Page Object Model (POM)

What is POM (Page Object Model)?

- Page Object Model is a **design pattern** to create an **Object Repository** for Web UI elements. For each page in the application, there should be a corresponding page class. This page class will find the **WebElements** of that web page and also contains **Page Methods** which perform operations on those Web Elements.
- Name of these methods should be given as per task that they are performing. For example, if a loader is waiting for payment gateway to appear, POM method can be: waitForPaymentScreenDisplay();

Why POM?

- UI Automation in Selenium WebDriver is NOT a very difficult task. You just need to find the elements and perform automations on it. But in time, there will be many webElements and more lines of code in the test suite. The script maintenance will be a very big problem. If there are 50 scripts for the same element, with any change in that element, you will need to change all 50 scripts. This will be time consuming and error prone.
- A better approach is to create a seperate class which would find the **webElements**, fill or verify them. The class can be reused in all the scripts using that element. If there is a change in the element, we would change in the class where we keep the element, not in all to 50 different scripts.

What are the advantages of POM?

- It makes the code more readable, maintainable, and reusable. You will have less and more optimized codes.
- POM says **operations** and **flows** in the UI should be separate from verification. This concept makes the code cleaner and easy to understand.
- **The object repository is independent of test cases,** so we can use the same object repository for a different purpose with different tools. For example, we can integrate POM with TestNG, JUnit for functional testing and at the same time with JBehave/Cucumber for acceptance testing.
- Methods get **more realistic names** which can be easily mapped with the operation happening in the UI. For example, if we land on the home page after clicking on the button, the method name would be "**goToHomePage()**".

What is Page Factory?

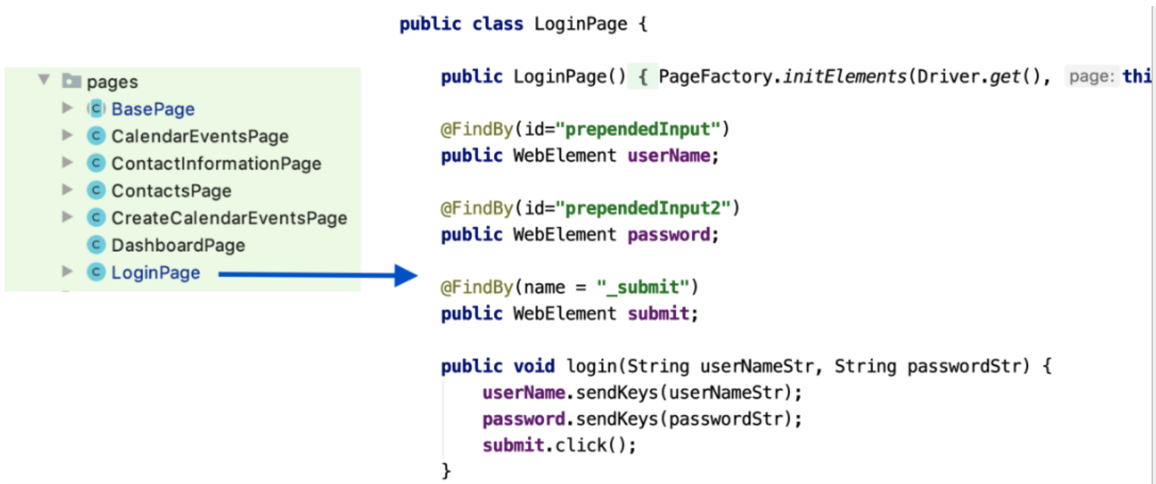
- Page Factory is an inbuilt Page Object Model concept for Selenium WebDriver but it is very optimized.
- We follow the separation of Page Object Repository and Test methods.
- Thanks to the Page Factory class, we use annotations @FindBy to find the elements and initElements method to initialize the web elements.
- @FindBy can accept **tagName**, **partialText**, **name**, **linkText**, **id**, **css**, **className**, and **XPath** as attributes.

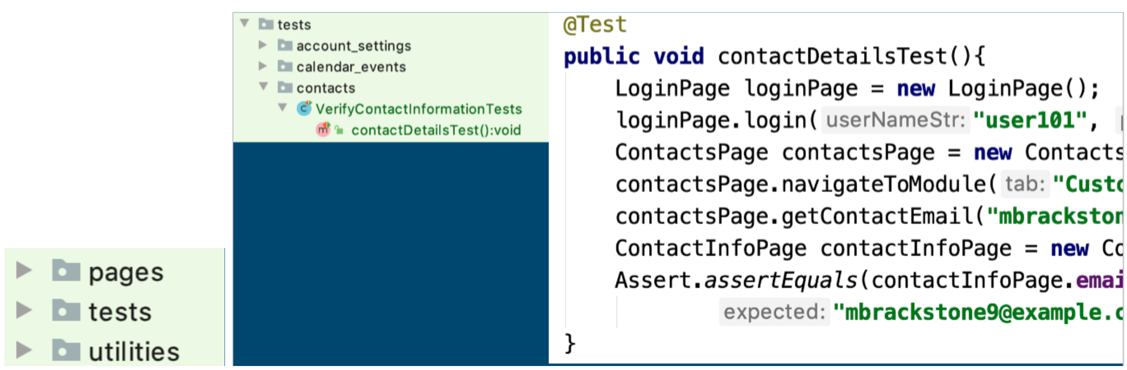
❖ **Syntax:**

```
@FindBy(css = "[title='Create Calendar event']")
public WebElement createCalendarEventBtn;

public BasePage() {  PageFactory.initElements(Driver.getDriver(), this);        }
```

How did you implement the POM in your framework?

I work on a fleet management application. I have the loginPage, LandingPage, and passwordPage, and I also have different menu options. In each page class, I store the web elements and their actions/ methods. For example, in the Vehicles Page, I have the elements for that specific page and also their actions like save & close, create new vehicle, etc. I separate all test cases from the elements and methods.





JUnit & TestNG

What is TestNG?

- TestNG is a testing framework inspired from JUnit but introducing new functionalities that make it more powerful and easier to use. TestNG helps you to generate test reports as well.
- TestNG can act as your Java compiler, you don't have to include the main method since TestNG can run your test codes.
- Thanks to **annotations** in TestNG, you can **group** your test cases using the **xml** file.
- TestNG has a builtin **html report**.
- TestNG can pass **parameters** from the xml file.

What are the advantages of TestNG?

- TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration, etc.
- Managing test suites and test cases.
- Grouping and Prioritizing tests.
- Reporting.
- Parallel execution.

What are assertions in TestNG?

