

**SOFTWARE ARCHITECTURE DOCUMENT**

***Author: Mohamed Salah***

Agenda

1. Introduction
   1. Purpose
   2. Objective
   3. Scope
2. Architecture Diagram & Data Flow
3. Architecture Explanation
4. Test Cases Covered
5. Used Technologies and Tools
6. Improvements To Be Done
7. Configuration And Run Steps

1. INTRODUCTION

Khazna application is becoming widely used now by different entities, people and companies. That means that Khazna’s scope is becoming bigger everyday and that leads to frequent changes in the company’s scope and the platform so here comes the automation testing role. In order to cope with these changes and to maintain the platform high quality we need to apply test automation.

This document provides a high level overview and explains the architecture of the automation done for khazna application

The document defines an overview on the architecture chosen and the covered scenarios and how it helps in testing these scenarios and maintaining the automation code.

1.1 PURPOSE

The Software Architecture Document (SAD) provides a comprehensive architectural overview of the test automation .

1.2 OBJECTIVE

Our system aims to automate the regression suite and the frequently executed test cases also automation helps in the following

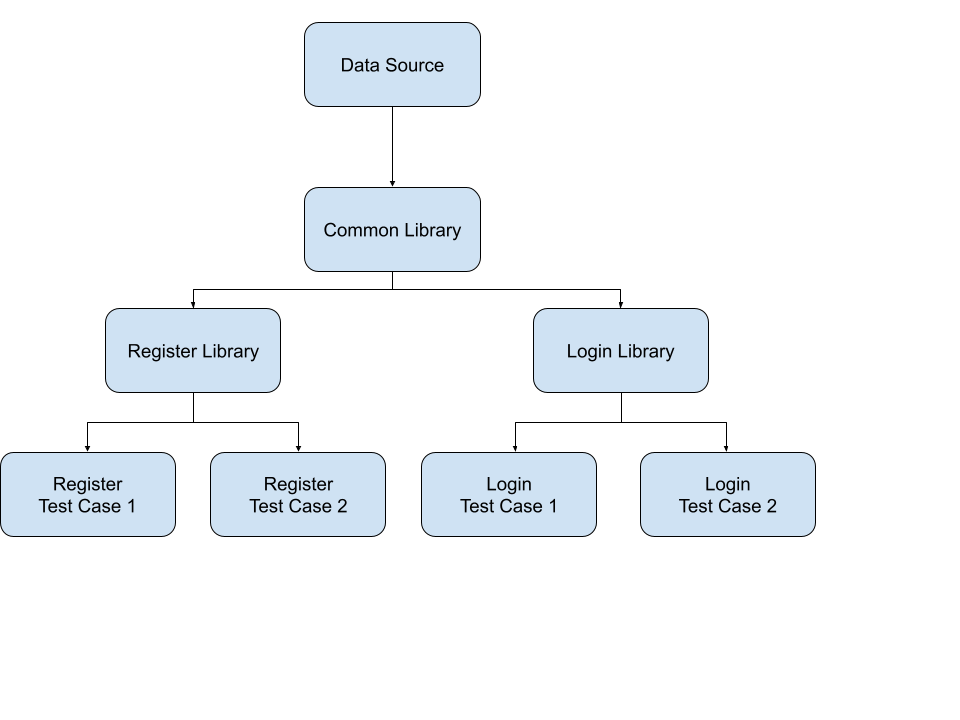
* Reduce Cost
* Save Time
* Foundation for CI and DevOps
* Accuracy and Reliability
* Increased Confidence
* Measure Quality Metrics

1. 3 SCOPE

The scope of this document is to show the following

* Architecture Diagram & Data Flow
* Architecture Explanation
* Test Cases Covered
* Used Technologies and Tools
* Improvements To Be Done
* Configuration And Run Steps

2.0 Architecture Design & Data Flow



In this architecture I have made reusable components to be used in all applications and called it a library. This approach will help me in code maintenance as any change in the application will modify it from my side to only one place and it will be reflected to the whole application automatically. I have made 3 libraries

* **Common Library**: It will be used by all the applications whatever the module being tested also it will feed the other libraries. In this library we will be handling also the data management layer
  + **Capabilities Class**: Setting the application configuration before running it
  + **ClossApp Class**: Close the application after running each test case to enhance the performance
  + **DataDriven Class**: Read the data from Excel using Apache POI and then frommating it to Object array to send it to TestNG
* **Module Library**: It will be used by specific module in the application, now we have 2 Module libraries
  + **Register Library**: It will be containing all the reusable code that will be used in Register module
    - **RegisterUtils Class**: Contains the steps of each page in the register flow so we can use them in all test cases and if there is any change in any element we will just change it in one place (enhanced solution to be done in the future to apply page object model and split this class to several classes each one serve only one page)
    - **LoginUtils Class**: Contains the steps of each page in the login flow so we can use them in all test cases and if there is any change in any element we will just change it in one place (enhanced solution to be done in the future to apply page object model and split this class to several classes each one serve only one page)
  + **Login Library**: It will be containing all the reusable code that will be used in Login module
* **Test cases Classes**: each class will be testing a test case separately here I am applying the single responsibility principle (SRP)



