WebDriver doesn't support read and write on excel file alone therefore, third party API's like **Apache POI** needed.

***Advantages of a Data-Driven Framework:***

* Tests can be executed with multiple data sets.
* Multiple scenarios can be tested quickly by varying the data, thereby reducing the number of scripts needed.
* Hard-coding data can be avoided so any changes to the test scripts do not affect the data being used and vice versa.
* You’ll save time by executing more tests faster.

***Disadvantages:***

* You’ll need a highly-experienced tester who is proficient in various programming languages to properly **utilize** this framework design. They will need to identify and format the external data sources and to write code (create functions) that connect the tests to those external data sources seamlessly.
* Setting up a data-driven framework takes a significant amount of time.

Hybrid Driven Framework

Combination of frameworks.

**Dropdown**

WebElement dDown = **driver.findElement(By.id("dropdown"));**

Select dropDownList = new Select(dDown);

***Methods:***

getAllSelectedOptions().get(0)

getOptions()

SelectByVisibleText("…tex…")

SelectByIndex(...int…);

SelectByValue("...value…");

**Alert**

**Alert alert = driver.switchTo().alert();**

alert.accept();

alert.dismiss();

alert.getTetx();

alert.sendKeys();

*or use WebDriverWait (See:https://seleniumatfingertips.wordpress.com/2016/07/05/check-whether-alert-is-present-on-the-webpage-or-not/)*

WebDriverWait wait = new WebDriverWait(driver, 2);

wait.until(ExpectedConditions.alertIsPresent());

Alert alert = driver.switchTo().alert();

alert.accept();

**iFrame**

**driver.switchTo().frame**("…**index** or **frame** **name** here...");

driver.switchTo().defaultContent();

defaultContent() method: Selects either the first frame on the page, or the main document when a page contains iframes.

*//switch to first frame in the frameset.*  
 driver.switchTo().defaultContent();  
*//if there are inline frames in the page, switch to the main document of the page.*

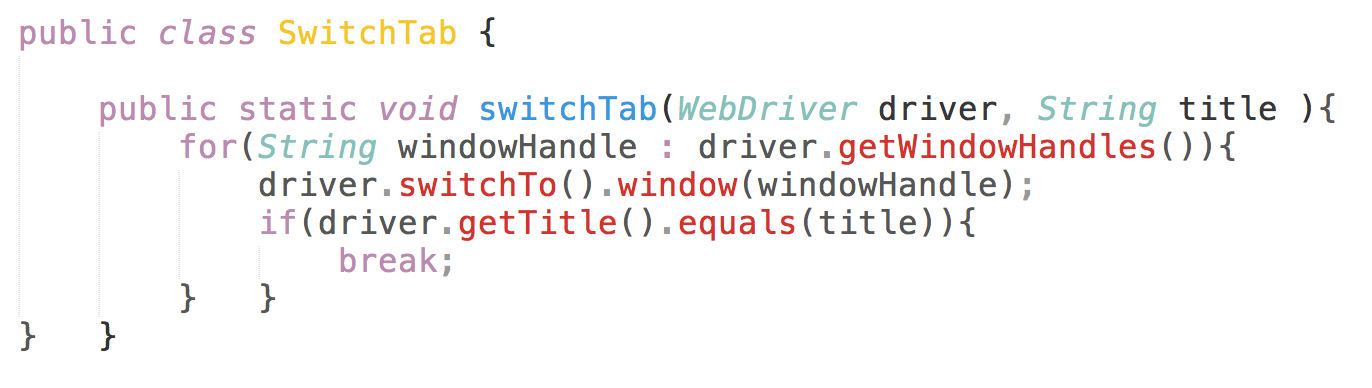
When we have multiple frames in the web page, we need to switch to the frames to perform operations in specific frame.  
There are 2 possibilities.

1. Page contains frame set
2. Page contains iframe

In first case, **DefaultContent()** will hand over the control to the first frame in the frame set.

In second case, **DefaultContent()** will hand over the control to the main document in the page.  
  
Ways to locate an iFrame: byName, byId, byIndex and byTagName and using xpath or CSS

**Switching Tab**



**Actions**

Action action = new Action(driver);

Action.build()

.perform()

.dragAndDrop()

.howerOver()

.moveToElement()

.doubleClick()

.rightClick()

In order to do action events, you need to use **org.openqa.selenium.interactions.Actions** class.

The Action class is user-facing API for imitate complex user action events. You can directly use this class rather than using the input devices, i.e. Keyboard or Mouse. It Implements builder pattern, in which a complex object is constructed that can be further used to create different representation of same object. In below code, a complex object ‘builder’ is created, which is used to create different actions.

AWS and Base Page

AWS is providing cloud VM. Create an EC2 instance. I can use this instance with remote desktop. Actually, after launch my instance I just use like a regular computer.

What is **Test Base** **Page**

Base page is where we store our common functionalities in a base class (parent class) later we extend that base class and use in sub class.

