**URL** :[**https://github.com/sanchit097/GurukulaUiAutomation.git**](https://github.com/sanchit097/GurukulaUiAutomation.git)

**Description :**

Functional aspects:

* 5 Scenarios are covered which includes login, new user Registration, create new Branch, edit existing Branch and delete existing branch. Since branch and staff have same CRUD operation so considering the time factor I have only implemented these 5 scenarios.
* Cucumber feature file is created containing background steps for login in application entering URL which is a parameterized in config file.
* Scenario is written in reusable way taking parameters making it reusable by just changing the data set.
* Cucumber test result can also be found in folder Target/cucumber/index.html
* Assertion is used to verify the end results.
* Positive scenarios are covered in this test and for the Registration test case assertion will fail as website is not able to create new registration.

Technical aspects:

* Framework is created with Cucumber and using Data object pattern for selenium Web Driver.
* Resource folder is there to cater feature files for scenarios and drivers along with config files.
* TestRunner class is created which is an entry point for the cucumber test where feature file is stitched with its step definitions.
* In test/java 4 packages are created :
  + stepDefinitions
  + pageObjects
  + utilities
  + enums
* **stepDefinitions**: Contains classes where methods are written which maps with feature file steps.
* **PageOjects**: Package contains each page wise web elements and method which do business logic work with web driver.
* **Utilities**:
  + Contains WebDriverManager class responsible for deciding which browser driver to initialize based on the value given in config file. Also this class is responsible for creating driver object which will be used throughout the test.
  + PageObjectManager class is responsible for passing page object class to step definition class. This class can be used to manage pageOjbect class’s objects for all the pages so that single object can be passed throughout the test.
  + TestContext class is used to pass PageOjbectManager object and webDriverManager to all the other classes.
  + ConfigFileReader class is created to read Config file and pass the driver path and browser type.
  + SeleniumBase class contains all the common method used to identify element or wait for element widely used throughout the automation. Also handles the driver exception so that it will not break the test.
  + Log file is used to pass logger object to all the classes. logging can be implemented to all the step definition to keep track of the test run.
* Hooks class is created inside stepDefinitions which containers @after tagged method to close the Driver object after the test is over.
* No driver related function is written in stepDefinitions class it calls the pageObject class where all the driver related work is done. stepDefinitions is used to keep the step definitions away from business logic and make it reusable.
* Solution contains Chrome Driver inside it so if no browser type is mentioned in config file, by default it will load chrome driver from the path inside it.