Describe your Framework and Tests

MY FRAMEWORK and how to explain to interviewer - Andy Lam and little bit of Alex

//based on **Data Driven and Behavior Driven** - Hybrid framework

//**Maven** - build tool and not only for dependency management but also as a command prompt tool using the pom xml file, I also have specific **xml file** that run my smoke,

//regression, and functionality tests

//**Java** as programming language - working frontend, backend(api), and database I use **Java Collection framework** to store data and compare

//i also have a **properties file** that stores sensitive/reusable data - URL, password, browser

//and i use **TestNG** testing tool to control flow and assert data, after formatting data to java, in order to find defects

I also created **Driver class** in utility package that uses **singleton pattern** to create and use only one universal webdriver

FRONT END ; **selenium webdriver**, and in my framework i am using **Page object model** as my design pattern; - create page objects; identify webElements and store as a webelement variable ,

POM = REUSABILITY OF ELEMENT/METHODS BASED PAGE OBJECT

//I also use **Page Factory design pattern** to instantiate my webelements using @FindBy - easier/convenient

//utility; browser util - where static reusable code that makes your life easier, to make codes for automating browser easier;

BACK END (Api)

//using **RESTASSURED** in your framework in order for the process of **Deserialization** and **serialization** to occur

//that way you store **Json response** into a java collection data structure(/i produce high level Pojos and map objects) and assert the data with the expected value (also stored in java data structure)

//i also have a api utility class - reusable codes -method where one line creates a Pojo

//i use **postman** for manual testing first BEFORE I INVOKE MY FRAMEWORK

DATABASE testing

//manual I **sql developer** for producing SQL queries

//FOR AUTOMATION ;I use **JDBC library** to integrate java by getting a **CONNECTION** from oracle database

//then creating **STATEMENTS** using SQL queries and then storing the data into a **RESULTSET** object.

//I use java data structures to use store data inside and compare them

//and since I'm using **DATA DRIVEN and CUCUMBER BDD** framework, all of these tests are stored inside **feature files**

//I have **RUNNER classes** that helps generate codes from **FEATURE FILE** and implement them into a file called b

//also have **HOOK class** that implements my codes that run before and after all my tests - this is where i invoke my **TAKESCREENSHOT** interface which triggers when i use scenario interface(when scenario fails)

//take a picture when you are on the step that failed

//S.D - this is where I stored my codes that based on **gherkin language** expected value

DDT

//if I'm working with small amounts of test data, I'm going to operate with **scenario outlines**, this where I create **examples** and store data using pipeline

//if there are large amount of test data its usually in an external file (excel) so i use **Apache POI** to **INVOKE DDT EXCEL AUTOMATION** and read from excel file and store the data

//into java data structure

//I also have a logging tool called **log4j2** to log my codes that are basically high risk

//and lastly for my reports, in my framework I use **Rerun.txt code** in cucumber "rerun:target/rerun.txt" generated by cucumber sandwich library
//this will store my failed cucumber feature files
//then i also have **failedScenario runner class** which has the location of failed scenarios (rerun.txt)

//i create a failedScenario xml file
//so whenever I have failed feature files I use mvn command ; **mvn -Drunner=failedScenarios. xml** file to run my failed tests
//reporting - I used html report that's located in target folder which is called cucumber-reports -"html:target/cucumber-report"
Parallel testing - used **cucumber jvm-parallel plugin** to generate runners and **maven fail-safe plugin** to run the tests

For **continuous integration (jenkins)**
//devops takes care of configuration
//have github path
//but the tool is invoked my a mvn command - mvn verify -drunner=smoketest.xml that is provided by the tester - xml file
//for reports each build will have a cucumber report that give graphical information of test and screenshot

How many test cases/scenarios/feature files: 23 test cases per sprint, 150 feature files, 400 scenarios My smoke test
2 feature files, 2 scenarios each, 5 mins smoke tests run once a day.

My regression
150 feature files
400 scenarios
3 years ; 1 tester worked 3 years, second worked 2 years

How often is your smoke test, how long does it take and tests what? Every morning at 5am, around 5 minutes, creates 5 students in two browsers with the necessary fields, checks if it is accurately saved from both UI and database, then populate the report from the browser and also download as an excel file and then compare.

How often is your regression test, how long does it take and tests what? Once a week 1am, half an hour, all districts every day, only SIMS. Validation regression tests on the weekends, 5 separate EC2 machines, parallel testing for each school, submission, validation, crossvalidation and certification. Our test validates with SCS.

Sample Scenario: Given the SIMS report is downloaded, and the file type is an Excel File type, then the first column title must be "LASID", and every cell in the first column must be alphanumeric, and every cell in the first column must be 9 digits, and all LASID numbers must be unique.

Sample Test Case: Given the user is logged in, when the SIMS report is executed, then every cell in the first column must be 9 digits, and also alphanumeric.
Must Have: precondition, steps, test data, expected result, actual result

Edge Case Scenario: null, negative numbers, empty list/string, duplicate control, checking the limits, extreme cases (length, size)

Risk based testing: when there is no time to do whole regression testing, you only test the parts that matters, that is related

Testing without requirements: Production defects usually don't have any requirements and I talk to developer to understand the situation better and then test it.

Example for Overloading method: Several overloaded methods in BrowserUtils for waits. Explicit waits by locator or WebElement.
Example for Overriding method: Below

Example for inheritance/abstraction: Top bar and sidebar pages are abstract classes and some of the methods are abstract because we have different implementations based on where they are and where they click it. School/District functions are populated differently on the dropdown menus.