# **Desired Capabilities**

**DesiredCapabilities describe a series of key/value pairs that encapsulate aspects of a browser,   
Basically this is used to set certain properties of browser for the WebDriver.**

**Properties such as:**

* set the path of the browser if not installed in default location.
* **set the browser Version, Browser Name, platform** (Normally we use in selenium GRID)
* to set the firefox profile
* configurations related to browser cookies, SSL security popups,

Let’s see some of the implementation of DesiredCapabilities in selenium:  
**A part of the code to set browser, platform and version:**



*and some more in these links about DesiredCapabilities* [*link1*](https://qavalidation.com/2015/06/desired-capabilities-in-selenium.html/)[*link2*](https://sites.google.com/a/chromium.org/chromedriver/capabilities) [*link3*](https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities) [*link4*](https://stackoverflow.com/questions/17527951/what-is-the-use-of-desiredcapabilities-in-selenium-webdriver)

**TestNG** (interview Questions: <https://www.softwaretestingmaterial.com/testng-interview-questions/> )

**TestNG** Annotations: @Test(priority=0) = (enable=false)

@BeforeTest @AfterTest

@BeforeClass @AfterClass

@BeforeSuite @AfterSuite

@BeforeMethod @AfterMethod

**JUnit** Annotations: @Test

@BeforeClass

@AfterClass

@Before

@After

@Ignore

**TestNG vs Junit Annotations**

* Both are testing framework to help us running **automation** scripts
* TestNG provides **HTML report**
* TestNG has **@Dataprovider** annotation same as Cucumber Scenario Outline for Data Driven Testing
* TestNG you can do **parallel testing** easier, but in Junit you have to divide your tests and run each divided package with separate Runner Class in different VMs. Easier way is Sauce Labs for Junit parallel testing.
* TestNG supports **group test**, Junit doesn't.
* Both TestNG and Junit are **parameterized testing** but TestNG parameterize test configuration is easy to configure.
* There are two ways to achieve parameterizing in TestNG:

@Parameters and TestNG XML file.

@DataProvider

**Cross Browser Test**

**Cross Browser and Parallel Test**

In my current project, we use **Sauce Labs** for cross browser testing but my previous project I used **TestNG XML file**.

Inside the **suite** there are 3 **keys** (**name**, **thread**-**count** and **parallel**) and I created 2 different tests, one of them is **Chrome** and the other is **Firefox**.

There is also **parameter** **annotation**, includes **name** and **value**;

Name is browser and value are Chrome

**<suite** name="Parallel Test" thread-count="2" parallel="classes"> #it may work without thread-count="2"

**<test** name="ChromeTest"> # you can give whatever name you want

<**parameter** name="browser" value="Chrome" /> #Chrome or whatever browser you want goes here

<**classes**>

<class name="com.simpletest.DriverSerup"/> # put the class name you want to test here

</classes>

</test>

<test name="Functional\_Test\_2"> # you can give whatever name you want

<parameter name="browser" value="Firefox" /> #Chrome or whatever browser you want goes here

<classes>

<class name="com.simpletest.DriverSerup"/> # put the class name you want to test here

</classes>

</test>

</suite>

**API vs Web Services**

**API**

Selenium WebDriver 🡪 Browser

JDBC 🡪 Database

Apache POI 🡪 Excel

**WebServices**

If an API uses **internet** for communication, it is a Web Service.

\*\*All Webservices are API, but not all APIs are webservices.

**API/Webservices**

No UI (For web application with UI we use Selenium WebDriver to test)

We use for Web Services:

**SOAP** 🡪 XML

**REST** 🡪 JSON, XML, TEXT

Postman, Rest Assured Library

HTTP methods used in REST based Architecture:

**Create 🡪 POST**

**Read 🡪 GET**

**Update 🡪 PUT**

**Delete 🡪 DELETE**

**Status Codes:**

|  |  |
| --- | --- |
| **1XX🡪 Informational** |  |
| **2XX 🡪 Success** | **200🡪 OK**  **201🡪 Created**  **202🡪 Accepted**  **204🡪 No Content** |
| **3XX 🡪 Redirection** |  |
| **4XX 🡪 Client Error** | **400 🡪 Bad Request**  **401 🡪 Unauthorized**  **403 🡪 Forbidden**  **404 🡪 Not Found (Service is down or URL is bad**  **405 🡪 Method Not Allowed** |
| **5XX 🡪 Server Error** |  |

Stateful vs Stateless Web Services

### Stateful Web Services

If the web server stores this data in a backend manner and uses it to identify you as a constantly connected client, the service is stateful.

As you use the web service, everything you do is referenced back to this stored state. When you request an account summary, the web service asks two things:

* Who is making this request?
* Using the ID stored for who is making this request, what should their page look like?

In a stateful web service like this, the response formed from a simple **GET** request is entirely dependent on the state registered by the server. Without knowledge of that state, your request cannot be returned properly.

**Stateful programming** is fine in some very limited applications, but it **has a lot of issues**. First and foremost, when you have to reference a state, you’re opening yourself up to a lot of incomplete sessions and transactions. Let’s say you make a call to present a piece of data. In a stateful system where the state is determined by the client, how long is the system supposed to leave this connection open? How do we verify if the client has crashed or disconnected? How do we track the actions of the user while maintaining the ability to document changes and roll back when necessary?

While there are certainly workarounds for all of these questions, more often than not, statefulness is only really useful when the functions themselves depend on the statefulness quality. Most consumers are able to respond to the server in intelligent, dynamic ways, and because of this, maintaining server state independent of the consumer as if the consumer was simply a “dumb” client is **wasteful** and **unnecessary**.

## **Stateless**

Stateless is the totally opposite of stateful, in which any given response from the server is **independent of any sort of state**.

The information is stored locally in such a way that the requests are **self contained** — it’s dependent only on the data you hold.

everything is contained within the request, and handled in two distinct phases, with a “request” and a “response”.

This is a stateless system. Your response is independent of the “0” or “1”, and each request is self contained.

### Stateless Web Services

Statelessness is a fundamental aspect of the modern internet — so much so that every single day, you use a variety of stateless services and applications. When you read the news, you are using **HTTP** to connect in a stateless manner, utilizing messages that can be parsed and worked with in isolation of each other and your state.