- TestNG Asserts helps us to verify the conditions of the test and decide whether the test has **failed or passed**. A test is considered successful ONLY if it is completed without throwing any exception. There is a difference between **Soft Assert** and **Hard Assert**.
- **Hard Assertion:**
  A hard assertion does not continue with execution until the assertion condition is met. Hard assertions usually throw an Assertion Error whenever an assertion condition has not been met. The test case will be immediately marked as Failed when a hard assertion condition fails. A scenario to use this kind of assertion is that, when you want to verify if you have logged in correctly and fail the test if you haven't made a successful login, there is no point in proceeding further if the pre-condition(login) itself fails.
- **Soft Assertion:**
  A soft assertion continues with the next step of the test execution even if the assertion condition is not met. Soft Assertions are the type of assertions that do not throw an exception automatically when an assertion fails unless it is asked for. This is useful if you are doing multiple validations in a form, out of which only a few validations directly have an impact on deciding the test case status.

What is the difference between TestNG and JUnit?

- Annotations:
  JUnit : @Test, @BeforeClass, @AfterClass, @Before, @After, @Ignore
  TestNG: @Test, @BeforeTest, @BeforeClass, @BeforeSuite, @BeforeMethod, @AfterTest, @AfterClass, @AfterSuite, @AfterMethod
- Both are testing frameworks to help us to run automation scripts.
- TestNG provide **html report**
- TestNG has **@Dataprovider** annotation same as Cucumber Scenario Outline for Data Driven Testing.
- In TestNG, we can do parallel testing, but JUnit doesn't support parallel testing, so we use **sauceLab/SeleniumGrid** for it.
- TestNG support **group test** but JUnit doesn't support
- TestNG and JUnit both of them have parameterized testing but TestNG parameterized test configuration is very easy to configure.
  There are two ways to achieve **parameterization** in TestNG;
  - @Parameters and TestNG xml file
  - @DataPrivider

S.N.	Description	TestNG	JUnit 4
1	Test annotation	@Test	@Test
2	Executes before the first test method is invoked in the current class	@BeforeClass	@BeforeClass
3	Executes after all the test methods in the current class	@AfterClass	@AfterClass
4	Executes before each test method	@BeforeMethod	@Before
5	Executes after each test method	@AfterMethod	@After
6	annotation to ignore a test	@Test(enbale=false)	@ignore
7	annotation for exception	@Test(expectedExceptions = ArithmeticException.class)	@Test(expected = ArithmeticException.class)
8	timeout	@Test(timeout = 1000)	@Test(timeout = 1000)
9	Executes before all tests in the suite	@BeforeSuite	n/a
10	Executes after all tests in the suite	@AfterSuite	n/a
11	Executes before a test runs	@BeforeTest	n/a
12	Executes after a test runs	@AfterTest	n/a
13	Executes before the first test method is invoked that belongs to any of these groups is invoked	@BeforeGroups	n/a
14	run after the last test method that belongs to any of the groups here	@AfterGroups	n/a

How do you cross/multi browser testing in TestNG?

- Inside the suite there are 3 keys (name, thread count, parallel) and I created 2 different tests, one of them is for Chrome and the other one is for Firefox.
- There is also parameter annotation and include name and value; name is browser and value is Chrome.
- Below the first picture is a combined .xml file and TestBase for multi/cross and parallel testing. We have two tests in the same xml file and have two browsers (chrome and firefox) which will help us to run the tests against the defined browsers.

```xml
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="WebOrders Automation Regression Tests" verbose="2"
parallel="tests">

    <test name="Chrome tests">
        <parameter name="browser" value="chrome"></parameter>
        <packages>
            <package name=".*"></package>
        </packages>
    </test>

    <test name="Firefox tests">
        <parameter name="browser" value="firefox"></parameter>
        <packages>
            <package name=".*"></package>
        </packages>
    </test>

</suite>
```

```java
public abstract class TestBase {

    protected WebDriver driver;
    // @parameters  --> means that this method will expect an arg
    // from the xml file
    // name of the parameter in the xlm file is browser
    // @optional --> we use this if we dont want to
    // provide parameters in xml all the time

    @Parameters("browser")
    @BeforeMethod
    public void setUp(@Optional String browser) {
        driver = Driver.getDriver(browser);
        driver.get(ConfigurationReader.getProperty("url"));
    }


    @AfterMethod
    public void tearDown() { Driver.closeDriver(); }
```

```xml
<?xml version="1.0" encoding="UTF 8"?>
<!DOCTYPE suite SYSTEM ...>
<suite ...>
   <test name="ChromeTest" ... >
            <parameter name="browser" value="chrome"/>
                     <classes>
                              <class name="testsuite..."/>
                     </classes>
   </test> <!   First Test   >
   <test name="FireFox" ... >
            <parameter name="browser" value="FireFox"/>
                     <classes>
                              <class name="testsuite…."/>
                     </classes>
   </test> <!   Second Test   >
</suite> <!   Suite   >
```

In my framework, “WebDriverFactory” utility class where I define the browsers to be passed from the configuration.properties file (**browser**=**chrome**) through “ConfigurationReader.java” which connects two of them.

```properties
browser=chrome
url=https://qa3.vytrack.com
qa1_url=https://qa1.vytrack.com
qa2_url=https://qa2.vytrack.com
qa3_url=https://qa3.vytrack.com
driver_username=User10
driver_password=UserUser123
salesmanager_username=salesmanager115
salesmanager_password=UserUser123
storemanager_username=storemanager57
storemanager_password=UserUser123
```

```java
public class WebDriverFactory {
    public static WebDriver getDriver(String browserType){
         WebDriver driver= null;
        switch (browserType.toLowerCase()){
            case "chrome":
                WebDriverManager.chromedriver().setup();
                driver=new ChromeDriver();
                break;
            case "firefox":
                WebDriverManager.firefoxdriver().setup();
                driver=new FirefoxDriver();
                break; }
        return driver; }    }
```

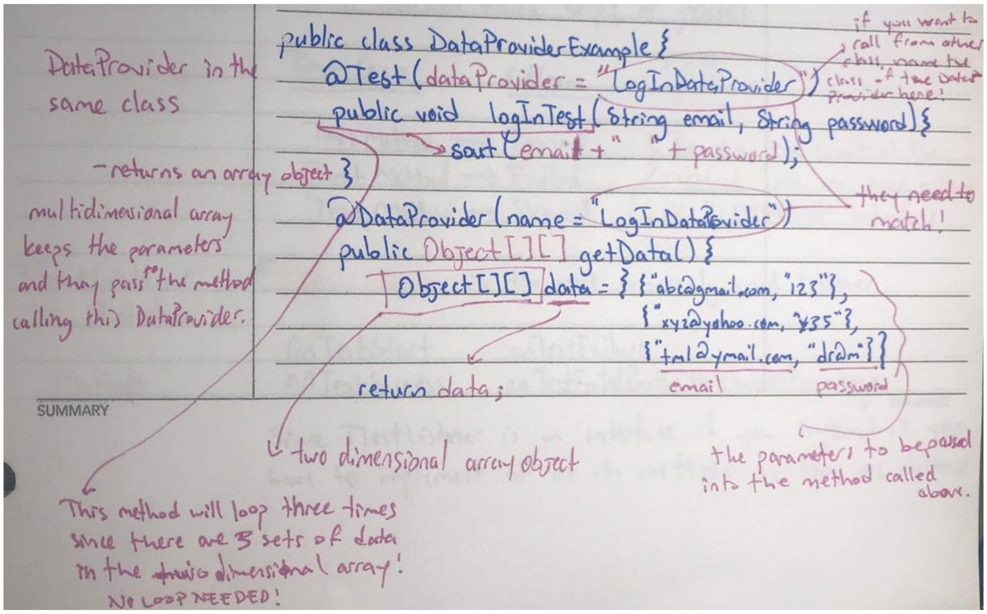
```java
public class ConfigurationReader {
    private static Properties properties;
    static {
        try {
            // what file to read
            String path = "configuration.properties";
            // read the file into java, finds the file using the string path
            FileInputStream input = new FileInputStream(path);
            // properties --> class that store properties in key / value format
            properties = new Properties();
            // the values from the file input is loaded / fed in to the properties object
            properties.load(input);
            input.close();
        } catch (Exception e) {
            e.printStackTrace(); } }
    public static String get(String keyName) {
        return properties.getProperty(keyName); }}
```

