**Automation Test Plan /**

**Test Strategy**

Revision History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version Number | Release  Date | Prepared By | Reviewed By | Approved By | Summary of Changes |
| 1.0 |  |  |  |  |  |

Contents

[1.0 Introduction 3](#_Toc11157541)

[1.1 Purpose 3](#_Toc11157542)

[2.0 Test Automation Approach 3](#_Toc11157543)

[2.1 Test Automation Objectives 3](#_Toc11157544)

[2.2 Functions and Features to be automated / automated 3](#_Toc11157545)

[2.3 Functions and Features that will not be automated 3](#_Toc11157546)

[2.4 Deliverables 4](#_Toc11157547)

[2.5 Assumptions 4](#_Toc11157548)

[2.6 Constraints 4](#_Toc11157549)

[2.7 Dependencies 4](#_Toc11157550)

[3.0 Test Automation Framework 5](#_Toc11157551)

[3.1 Requirements 5](#_Toc11157552)

[3.2 Design 5](#_Toc11157553)

[3.3 Standards 5](#_Toc11157554)

[3.3.1 Key Components of Framework: 6](#_Toc11157555)

[3.4 About Framework 6](#_Toc11157556)

[3.5 Test Tools 6](#_Toc11157557)

[3.6 Script Execution: 7](#_Toc11157558)

# Introduction

## Purpose

The purpose of this document is to a high-level test plan / test Strategy on how we are going to perform Functional Test Automation Script Creation for ServiceNow Gurukula applications.

Detailed description for implementing the framework process flow and methodologies have been drafted under individual sections below.

# Test Automation Approach

## Test Automation Objectives

Following are the key objectives:

* Setup Automation framework and Functional Test Automation script creation for Gurukula Application .
* Capability to Integrate Automation scripts with CI/CD.

## Functions and Features to be automated / automated



## Functions and Features that will not be automated

|  |  |  |
| --- | --- | --- |
| **S.No** | **Application** | **Modules and Features will not be automated** |
| 1 | Gurukula | Options under Home🡪Account Menu  Settings, Password, Session |
| 2 | Pagination in Branch page (No links available) |
| 3 | Pagination in Staff page – Time Constraint |
| 4 | Logo Verification in home page |
| 5 |  | Staff creation without branchName – Found this scenario in the last minute |

## Deliverables

|  |  |
| --- | --- |
| **S. No** | **Deliverable** |
|  | Test Automation Plan |
|  | Complete Test Automation Suite (Includes test scripts, Reusable functions and libraries required) |
|  | Test Script Suite  **Functional Automation:**  **16** Test Automation Scripts  Detailed Test Execution & Summary report |

## Assumptions

Following are the assumptions that are made to execute this project.

* Test Environment will be local. If running scripts other than in local environment a separate property file will be created.
* Current scripts will run only in Chrome Web browser.

## Constraints

None

## Dependencies

None

# Test Automation Framework

## Requirements

* Google Chrome
* Selenium
* Gradle
* Java

## Design

* Page Object Model framework being used, where all the necessary functions required by a specific page can be placed individually.
* A class is created to describe the page itself, in terms of elements on it, actions that can be taken on the page, and resultant front-end behavior.
* Actual actions that need to be taken are defined in the test class. This keeps element definitions separate from test actions, thereby making the overall structure and organization of automation scripts more modular and modifiable.
* A base class created for test actions (TestBase.java) that are commonly used across tests and page agnostic. Initializing the driver is an example of this. If required, a similar base class can be created for page classes as well.
* Reports.
  + - All the Reports on success and failure should be displayed in Html Format.
* Test Data
  + - The current framework is using JSON Objects in a separate file to provide the test data to specific scripts.
    - If we need to create scenarios with specific data, new test data needs to be created
    - All the test data which is driving the test case must be contained in the folder “data”.