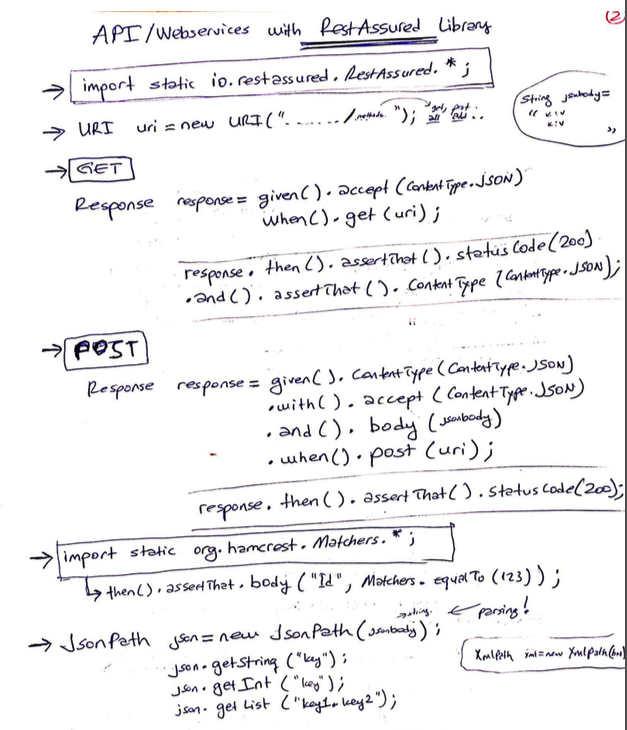
If you have **Twitter** on your phone, you are constantly utilizing a stateless service. When the service requests a list of recent direct messages using the Twitter REST API, it issues the following request:

GET https://api.twitter.com/1.1/direct\_messages.json?since\_id=240136858829479935&amp;count=1

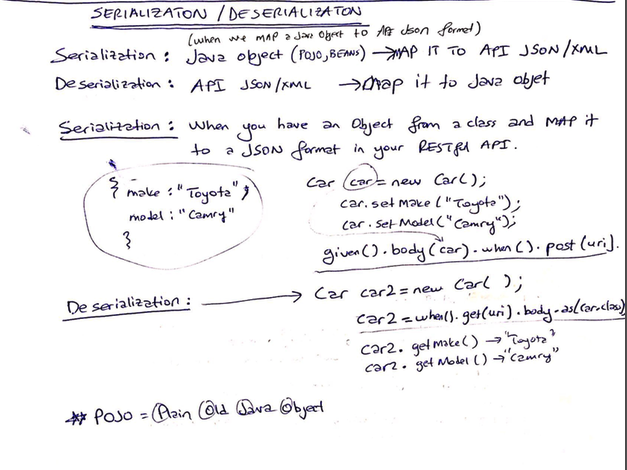
The response that you will get is entirely independent of any server state storage, and everything is stored on the client’s side in the form of a cache.

[REST](http://nordicapis.com/top-specification-formats-for-rest-apis/) is specifically designed to be **functionally stateless**. The entire concept of Representational State Transfer (from which REST gets its name) hinges on the idea of passing all data to handle the request in such a way as to pair the data within the request itself.

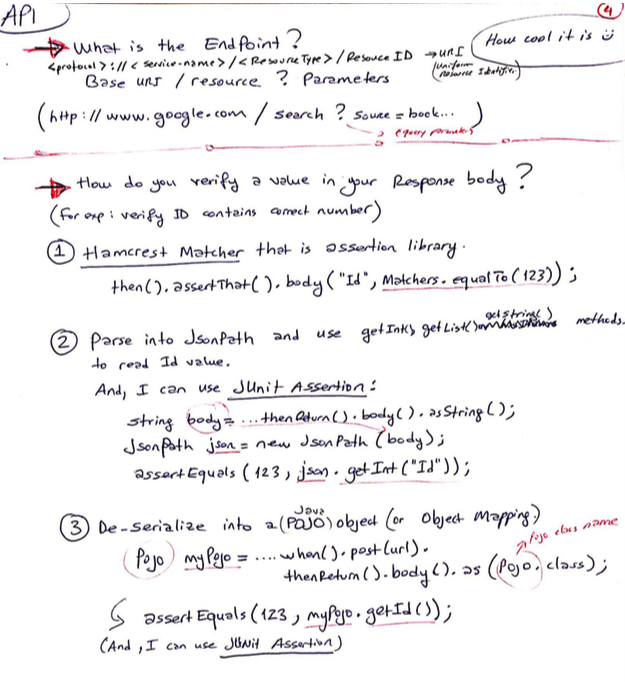
API/Webservices with RestAssured Library