In our project, the value of the browser is in two places: xml file and configuration file. We first try to use the value from the xml to create a webdriver object. If no value is passed from the xml file, then we use the value from properties file. All of this logic about deciding which one to use is in the Driver class.

```java
public static WebDriver getDriver(String browser) {


    // String browser ==>  it originally comes from xml file to test base class, from test base


    if (driver == null) {
        // first we check if the value from xml file is null or not
        // if the value from xml file NOT null we use
        // the value from xml file IS null, we get the browser from properties file

        browser = browser == null ? ConfigurationReader.getProperty("browser") : browser;

        switch (browser) {
        case "firefox":
            WebDriverManager.firefoxdriver().setup();
            driver = new FirefoxDriver();
            break;
        case "chrome":
            WebDriverManager.chromedriver().setup();
            driver = new ChromeDriver();
```

How can we create a data driven framework using TestNG?

- DataProvider helps us to pass test data to our test method to read data from configuration file or database at run time.
- DataProvider feature helps you to write data driven tests which essentially means that the same test methods can be run multiple times with different datasets.
- By using **@DataProvider** annotation, we can create a Data Driven Framework
- DataProvider marks a method for supplying the data to other methods.
- Annotated methods return an array of Object, i.e. Object[][] two dimensional array.
- DataProvider can have a name, and it will be used in other methods by using its name.
- It can be implemented in the same class or different classes.



How to create Groups in TestNG?

- These groups are called meta groups.
- Example: you may want to define a group all that includes **regression** and **smoke**. Put the tags on the @Test methods and create a new .xml file for these tests so you can define the suite accordingly.
- When we use groups option, we need to add **(alwaysRun=true)** to @BeforeMethod and @AfterMethod: @BeforeMethod **(alwaysRun=true)**.

```
@Test (groups = {"regression", "smoke"})
public void wrongEmailTest() {
    extentLogger = report.createTest( testName: "Wrong email test");
    extentLogger.info("Entering login information");
    pages.login().login( usr: "admin",  pass: "test");

    String actualError = pages.login().errorMessage.getText();
    extentLogger.info("Verifying error message");
    assertEquals(actualError, LOGIN_ERROR_MESSAGE);
    extentLogger .pass("Passed:Wrong email test");

}

@Test (groups = "regression")
public void wrongPasswordTest() {
    extentLogger = report.createTest( testName: "Wrong password test");
    extentLogger.info("Entering login information");
    pages.login().login(ConfigurationReader.getProperty("username"),  pass: "
```

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Web Orders Regression Tests">

    <test name="Regression">
        <groups>
            <run>
                <include name="regression"/>
            </run>
        </groups>

        <classes>
            <class name="com.weborders.tests.functional_tests.login.ErrorMessageValidationTest"/>
            <class name="com.weborders.tests.functional_tests.order.checkPrice"/>
        </classes>

    </test>
</suite>
```

How to run test cases in parallel using TestNG?

- We can use "parallel" attribute in testng.xml to accomplish parallel test execution in TestNG
- The parallel attribute of suite tag can accept four values:
    - Classes: All the test cases inside a java class will run parallel
    - Methods: All the methods with @Test annotation will execute parallel
    - Instances: Test cases in the same instance will execute parallel but two methods of two different instances will run in different threads.
      <suite name="softwaretestingmaterial" parallel="methods">

Parallel execution in testNG

- In **xml file** write:
  parallel="tests"thread-count="4"
  Thread-count is how many browsers you want to open same time
- In the **xml file** you can add .* to run everything. Ex:
  <package name=".*"></package>

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="WebOrders Automation Regression Tests" verbose="2"
parallel="tests">

    <test name="Chrome tests">
        <parameter name="browser" value="chrome"></parameter>
        <packages>
            <package name=".*"></package>
        </packages>
    </test>

    <test name="Firefox tests">
        <parameter name="browser" value="firefox"></parameter>
        <packages>
            <package name=".*"></package>
        </packages>
    </test>

</suite>
```

```
public abstract class TestBase {

    protected WebDriver driver;
    // @parameters  --> means that this method will expect an arg
    // from the xml file
    // name of the parameter in the xlm file is browser
    // @optional --> we use this if we dont want to
    // provide parameters in xml all the time

    @Parameters("browser")
    @BeforeMethod
    public void setUp(@Optional String browser) {
        driver = Driver.getDriver(browser);
        driver.get(ConfigurationReader.getProperty("url"));
    }


    @AfterMethod
    public void tearDown() { Driver.closeDriver(); }
```

How to ignore a test case in testNG?

- To ignore the test case, we use the parameter enabled = false to the @Test annotation **@Test(enabled=false)**

How to exclude a particular test method from a test case execution?

- By adding the exclude tag in the testing.xml

```
<classes>
   <class name="TestCaseName">
      <methods>
         <exclude name="TestMethodNameToExclude"/>
      </methods>
   </class>
</classes>
```

How to exclude a particular test group from a test case execution?

- By adding the exclude tag in the testing.xml

```
<groups>
   <run>
      <exclude name="TestGroupNameToExclude"/>
   </run>
</groups>
```

What are the different ways to produce reports for TestNG results?

- TestNG offers two ways to produce a report
  - Listeners implement the interface **org.testng.testListener** and are notified in real time of when a test starts, passes, fails, etc...
  - Reporters implement the interface **org.testng.reporter** and are notified when all the suites have been run by TestNG.
- The IReporter instance receives a list of objects that describe the entire test run

What is the use of @Listener annotation in TestNG?

- configure reports and logging.
- widely used listeners : ITestListener interface.
- It has methods like onTestStart, onTestSucess. onTestFailure, on TestSkipped...
- we should implement this interface creating a listener class of our own,
- Next, we should add the listeners annotation (@Listeners) in the class

What Is a Regular Expression, Regexp, or Regex?

- A regular expression is a special text string for describing a search pattern.
- You can think of regular expressions as wildcards on steroids.
- You are probably familiar with wildcard notations such as *.txt to find all text files in a file manager.
- Regex equivalent is.*\.txt.