## Standards

Below standards are followed while test script authoring

* Naming Standards [for scripts, test results, directory structures, and verification point’s help to keep everyone on the same page].
* Coding standards
  + Used indents to make the code more readable
  + Variable names are meaningful.
  + Reusable functions being created to avoid duplicating code or reinventing the wheel

### Key Components of Framework:

* **Property file –** The global settings and application access parameters are maintained in the configuration file. For now only URLs are maintained.
* **Data –** Test scripts will be executed against small data sets available as part of the local environment
  + The current framework is using JSON object to provide the test data to specific scripts.
  + For scenarios for which the test data is required dynamically changing, we are fetching the data from UI and passing the same to scripts. Ex: to Check Delete functionality.
* **Test Scripts –** Test Scripts built based on best of the breed coding practices, with extensive comments, descriptions and maximum reusability
* **Test Results Dashboard –** Result reports are produced in HTML formats.
* **Gradle:** Gradle being used as build tool. All required dependencies, tasks and tests are mentioned in the file *build.gradle*

## About Framework

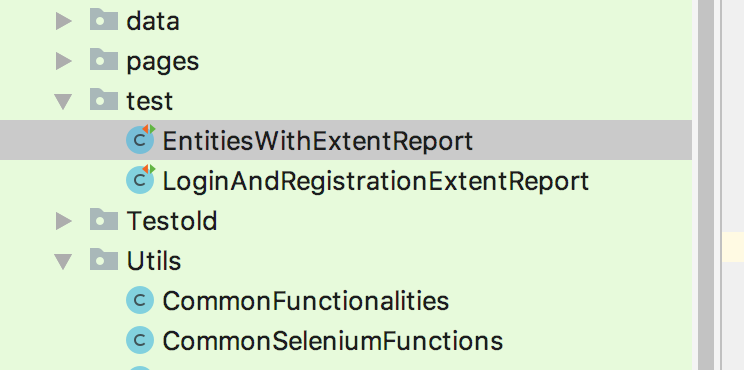
* Folders Structure
  + data – Test data maintained here.
  + pages – since we are using “Page Object model where all the necessary functions required by a specific page can be placed individually
  + tests – Actual test cases are written and executed from here
  + utils – All common, reusable function are placed here
    - CommonSeleniumFunctions : This class contains Selenium related reusable functions and methods in this class can be used globally
    - GenerateReport : This class is used to generate output reports
    - GenericFunctions : This class contains various generic functions, like string related, date etc.,
    - ReusableFunctionalities : This class contains couple of reusable functions like Login and create Branch and Staff data.
    - TestBase: Is used across tests and is page agnostic. Initializing the driver is an example of this.