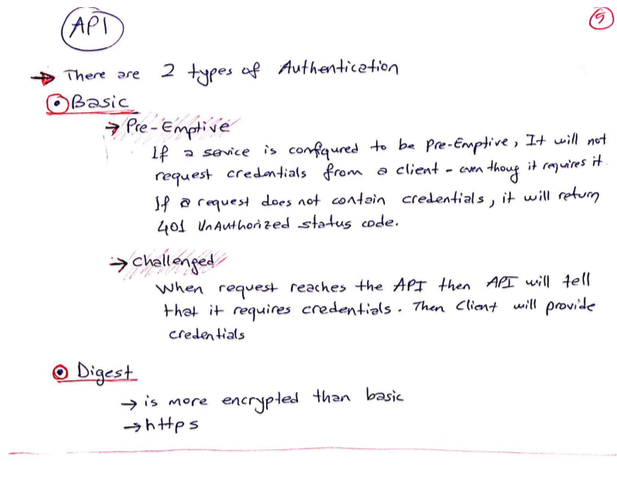
Serialization / De-Serialization



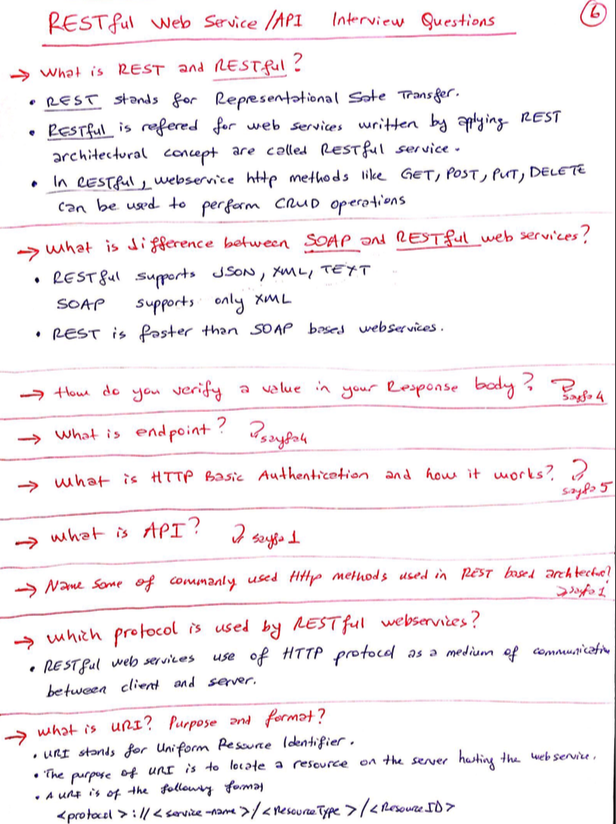
API- What is End Point, How Do you verify Value in your Response Body?



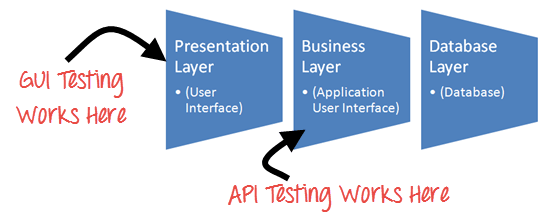
API- Type of Authentications



What is Rest and RESTful – SOAP vs RESTful etc..



## **What is API testing?**



Api Testing is entirely different from [GUI Testing](https://www.guru99.com/gui-testing.html) and mainly concentrates on the business logic layer of the software architecture. This **testing won't concentrate on the look and feel of an application.**

Instead of using standard user inputs(keyboard) and outputs, in Api Testing, you use software to send calls to the API, get output, and note down the system's response.

Api Testing requires an application to interact with API. In order to test an API, you will need to

* Use Testing Tool to drive the API
* Write your own code to test the API

## Set-up of API Test environment

* Api Testing is different than other testing types as GUI is not available, and yet you are required to setup initial environment that invokes API with required set of parameters and then finally examines the test result.
* Hence, Setting up a testing environment for API testing seems a little complex.
* Database and server should be configured as per the application requirements.
* Once the installation is done, API Function should be called to check whether that API is working.

## Types of Output of an API

Output of API could be

1. Any type of data
2. Status (say Pass or Fail)
3. Call to another API function.

Let's look at an example of each of above Types

**Any Type of Data**

Example: There is an API function which should add two integer numbers.

Long add(int a, int b)

The numbers have to be given as input parameters. The output should be a summation of two integer numbers. This output needs to be verified with expected outcome.

Calling needs to be done such as

add (1234, 5656)

Exceptions have to be handled if the number is exceeding the integer limit.

**Status (say Pass or Fail)**

Consider the below API function -

1. Lock()
2. Unlock()
3. Delete()

They return any value such as True (in case of success) or false (In case of error) as an output.

A more accurate[Test Case](https://www.guru99.com/test-case.html)would be, can call the functions in any of the script and later check for changes either in the database or the Application GUI.

**Calling of another API / Event**



In this case, we call one of the API function which in turn will call another function.

For example - First API function can be used for deleting a specified record in the table and this function in turn calls another function to REFRESH the database.

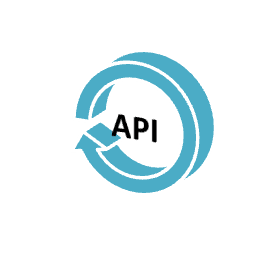
## Test Cases for API Testing:

Test cases of API testing are based on

* **Return value based on input condition:** it is relatively easy to test, as input can be defined and results can be authenticated
* **Does not return anything:**When there is no return value, behavior of API on the system to be checked
* **Trigger some other API/event/interrupt:**If output of an API triggers some event or interrupt, then those events and interrupt listeners should be tracked
* **Update data structure:**Updating data structure will have some outcome or effect on the system, and that should be authenticated
* **Modify certain resources:**If API call modifies some resources then it should be validated by accessing respective resources

## Approach of API Testing:

Following points helps the user to do API Testing approach:



1. Understanding the functionality of the API program and clearly define the scope of the program
2. Apply testing techniques such as equivalence classes, boundary value analysis and error guessing and write test cases for the API
3. Input Parameters for the API need to be planned and defined appropriately
4. Execute the test cases and compare expected and actual results.