How to write regular expressions in testing.xml file to search @Test methods containing ”smoke” keyword?

- Regular expression to find **@Test** method containing keyword “**smoke**” is a mentioned below

```
<methods>
   <include name=".*smoke.*"/>
</methods>
```

What are @Factory and @DataProvider annotations?

- **@Factory** executes all the test methods present inside a test class using a separate instance of the class with different set of data
- **@DataProvider** is a test method that uses dataProvider to execute the specific methods multiple times based on the data provided by the dataProvider.

Annotations - priority

- Doesn't matter what number you start Ex: **@Test(priority=0)**
- **DependsOnMethods** = "test method name" You Can add multiple test names
- If the first one fails, the 2nd test won't run at all
- If the first method failed, your report will show that the 2nd test will be skipped @Test(dependsOnMethods={"WebLogIn"})

What is TestBase Class ? and How do you implement it in your framework ?

- **Test Base** class is the class where I have **most used method**s in my tests.
- My test classes **extend** the Test Base class and thus have access to those methods. This helps me us make my **code reusable**
- Before/after test methods wait/synchronization utility methods.
  - SwitchToWindow(title)
  - WebDriver driver;

```
public abstract class TestBase {
// * ExtentReports itself does not build any reports, but allows reporters to access information, which in
// * turn build the said reports. An example of building an HTML report and adding information to ExtentX:
// * ExtentHtmlReporter html = new ExtentHtmlReporter("Extent.html");
// * ExtentXReporter extentx = new ExtentXReporter("localhost");

  protected static ExtentReports extentReports;
  //   The ExtentHtmlReporter creates a rich standalone HTML file. It allows several
  protected static ExtentHtmlReporter extentHtmlReporter;
  //   Defines a test. You can add logs, snapshots, assign author and categories to a test and its children.
  protected static ExtentTest extentTest;

  @BeforeTest
  @Parameters({"test","env_url"})
  public void beforeTest(@Optional String test,@Optional String env_url){
    //location of report
    //it's gonna be next to target folder, test-output folder
    String reportName ="report";
    if(test != null){
      reportName = test;
    }
    String filePath = System.getProperty("user.dir") + "/test-output/" + reportName + ".html";
    extentReports = new ExtentReports();
    extentHtmlReporter = new ExtentHtmlReporter(filePath);
    extentReports.attachReporter(extentHtmlReporter);
    extentHtmlReporter.config().setReportName("Vytrack Test Results");
    //system information
    String env = ConfigurationReader.getProperty("url");
    if(env_url != null){
      env = env_url;
    }
    extentReports.setSystemInfo("Environment", env);
    extentReports.setSystemInfo("Browser", ConfigurationReader.getProperty("browser"));
    extentReports.setSystemInfo("OS", System.getProperty("os.name")); }

  @AfterTest
  public void afterTest(){
//       Writes test information from the started reporters to their output view
    extentReports.flush(); }
  @BeforeMethod
  @Parameters("env_url")
  public void setup(@Optional String env_url){
    String url = ConfigurationReader.getProperty("url");
    //if name parameter was set, then use it
    //if it's null that means it was not set
    if(env_url != null){
```

```
        url = env_url; }
      Driver.get().get(url); }
  //ITestResult class describes the result of a test. (in TestNG)
  @AfterMethod
  public void teardown(ITestResult result){
    if(result.getStatus() == ITestResult.FAILURE){
      extentTest.fail(result.getName());
      extentTest.fail(result.getThrowable());
      try {
        //BrowserUtils.getScreenshot(result.getName()) - takes screenshot and returns location of that screenshot
        //this method throws IOException (which is checked exception)
        //any checked exception must be handled
        extentTest.addScreenCaptureFromPath(BrowserUtils.getScreenshot(result.getName()));
      } catch (IOException e) {
        //print error info
        e.printStackTrace(); }
    }else if(result.getStatus() == ITestResult.SKIP){
      extentTest.skip("Test case was skipped : "+result.getName());      }
    Driver.close();   }
```

How to rerun the failed tests again in TestNG?

- In my TestNG framework, failed tests are reported in the testng_failed_.xml file in the target folder.
- We can add this file in the pom file so that maven will try to run the failed tests every time.
- It will run only if there are failures in the test.

TESTNG FRAMEWORK

- Test automation framework is an environment where we write and execute tests.
- Test automation frameworks use different tools, designs in order to efficiently **generate**, **run** and **report** automated tests.
- In frameworks we use different packages, folders to make them more efficient, understable, scalable etc…
- Types of test automation framework:
    - data driven framework: test and data different, execute same test with different input
    - keyword driven framework: in excel we have the test steps and also the test data.
    - behaviour driven framework will talk about it when we do cucumber
    - component based framework: tests are organized in packages based on component/module
    - page object model based framework uses page object model
    - hybrid driven framework uses 2 or more of the types above

MAVEN

What is Maven?

- Maven is a very powerful **project management** tool that is based on POM (Project object model). Maven is a tool that can be used for building and managing any Java based project.
- It is used for project build, dependency and documentation.

What does Maven do?

- Building/creating a project.
- Adding jars and other dependencies of the project.
- Maven provides project information: log document, dependency list, unit test reports, etc.
- We can run tests.
- Create reports.
- Integrate our project with source control systems such as Subversion or Git.

Why Maven?

- **Central Repository to get dependencies:**
  You don't have to download jars to your computer as maven will take care of it for you.
- **Maintaining common structure across the organization:**
  If we use maven, everyone in the team will follow the same template given by maven. Otherwise, each team would have their individual project folders. You will have a common structure with maven.
- **Flexibility in Integrating with CI tools:**
  Maven can easily integrate with CI tools like Jenkins.
- **PlugIns for Test Framework Execution:**
  Mave has plugins supporting tools like TestNG and JUnit which help you to use all features of TestNG in maven as well.

Maven Commands?

- **mvn clean:**
  Clears the target directory into which Maven normally builds your project. It is recommended to run **clean** previously built executions.
  C:\users\adurhan\Mavenjava>mvn clean
- **mvn compile:**
  Assume that you have hundreds of test cases, it will be really difficult to find **syntax errors** out of hundreds of lines of codes. The **Compile** command will **NOT** run the test but **check the syntax errors.** Compiles the source code of the project.
  C:\users\adurhan\Mavenjava>mvn compile
- **mvn test:**
  Runs the tests against the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed. When you run "test", it will check for the local JARS and if they are in the folder, it will run the test BUT if it cannot find the JARS, it will download the JARS from the repository of Maven.
  C:\users\adurhan\Mavenjava>mvn test -Dcucumber.options="--tags @Smoke"

What is the Maven LifeCycle?

- ❖ Commands can only run in the same directory where the specific **pom xml file** is located.
- ❖ There are 3 builtin build lifecycles.

  Clean : Handles project cleaning.
  Default : Handles your project deployment.
  Site : Handles creation of project's site documentation.