## Test Tools

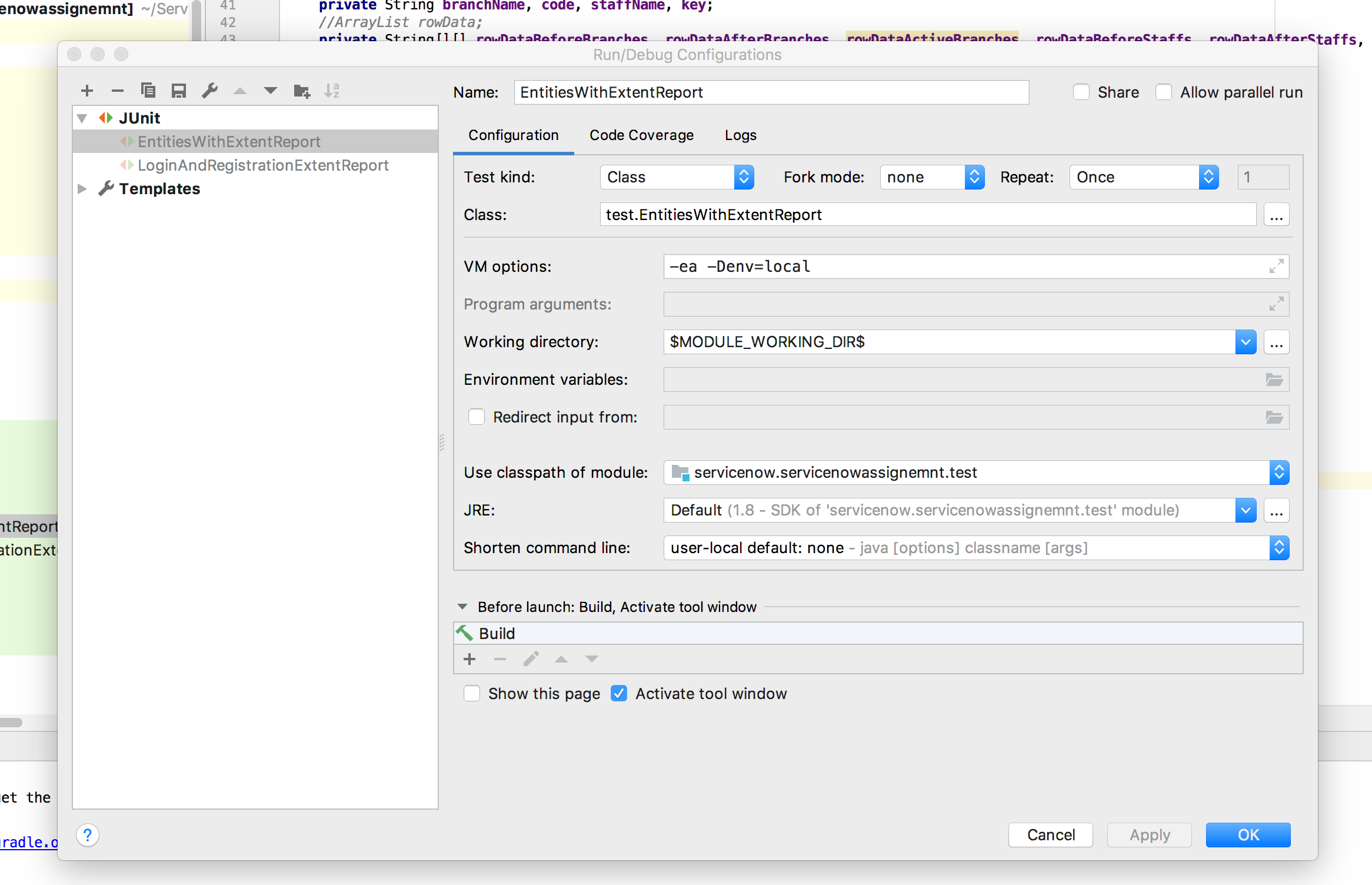
|  |  |
| --- | --- |
| **Test Type** | **Tool** |
| Test Automation (GUI) | Selenium Java |
| Version Control | Git |
| Build tool | Gradle |
| Assertions / Annotations | Junit |

## Script Execution:

* Script can be executed by running class which are in tests folder



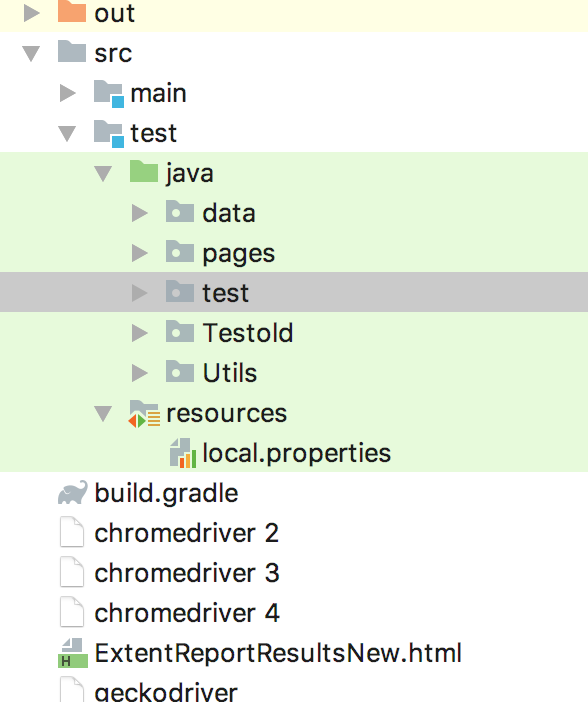
While running from Intellij: Need to set VM option value as -**ea -Denv=local** as shown below and add class we want to run.

Menu🡪Run🡪Edit Configuration

* Script can be executed from terminal by using gradlew command

**./gradlew clean Regression -Denv=local**

* Once Script is executed find results from:
* Extent Report



* + Junit Report