## Difference between API testing and Unit testing

|  |  |
| --- | --- |
| Unit Testing | API Testing |
| Developers perform it | Testers perform it |
| Separate functionality is tested | End to end functionality has been tested |
| Developer can access the source code | Testers cannot access the source code |
| UI testing is also involved | Only API functions are tested |
| Only basic functionalities are tested | All functional issues are tested |
| Limited in scope | Broader in scope |
| Usually ran before check-in | Ran after build is created |

## What to test for in API testing

API testing should cover atleast following testing methods apart from usual SDLC process

* **Discovery testing:** The test group should manually execute the set of calls documented in the API like verifying that a specific resource exposed by the API can be listed, created and deleted as appropriate
* **Usability testing:**This testing verifies whether the API is functional and user-friendly. And does API integrates well with another platform as well
* **Security testing:**This testing includes what type of authentication is required and whether sensitive data is encrypted over HTTP or both
* **Automated testing:**API testing should culminate in the creation of a set of scripts or a tool that can be used to execute the API regularly
* **Documentation:**The test team has to make sure that the documentation is adequate and provides enough information to interact with the API. Documentation should be a part of the final deliverable

## Best Practices of API Testing:

* Test cases should be grouped by test category
* On top each test, you should include the declarations of the APIs being called.
* Parameters selection should be explicitly mentioned in the test case itself
* Prioritize API function calls so that it will be easy for testers to test
* Each test case should be as self-contained and independent from dependencies as possible
* Avoid "test chaining" in your development
* Special care must be taken while handling one time call functions like - Delete, CloseWindow, etc...
* Call sequencing should be performed and well planned
* To ensure complete test coverage, create test cases for all possible input combinations of the API.

## Types of Bugs that API testing detects

* Fails to handle error conditions gracefully
* Unused flags
* Missing or duplicate functionality
* Reliability Issues. Difficulty in connecting and getting a response from API.
* Security Issues
* Multi-threading issues
* Performance Issues. API response time is very high.
* Improper errors/warning to caller
* Incorrect handling of valid argument values
* Response Data is not structured correctly (JSON or XML)

## Tools for API testing

Since API and [Unit Testing](https://www.guru99.com/unit-testing-guide.html) both target source code, similar tools can be used for testing both.

* [SOAPUI](https://goo.gl/XczVPL) - It's an easy-to-use tool with extensive REST and SOAP API testing capabilities – no scripting experience is required.
* Runscope
* Postman with jetpacks
* Postman with Newman
* Curl
* Cfix
* Check
* CTESK
* dotTEST
* Eclipse SDK tool- Automated API testing

*Check out top API Testing Tools*[*List*](https://www.guru99.com/top-6-api-testing-tool.html)

## Challenges of API Testing

Challenges of API testing includes:

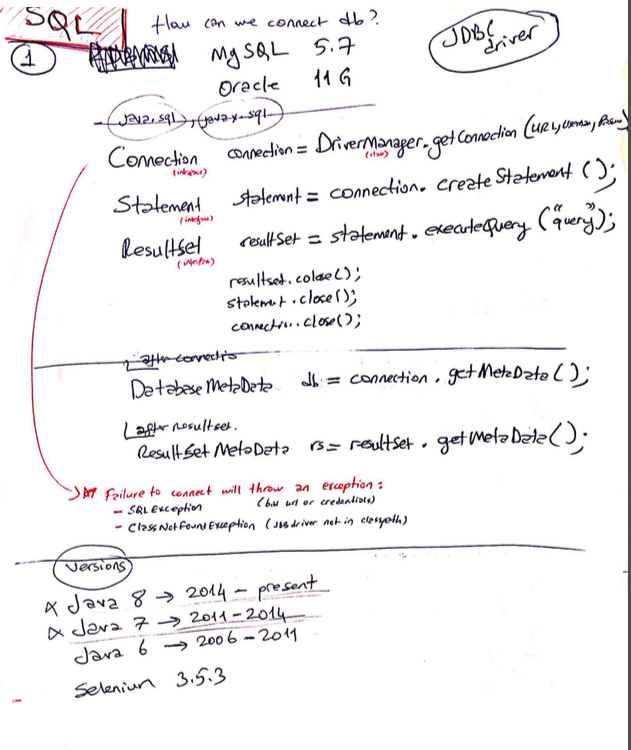
* Main challenges in API testing is **Parameter Combination, Parameter Selection, and Call Sequencing**
* There is no GUI available **to test the application which makes** difficult to give input values
* Validating and Verifying the output in different system is little difficult for testers
* Parameters selection and categorization is required to be known to the testers
* Exception handling function **needs to be tested**
* Coding knowledge is necessary for testers

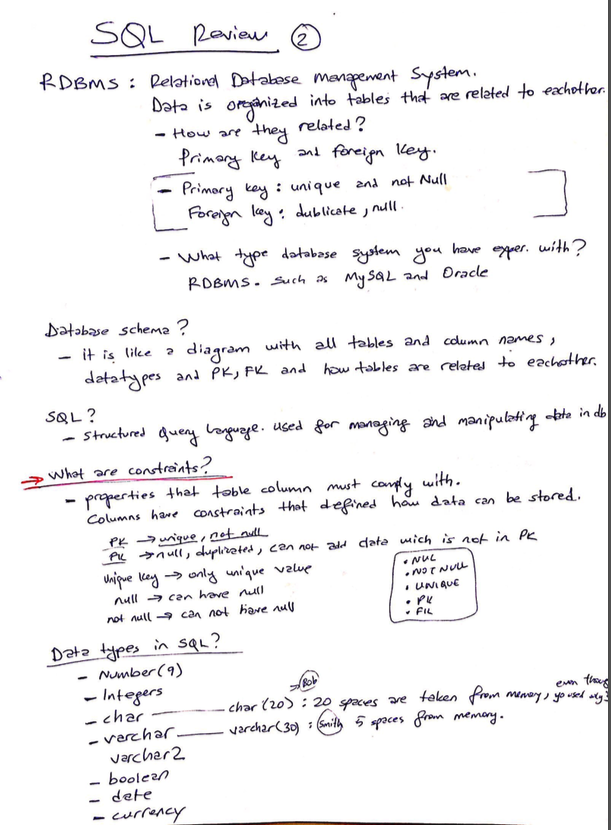
## Conclusion:

API consists of a set of classes / functions / procedures which represent the business logic layer. If API is not tested properly, it may cause problems not only the API application but also in the calling application.