Clean Lifecyle
pre-clean
clean
post-clean

Default Lifecyle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	package
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	install
generate-test-resources	deploy
processs-test-resources	

Site Lifecyle
pre-site
site
post-site
site-deploy

How do you run your tests in Maven?

- ❖ I run my tests in Maven using the **mvn test** command. I need my project to be run from Maven because **Jenkins** runs the tests using Maven.
- ❖ Sometimes when I run locally I use **mvn verify** because I use a plugin in my **pom file** that generates reports to verify lifecycle.
- ❖ When I run in Jenkins, I use the command **mvn test**. I don't need to say **verify** here because Jenkins creates reports using its own plugin, it does not use the plugin in the pom file.

Apache Log4J2

- ❖ Log4J is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License. For Selenium, it is used for logging purposes.

Why Log4J?

- ❖ You store **the entire logs of the file**, keep the logs of all actions and activities such as *clicked on the button, successfully sign-in, selected the checkbox, etc.*
- ❖ You can get **only failed logs** from one package and also all logs from the other with log4J's **flexibility**.
- ❖ You log everything with **Timestamps** which means every activity will be stored with the time occurred so you will know it happened.
- ❖ If your client asks you to get the logs from previous weeks (**history**), you can find them.
- ❖ We can **debug** the issues that we may have with the application.
- ❖ You can use log4J on any Java written testing tool: Selenium, Appium, REST API.

Components of Log4J

- Log4J has three main components:
    1. **Loggers**: Responsible for capturing logging information.
    2. Appenders: Responsible for publishing logging information to various preferred destinations.
    3. Layouts: Responsible for formatting logging information in different styles.

Log4J Configuration: where to log, how to log, and what to log?

Where to log? : Appenders Tag _Console
<AppenderRef ref="Console"/>
<AppenderRef ref="File"/>

How to log? : Appenders Tag_PatternLayout

What to log? : Loggers level - when you change the level, you can define what to log
<Root level="error"> logs the errors only
<Root level="trace"> logs everything; all the text

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
 <Properties> this is to make it independent of any machine to avoid hard coding.
         <Property name="basePath">./src/logs</Property>
 </Properties>
 <Appenders>
  <RollingFile name="File" fileName="${basePath}/prints.log"
filePattern="${basePath}/prints-%d{yyyy-MM-dd}.log"> this is the path of the file and its name
         <SizeBasedTriggeringPolicy size="500"/> this is for the size of the file
  </RollingFile
<Console name="Console" target="SYSTEM_OUT">//where to log
   <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/> //how to log:
  </Console>  </Appenders>
 <Loggers>                     "Additivity" is to avoid duplicate printing in the report
  <Logger name="alpha.Demo" level="trace" additivity="false">
         It will log everything on the this class: "alpha.Demo"
   <AppenderRef ref="Console"/> this can also be "File"
  </Logger>
  <Root level="error"> //what to log: It will log everything on the root level
   <AppenderRef ref="Console"/> this can also be "File"
  </Root>
 </Loggers>
</Configuration>
```

When to Use log. Error, debug and info methods in Selenium test cases

- Use **log. Error()**
  to log when elements are not displayed in the page or if any validations fail
- Use **log. Debug()**
  When each Selenium action is performed like click, SendKeys, getText()
- Use **log.info()**
  When operation is successfully completed ex: After loading page, or after any successful validations. It's just a counterpart to **log. Error().**

Where and How to use Log4J?

I use it in my BrowserUtils class under many different methods. Two examples are:

```
private static final Logger logger = LogManager.getLogger();
1: public static void waitPlease(int seconds) {
        try {
          Thread.sleep(seconds * 1000);
        } catch (Exception e) {
          logger.error(e);
          System.out.println(e.getMessage());        }      }
2: public static void verifyElementDisplayed(WebElement element) {
        try {
          assertTrue("Element not visible: " + element, element.isDisplayed());
        } catch (NoSuchElementException e) {
          e.printStackTrace();
          logger.error(":::Element not found:::");
          Assert.fail("Element not found: " + element);        }      }
```

JIRA

What is JIRA?

- It is a project management tool to **track defects/bugs.**
    - Allows for planning and time management
    - Tracks due dates/assignment
- Tester only in backlog, and active sprints.

For bug tracking what tools do you use?

- JIRA treats all work inside it as an Issue
- So, in JIRA to create a defect would be to create an issue of the type “Bug”.
- Defect reporting :
    - Defect ID
    - Defect title
    - Defect description (steps to reproduce)
    - Environment information
    - Screenshot(attachment)
    - Severity
    - Assign it to Developer

What is an Active Sprint Board?

- **Workflow**:
  To do>In Progress(can have blocked in here too)>Review(Tech lead review the code before moving to done>Done
- **Blocked**: Go to story and click on options and click on more options> blocked
    - Write a comment about why it is blocked
    - Scrum master will have to deal with ASAP
    - Nothing should be in the block for more than a day
    - Once it's fixed you can change to blocker resolved

What are the Jira terms?

- Issue → We need to do and fix
- Types of Issue
    - Story
    - Task
    - Bug
    - Epic

What's the difference between epic and tickets?

- Epic are written by B.A, tickets are created by testers
- Description box
    - Example reporting a bug
    - You write in the box
        - What the bug is about
        - What functionality is breaking
        - What are the steps of recreating the bug (with necessary data)
        - Attach report and screenshot of bug
        - Expected results
        - Actual results

How do you automate User Stories from JIRA?

- Look at the description - Agile story
- Create feature file and save file as Jira story.feature
    - Add scenario located in Acceptance criteria
- Run cukesRunner with dryRun=true
- Implement the methods
- *BEFORE AUTOMATING THE TEST CASES IN JIRA ALWAYS MANUAL TEST IT FIRST*

How do you integrate Selenium with Jira?

- Selenium does not have a built-in integration with Jira.
- But there are **plugins** that integrate selenium testing framework with Jira.
    - **Xray** (Jira plugin, Jenkins plugin)
    - Zephyr (Jira plugin)

Jira Xray integration with Jenkins:

I was not responsible for any configuration, the DevOps team did take care of it.

1. Download xray plugin for Jenkins. You will need to download this plugin from Xray website
2. Install plugin
3. Provide jira information in jenkins settings
4. Add jira xray into the jenkins job as post build action

How xray maps your cucumber test scenarios with jira tests?

- Xray plugin knows your scenarios based on the tags you add at the top of them. For example, I have a "@CT25-19" tag for *"Verify Dashboard Page"* scenario, this tag is also used in the Jira so Xray can map it out.
- Each scenario is a test in Jira; each feature file is a test set in Jira since there are more than one tests in a feature file.

This is from your **feature file:**

```
@smoke_test @CT25-22 @CT25-21
Feature: Smoke test

  Background: open login page and login as store manager
    Given user is on the login page
    Then user logs in as store manager

  @CT25-20
  Scenario: Verify dashboard page
    And user verifies that "Dashboard" page subtitle is displayed