The data flow Diagram explains how the data flow from the data source till it reaches the test case by the following steps. The importance of this approach is that we can run the same test case with different data so we can try the positive scenario and negative scenario with the same test case which will save a lot of time doubling the test cases. **(It the same way used for Jira Automation Testing the only additional thing is password encryption)**

* Create the data source and fill it
* Read it in the common Library
* prepare the data and do some transformation
* Use it in the test cases through TestNG

3.0 Test Cases Covered

|  |  |  |  |
| --- | --- | --- | --- |
| **Module** | **Test Case ID** | **Test Case Title** | **Test Case Description** |
| Register | TC\_01 | Register with a Company with adding new image | Do the full registering cycle by taking new photos for the front and the back of the ID |
| Register | TC\_02 | Register with a Company with adding image from gallery | Do the full registering cycle by choosing photos for the front and the back of the ID from gallery |
| Register | TC\_03 | Register individually with valid bank account | Register individually with valid bank account |
| Register | TC\_04 | Register individually with valid Visa | Register individually with valid Visa |
| Register | TC\_05 | Register individually with valid mobile wallet | Register individually with valid mobile wallet |
| Register | TC\_06 | Register individually with invalid bank account | Register individually with invalid bank account and make sure it is not completing the registration |
| Register | TC\_07 | Register individually with invalid Visa | Register individually with invalid Visa and make sure it is not completing the registration |
| Register | TC\_08 | Register individually with invalid mobile wallet | Register individually with invalid mobile wallet and make sure it is not completing the registration |
| Register | TC\_09 | Enter invalid ID number | At the first step in the registration add invalid format for the ID number and make sure it is not completing the registration |
| Register | TC\_10 | Enter invalid mobile number | At the last step in the registration add invalid format for the mobile number and make sure it is not completing the registration |
| Register | TC\_11 | Search with non-existing company | At the first step in the registration try to search for a company not existing in the application |
| Login | TC\_12 | Login successfully with valid mobile number | Login with a correct mobile format and existing in the application |
| Login | TC\_013 | Login with invalid mobile number | Login with wrong mobile format and also login with correct format but not existing in the application |

4.0 Used Technologies And Tools

**Used Technologies, Libraries And Tools:**

**APPIUM Java**: APPIUM is a freely distributed open source mobile application UI Testing framework. Appium allows native, hybrid and web application testing and supports automation testing on physical devices as well as an emulator or simulator both. It offers cross-platform application testing, i.e. single API works for both Android and iOS platform test scripts.

**Apache Maven**: Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

**TestNG**: TestNG is an automation testing framework in which NG stands for "Next Generation". TestNG is inspired from JUnit which uses the annotations (@). TestNG overcomes the disadvantages of JUnit and is designed to make end-to-end testing easy. Using TestNG, you can generate a proper report, and you can easily come to know how many test cases are passed, failed, and skipped. You can execute the failed test cases separately.

**Log4J**: Apache Log4j is a Java-based logging utility.

**Base64**: Base64 encoding schemes are commonly used when there is a need to encode binary data that needs to be stored and transferred over media that are designed to deal with ASCII. This is to ensure that the data remains intact without modification during transport.

**Apache POI**: The Apache POI in Selenium is a widely used API for selenium data driven testing. It is a POI library written in Java that gives users an API for manipulating Microsoft documents like .xls and .xlsx. Users can easily create, modify and read/write into excel files. POI stands for “Poor Obfuscation Implementation.”

**Eclipse**: Eclipse is an integrated development environment (IDE) used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment.

**Android Studio**: Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools.

5.0 Improvements To Be Done

There are some points need to be improved not done for now due to time limitation which are the following

* Replace the the excel data source by database and introducing the Object–relational mapping (ORM) to ease the data access
* Integrate the the automation testing in CI/CD by running the test cases using Jenkins and Docker
* Enhance the way of form validation to ease the automation
* Replace the Log4J by Allure for better visualization
* Replace the Base64 encoding by EAS or RSA encryption for better security
* Introduce Page Object Model so we have an object for each page in the application

6.0 Configuration And Running Steps

You can find here in the [readme.md](https://github.com/mohamedsalah939/KhaznaTest/blob/master/README.md) all the configuration and running steps to run this application on other environments