SQL

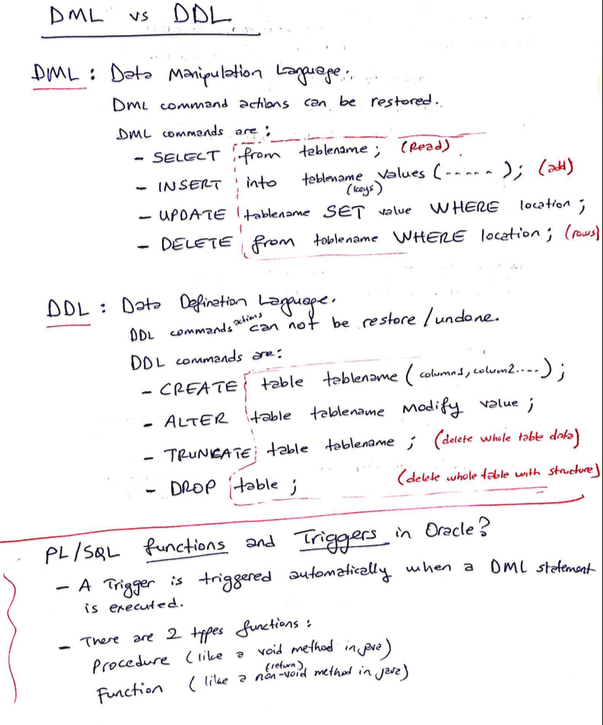
**Connecting Database**



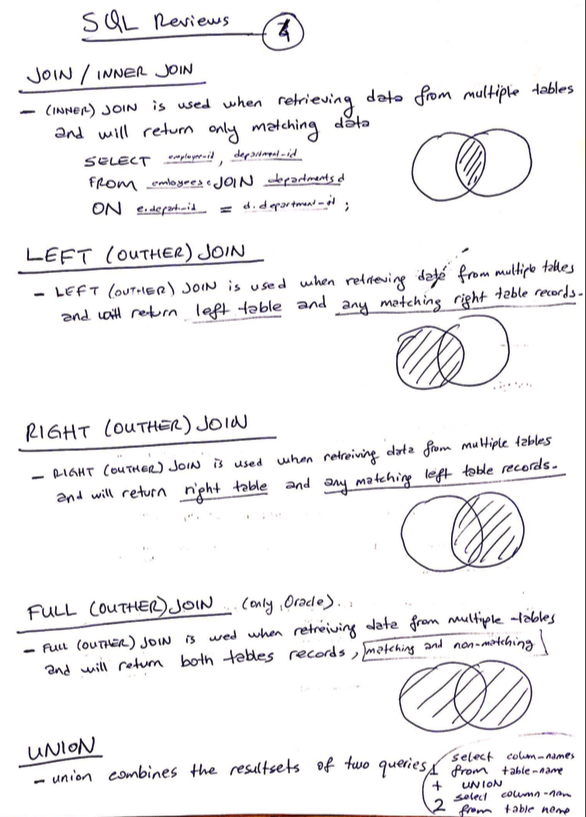


DML vs DDL (Data Manuplation and Data Definition Language)