  @CT25-19
  Scenario: Verify Manage Dashboards page
    And user navigates to "Dashboards" then to "Manage Dashboards"
    Then user verifies that "All Manage Dashboards" page subtitle is displayed
```

This is from your **Jira account:**

Create Test | Add Tests

Show 10 entries   Columns

	Key	Summary	Status
1	CT25-20	Verify Accounts page	TO DO
2	CT25-19	Verify Vehicle page	TO DO
3	CT25-17	Verify dashboard page	TO DO
4	CT25-18	Verify Manage Dashboards page	TO DO

Showing 1 to 4 of 4 entries   First Previous 1 Next Last

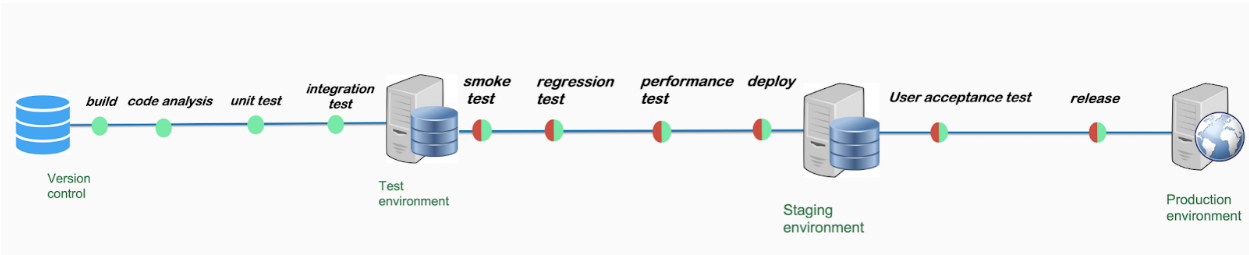
1. Create a test on Jira.
2. If needed, we can create a test set. Keep in mind that one jira test/ticket is used for only one scenario or scenario outline.
3. Export test/test set as a cucumber feature file.
4. Import that feature file into your project.

JENKINS

What is Pipeline?

Pipeline is a set of processes that take the code from version control and compile, build, test and deploy to production in automated fashion. The pipeline breaks down the software delivery process into stages. Each stage is made of different tasks which can be carried out in parallel. When all tasks in a stage passes, the next stage is triggered.

Production Pipeline:



- Every company has to have **4 environments:**
  dev → qa → stage → production
  - **dev:** reserved for developers, once changes are committed, devs can build a new version of software and deploy it to the dev environment. Unit testing is done here.
  - **qa:** this environment is reserved for the testers. Smoke testing is done here.
  - **stage:** preparation for the release, this is where performance testing and UAT testing can be done before release. Regression testing is done here.
  - **production:** environment of **end-user;** real data!

- Every environment has its own database and all of them, except **PRODUCTION**, have dummy/fake data.

What is Continuous Integration (CI)?

Continuous Integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project. The CI process consists of automatic tools that assert the new code's correctness before integration. It reduces integration problems allowing to deliver software more rapidly by providing quick feedback everytime new code is added to the source control. Usually, CI does not involve testing the functionality of the application.

What is Continuous Delivery?

Continuous Delivery is an automated build and execution of unit and integration tests, performing code analysis, functional tests and also deploying to any supported platform any time. Each time a build or a set of code passes the tests, it's automatically deployed out to a staging environment. In Continuous Delivery **releasing to end users is a manual process.** Continuous delivery involves **human decision-making** when it comes to deciding when to release the software to the customers.

What is Continuous Deployment?

Continuous deployment means that every change that you make, goes through the pipeline, and if it passes all the tests, it **automatically gets deployed into production**.When a developer checks in code, the automated processes take the code and move it through the entire lifecycle and if it passes each gate, it gets deployed directly to production. The delivery speeds are notably faster due to elimination of manual steps.

What is Jenkins?

- It is a Continuous Integration / Continuous Deployment tool written in Java.
- It is used to build and test your software projects **continuously** making it easier for developers to integrate changes to the project, and making it easier for users to obtain a **fresher / up-to date** product.
- You can automate the build and testing, and also schedule it.
- Jenkins has 3 components:
    - **Code Change**: Devs makes changes to the application code
    - **Test**: CI tool automatically picks up the changes and tests the application
    - **Deploy**: CI tool deploys the application with changes

What is Jenkins Jobs?

- In Jenkins, everything is done by creating a **job**
- It is a task that Jenkins performs based on its schedule which set by DevOps or by the testers depending on the company's culture
- It includes several steps such as **pulling** the **code**, **running** the **tests**, **generating** the **reports**, sending the **email notifications**, etc.
- It can have **triggers** that determines when it runs

What's the difference between Jenkins Job and Pipeline?

- Pipeline is a more advanced version of Jenkins job. To declare Jenkins pipeline, you need to use DSL that is based on groovy. With a pipeline, you have more freedom. You can parallelize your tasks. Pipeline looks like a script and usually saved as a jenkins file. Pipeline can be used to develop CI/CD pipeline or to develop smoke/regression jobs for testing.
- Pipeline is done by **Devops** or **Senior Developers** so as testers we **DON'T** get involved with this process and only work with Jenkins jobs. Not all companies do Pipeline, so no need to mention in the interview.

How do you use Jenkins in your project and create a Smoke Test?

- In my project, we use Jenkins for CI/CD and we have Jenkins jobs for regression and smoke tests. *We usually don't have access to Jenkin configuration which is done by Devops but here are the steps:*
    1. Pull the code from repo (GitHub)
    2. Specify the maven goal - command - run "mvn test"
    3. Generate the cucumber HTML report
    4. Send email notifications / slack notifications
    5. Publish tests results on Jira with Xray (optional)

- Our **Smoke test** has 2 triggers, which helps to automate the process:
    - First trigger is based on **time:** The smoke test runs everyday **at 5am.**
    - Second trigger is based on a **new build**: Once the new build has been deployed, smoke test triggers automatically to make sure everything is up and running.
- In my company, we have Smoke Tests only. I know some other companies have "sanity tests" which are kinda a smaller scale of smoke tests but my company implements only smoke testing.

How do you maintain Jenkins and who sets it up?

- In my company, we have the DevOps team which is in charge of setting and maintaining Jenkins for deployments.
- DevOps work with developers to create jobs for building and deploying the application.
- For regression and smoke tests, they work with the automation testers who provide the information of the tests, configuration info, and also the emails for the notifications.
    - Git path
    - Mvn code; goals - compile, or verify -Drunner=xml, etc.
    - Times to schedule certain tests

Who sets up Smoke Tests in Jenkins?

- I work with the DevOps team to create my smoke tests in Jenkins.
- To run my tests, I need certain plugins to be installed on Jenkins, I also need java, maven configured on jenkins, I also need browsers installed on the server where the tests will run.
- Since only DevOps team members have the right to do the configurations in Jenkin, I work with them for certain setup or configuration.
- Once the configuration of the Jenkins is completed, I can create and run the **smoke test job** (Jenkins Job).

How do you create Jenkins Job for your Smoke Tests?

1. Click on New Item
2. Enter name: smoke_tests *(use unique name and no space as convention)*
3. Select freestyle project
4. Click **OK**. *(if not clickable, refresh the page)*