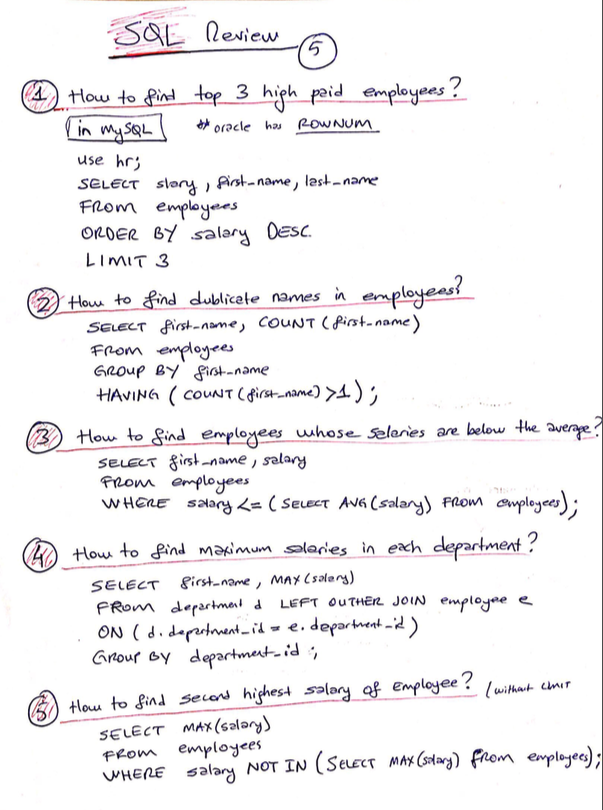
PL/SQL Functions and Triggers in Oracle



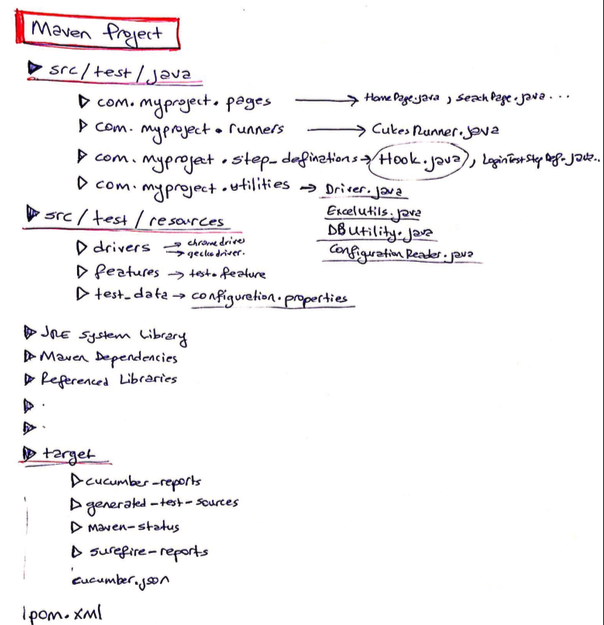
SQL - JOIN



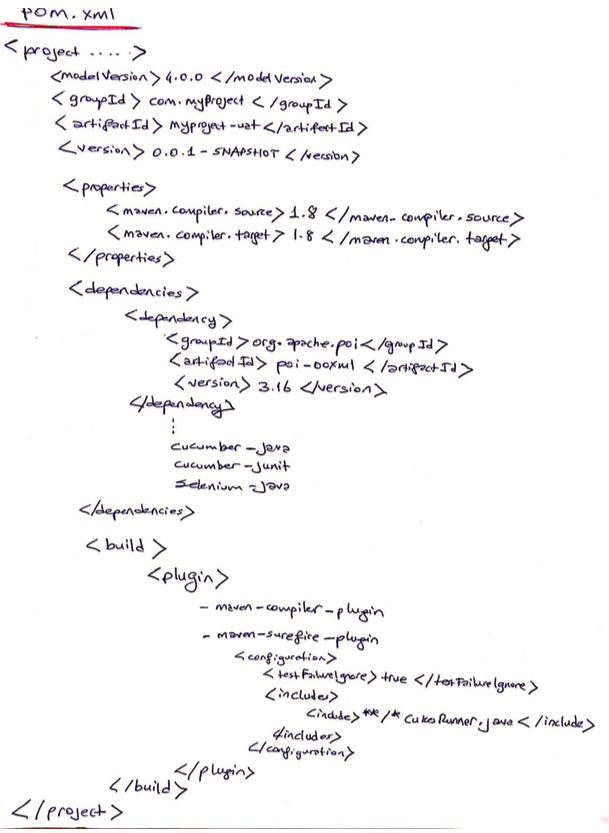
SQL Query Examples



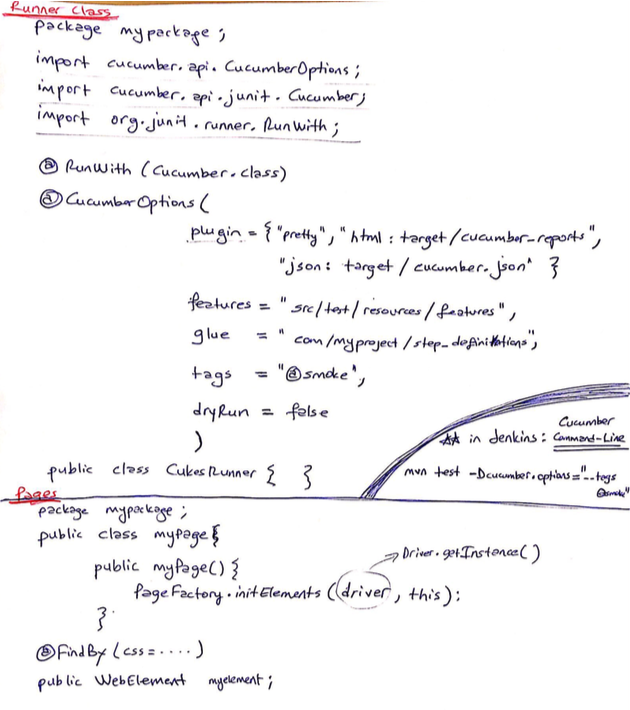
**Maven Project**



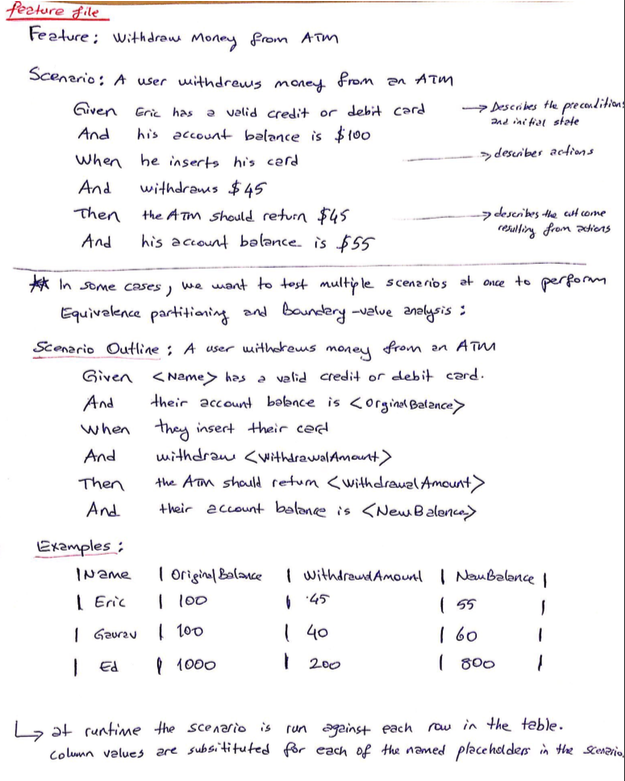
POM XML



**Cucumber Runner Class**



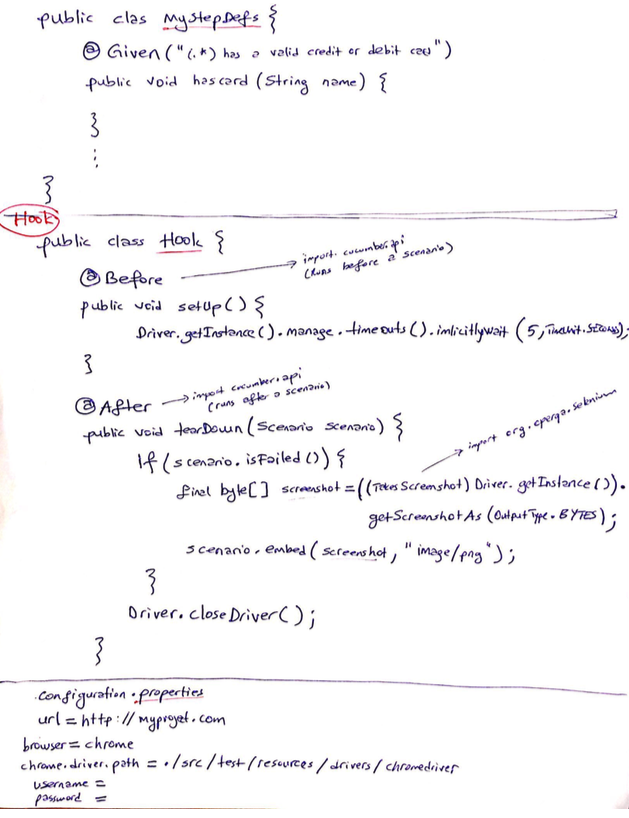
**Cucumber Feature File**



**Cucumber StepDef Class,**

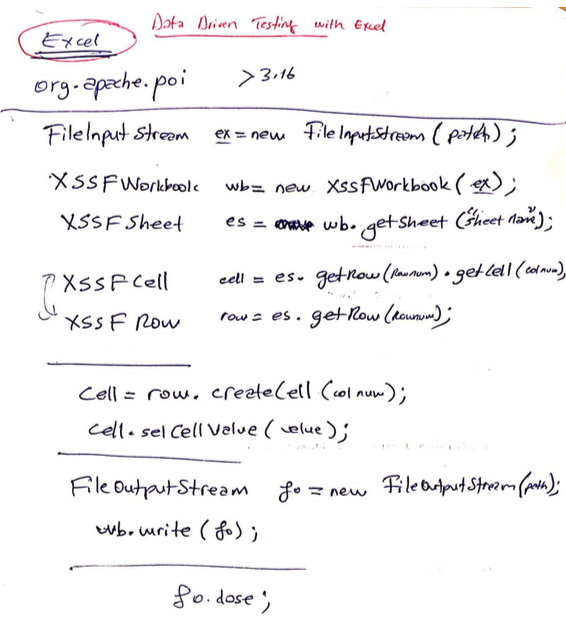
**Hook Class**

**Configuration.properties**

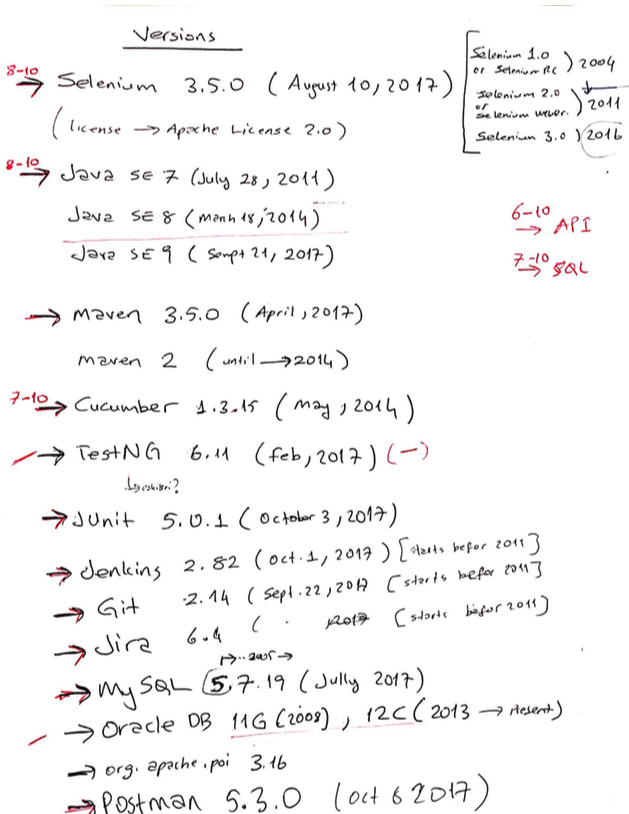


Apache POI

DataDriven Testing with Excel



**VERSIONS**



**Entrance and Exit Criteria**

Entrance Criteria = (DEV to TEST) which in turn is exit criteria for DEV

1. 100% requirements (User Story) are complete.
2. Unit testing performed in DEV environment
3. ALL Critical Defects are closed

Exit Criteria = (TEST to PRD) = Entrance Criteria for PROD

1. 100% test cases executed and passed
2. All Critical and High Defects are closed
3. Some Low defects might stay Open

-------------------------------

|  |  |  |  |
| --- | --- | --- | --- |
|  | **VS** | |  |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |

API/Webservices with RestAssured Library

🡪 Import static io.resrassured.RestAssured.\*;

🡪 URI uri=new URI("……/*methods*"); (*methods:*

🡪 GET

Git:

Team Share commit and push