5. Pull the code from the **GitHub**. You need to get the project's URL and paste it to Jenkins.
6. Specify **Build Triggers** to define the times, days, or dates that you want the test to be run.
7. Go to **Build** → click on **"add build step"** and select **"invoke top-level maven targets"** to be able to execute maven life-cycle commands like test, verify, install, deploy, etc.
    Enter command: **clean test -Dcucumber.options="--tags@smoke_test"**
    Or for **regression** enter: **clean test -Dcucumber.options="--tags@regression"**
8. Go to **"Post Build-Actions"** and click **"add post-build actions"** and select **"cucumber report."** *(This is available because we have installed the cucumber html reports plugin to our jenkins. Cucumber reports plugin will generate html reports for every build. Every time we run smoke tests on jenkins, we get a new report. and all reports for all builds are saved.)*
9. Go to **"Post Build-Actions"** and click **"add post-build actions"** and select **"Editable Email Notification."** *(Here I configure it so that everyone in my agile team is notified about the test results.)*
    Remember: Change email trigger to "ALWAYS"
10. Save and click Build Now!

What kind of test can be done in Jenkins?

- Jenkins can run any automated test.
- Unit tests; smoke tests; integration tests; regression tests; sanity tests, etc.

What layers of tests can be tested using jenkins?

- Jenkins does not care if we are testing UI or DataBase or API. it only kicks off the tests and sends reports.
- Testing different layers (UI, DataBase, API) of the application is done with our test code.
- If my automated test is a UI test, it means Jenkins is running UI tests, or if it is an API test, Jenkins is running API tests.

How Is the Code Deployed To Your Environment?

- Devs write the code, test it then it is deployed in jenkins from **dev** to **test** environment
- What if it doesn't?
    - I talk to the developer and ask them to deploy it.

How do you schedule a build in Jenkins?

- In Jenkins, under the job configuration we can define various build triggers.
- Simply find the **'Build Triggers'** section and check the **'Build Periodically'** checkbox.
- With the periodically build, you can schedule the build definition by the date or day of the week and the time to execute the build.
- The format of the 'Schedule' textbox is as follows:
    - MINUTE (0-59), HOUR (0-23), DAY (1-31), MONTH (1-12), DAY OF THE WEEK (0-7)
- How do you do scheduling in Jenkins? How will you schedule the test to be executed every 3 hours?
    - H 3*** If you want to schedule your build every day at 7h00, this will do the job : 0 7 * * *

Jenkins Cucumber Report

- Jenkins has the Cucumber report plugin that can generate User Friendly reports.
- Only data Cucumber report plugin needs is the **Json format report**.
- Once the Cucumber report is generated, send the URL of the report to anyone who asks for the report.
- All the history of automation tests are stored in Jenkins. It shows dates and hours and other details so if needed you can revisit previous reports if needed.
- If you need to see the automation report from the past, you find it in Jenkins.

How many jobs and tests do you have on Jenkins?

We have one smoke test and two regression tests in my company:

- Smoke Test:
  - I have two smoke test triggers:
    First trigger is based on **time:** The smoke test runs everyday **at 5am.**
    Second trigger is based on a **new build**: Once the new build has been deployed, smoke test triggers automatically to make sure everything is up and running.
  - We have about 80 smoke test cases which takes about 15mins
- Regression Test:
  - At the end of every 3 sprints or when there is a major bug
  - Almost 600 test cases (200+ are written by myself)
  - Takes about 1/1.5 h with parallel testing
- How:
  - Tags for Regression and Smoke
  - Jenkin kicks the regression
  - Jenkin generates HTML report with detailed test steps and screenshots
  - Analyze the test results
  - If it fails because of my code I fix my code. (Perhaps during that time, the application was down, and I ran my script at the wrong time.) If there is really a defect, I log the defect and test it again until it is fixed.

What was your biggest achievement in your company?

- I integrated Jenkins with slack. To do this, you just need to download the slack plugin --> in settings provide slack api key and workspace info --> then we need to select slack plugin as post build action.

What was your biggest challenge?

- Understanding the framework was my biggest challenge: When I first started the job, unfortunately I was not introduced to the project very well and whenever I had any questions about the framework, I did not have anyone who could help me enough since the person who developed the framework had already left.
- For example, I had cases where there were no page classes for some of the objects. I had the step definitions but could not locate the page classes. Since nobody knew it very well, I had to figure that out myself. I created a page class for each feature file and named them with their functionality so whoever reads it could have an idea what it is for.
- I had to do a lot of debugging to understand the code flow; see where all test data is coming, etc.
- The main problem was because there was not a well written documentation file. It took almost about two weeks to figure things out. Now, we have a better documentation file.

Git and GitHub

What is Git and GitHub?

- Git is a distributed version control system.
- GitHub is the tool providing hosting for software development version control using Git.
- GitHub keeps track of new/old versions of documents.
- Git is a VCS. GitHub is hosting Git repositories.

What is a repository?

- Group of files that VCS (version control system) tracks.
- Folder where the files are saved and it may contain single, collections of files, or single project.

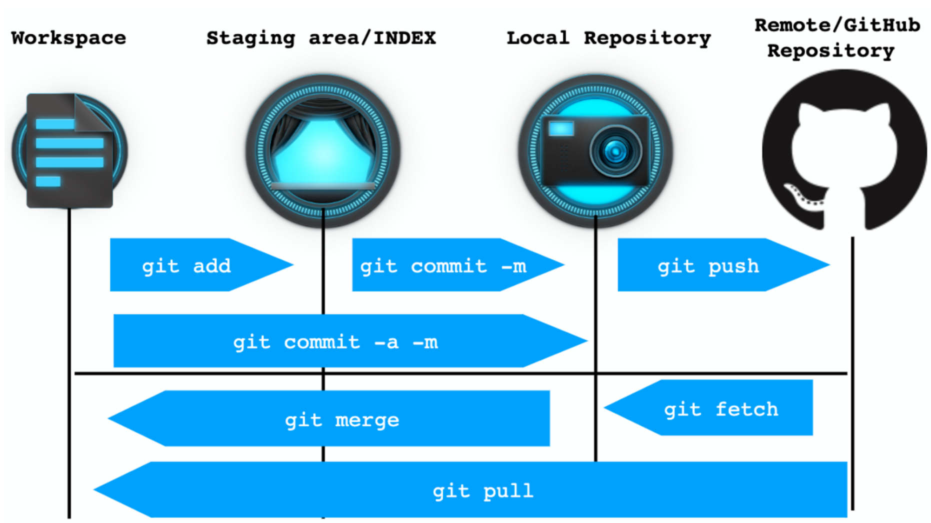
What is Remote & Local Repository?

- **Remote Repository:**
  Host on server(GITHUB) Our changes go from local to remote repo
- **Local Repository:**
  Typically, on your computer -Our changes are done here consist of Working Directory, index andHEAD

What are the benefits of VCS?

- Collaboration on the project
- Tracking the changes:
  version 1 → version 2 → version 3 → version 4
  You will know what changes were done in every version of your project. You can revert them if needed.

Git Workflow: Workspace → Staging → Local Repo → Remote Repo



What are the Git commands?

git init	**:** to create a local repo
git add.	**:** to add all files to the staging area (also it calls index) . (dot) is to add all files - git add **src/** to add specific file
git commit -m "*your text here*"	**:** to commit your code to the local repository which is on your computer. : **-m** means the commit message in the **" "**
git commit -a -m ""	**:** to add all files and commit, so you don't need to run git add src/ before commit
git remote add origin	: https:/github.comAliDurhan/MyGitProject.git This is to connect your **local** repository with the **remote** repository on the GitHub. We have to connect ONCE!
git push -u origin master	: to push the code to the remote **master** branch. By default, there is only one branch, master. You do this only ONCE and than just **git push** Then, every following commit will be sent to the master branch unless you **switch to** branch.
touch .gitignore	: to create git ignore file which contains the list of files that you don't even want to commit at all. For example, intelliJ config file or target folder. *You can also add **.ignore** file manually but need to make sure that it is created under the project.*
git push	: to send your code to the remote repository
git pull	: to get the latest and the most updated version of the project. *I pull at least twice a day; in the morning and at the end of the day.*
git status	: shows what branch you're on and any changed files that aren't tracked.
git version	: will give the version of git.
git remove -v	: will show the URL of your github repo.
git diff	: will show the changes.

How do you use Git in the terminal?

- create new repo-git
- echo "# SqlMentor" >> README.md
    - git init
    - git add README.md
    - git commit -m "first commit"
    - git remote add origin
    - https://github.com/Andylam224/SqlMentor.git git push -u origin
    - Master
- push an existing repo-git

- git remote add origin https://github.com/Andylam224/SqlMentor.git
- git push -u origin master

What is Git Branch?

- Branches are part of our everyday development process. They are effectively a pointer to a snapshot if your changes.
- It's a one version of your project. With git, we can have multiple versions of the project that exist in parallel. Every branch can have independent commits history.
- When you want to add a new feature or fix a bug (no matter how big or small), you create a new branch to encapsulate your changes. This makes it harder for unstable code to get merged into the main code base, and it gives you the chance to clean up your feature's history before merging it into the main branch.
- You can merge a branch into another branch using a pull request.
- Master branch should have the latest stable and confirmed code. Ideally, the master branch shouldn't accept direct commits. Otherwise, changes can be committed without code review.

Tester's Side Notes:

- In my project, the test lead was review code. But if there is no team lead/test lead then we do peer review. Everyone's code must go through review. But, the test lead can decide/choose the flow. Every commit must be reviewed.
- I would commit my changes, whenever I completed automation of some user story. Then I would add a commit message like: "completed automation of VYT-4321 Driver should be able to add fuel logs". It's not only about scripts, you can also add utilities, fix something… The main idea is that we have to be careful with the master branch.
- Protect master branch from direct commits and enforce pull requests with code review. Code review will improve **code quality**. Also, we will make sure that the master branch has 100% working code.

GIT Branch Common Options:

git branch	: to list all branches in your repository.
git branch <branch name>	: to create a new branch but doesn't check out.
git branch -d <branch name>	: to delete the branch on local.
git branch -D <branch name>	: to **force** delete even if it has unmerged changes.
git branch -m <branch name>	: to rename the current branch to **<branch>**
git branch -a	: to list all remote branches.
git checkout	: to switch to another branch and check it out into your working directory.
git checkout <branch name>	: to switch to the given branch.
git merge <branch name>	: to merge the branch
git push --set-upstream origin <branch name>	: Pushing branch to remote (gitHub WebSite) from local (intellij)

What is a pull request?

- Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.

How do you create a pull request?

- git request-pull [-p] <start> <url> [<end>]

How do you do code review in your company?

- **pull request:** Once I push my code to the remote repo, I create a pull request to merge my branch with the master branch.
- Our test lead along with one of my co-workers reviews the code, confirms the pull request and merges it.

Who is setting up GIT in your project?

- Our DevOps team is responsible for setting up and configuring tools like GIT and Jenkins.

How often do you clone and pull the project at your work?

- Before committing my code, I **pull** to avoid any conflicts because someone might have committed his code so it means I may not have the most updated version of the project. I pull the project before each commit to avoid this issue.

What is an untracked file(s)? - brand new code

- It is the file that was not added to the staging area.

How do you check if you're connected with the remote repo?

git remote -v

Potential Issue and Fixing it

If you run command: git commit **without** -m "*your text*", the system will show a window in the **vim editor** to enter a commit message.

Enter the below command and press ENTER.

:qw	- quit & write
:q	- quit only
:q!	- quit but not save

If you don't know what to do, just press **cntrl + c** to stop/exit anything.

Case Study: Conflict and Solution 1:

Case:
You have changed a file in your local repo and one of your teammates also worked on the same file, MADE some CHANGES and committed that file to the remote repo BEFORE you did it. When you try to pull or commit, you will have a CONFLICT.

Solution:
You can **STASH** the changes which means you can SAVE the changes and REVERT to the previous version.

git stash save "*your message*" : to stash the changes.
git stash list : this will show the history of stashes.

--- Intellij has a conflict tool. When there is a merge conflict, you can fix it with intellij's "**resolve conflict"** tool.

Project → Git → Resolve Conflicts

Case Study: Conflict and Solution 2:

Case:
You add a new file, your teammates add another. When you commit and push your file, since there is a new file in the remote repo added by your teammates, you will have a conflict.

Solution:
It is a good practice to **git pull** first so you have the most updated version BUT in this case since you did not pull first, the system will get into the **vim editor** in the terminal so you will need to exit by clicking **esc** and typing **:q** and hitting enter.

What are the Stash Steps?

1. git stash save *"your message"*
2. git pull
3. git stash pop
4. git add **.** or git add src : *this step is to add the fixed codes*
5. git commit -m *"your message"*
6. git push

What does "repository is clean" mean?

- No untracked or modified files since the last commit.

Merge conflict

It's when multiple contributors (2 people) change the same file. To resolve conflict, we can do couple things:

1. Stash changes (if you didn't finish,and you cannot combine changes, stash your changes. In this case, you are not losing changes forever.)
2. Discard your changes (you are losing your changes forever) either git checkout file or git reset --hard
3. Resolve conflict first of all we have to commit our changes then pull; and check conflicting files (fix manually or through other tools, like in intellij); accept theirs changes; accept yours (your changes); merge changes (compromise); once conflict resolved, we can commit and push changes to github
4. Just clone project again and copy paste your changes (NO, but this is what people usually do)