2. Test data

Test data in automated tests

When writing UI automated tests, a big challenge appears in the definition of test data: What test data should I use in my tests? How can I structure my test data files? Where to keep the data? How can I be sure that my defined test data is stil available?

As a fast approach, test data is added in the current test, hard-coded in the test steps. This can be done fast but after a time passed each possible modification in the test data sets will take a longer time to modify it. For example, we have 30 tests that are using the same user and password. If the user of password is changed, then a maintenance is required for all tests in changing the test data by hand for all 30 tests. This approach is not working for a longer time because all time gained will be invested in the test maintenance.

The best solution is to store the test data in different files, external components and use the test data from those containers in the current tests. Like that, when a test data is changed, there is no need to update the tests, but just change the test data value in the test data container.

There are different ways of storing the test data. It can be done at:

- class level using a HashTable or defining global variables
- files level, add test data in different files (.txt, .xml, .csv, .xls) and implement a method for extracting the data from that file.
- data base level (define a default data base that will be used only for automated tests.)

No matter what decision is taken, a new test data Package should be added in our current UI automation structure.

Test data in Java Classes

We can define a new class in our TestData Package that will contain some general and global variables for storying different test data.

Another way is to create a HashTable that defines each of those variables in the Users class and implement a method that will also return the value from the table, like:

```
public class Users {
 public static String users(String key){

  Hashtable<String,String> users = new Hashtable<String,String>();
  users.put("admin","selenium");
  users.put("adminpass", "seleniumpassword");
  return users.get(key);
 }
}
```

When someone will call the method, it will look like:

```
Users.users("admin")  //will return the admin username
```

I don't want to recomand this method of storyng test data dirrectly in the code because it can add impediments in quickly setting up or changing the values of a given variable. When I want to change something, you need to take the source code, edit the values, build it again and then run it.

Test data in files

Test data can be stored in a set of files and configured in such a way that it will be helpful for the testers to extract the data from it.

This is a very good solution because it will help us in changing the test data (environment, users) directly in the source repository and then run the test (locally or from a CI environment) without having to change the code and rebuild it again.

Using this approach, a general "environment" file can be created that will contain the current environment for test. The environment URL is extracted from the file and used in each test class from the automation set without needing to update each test scenario but just making a 5 seconds change in the "environment" file.

In the same way test data for a given functionality can be created and defined, in different files based on the project functionality (e.g. Users.txt, NewUserData.txt, Cars.txt).

For example, the content of a Users.txt file can look like:

ADMINUSER=mirimiramoii
ADMINPASS=Aalndala99

NORMALUSER=abc

NORMALPASS=abcpass

There will be a general method of parsing the data from the file:

```
/*class for extracting the user values from Users.txt file*/
public class GetUsers{
 public static void UsersData(){
  String currentPath = new File("").getAbsolutePath();
  try {
   BufferedReader br = new BufferedReader(new FileReader(currentPath + "\\"+"Users"));
   String line;
   while (br.ready()){
   line = br.readLine();
   //parse the txt file based on the given keyword "="
   int idx = line.indexOf("=");
   String key = line.substring(0, idx);
   String value = line.substring(idx+1);
   //verify each main key value, take the value associated with it and set it in the
public value of class Data
   if (key.toLowerCase().equals("ADMINUSER".toLowerCase())){
    Data.ADMINUSER= value;
   }
   if (key.toLowerCase().equals("ADMINPASS".toLowerCase())){
    Data.ADMINPASS= value;
   }
   if (key.toLowerCase().equals("ADMINPASS".toLowerCase())){
    Data.NORMALUSER= value;
   }
   if (key.toLowerCase().equals("NORMALPASS".toLowerCase())){
    Data.NORMALPASS= value;
   }
  }
   br.close();}
  catch (Exception e) {
   e.printStackTrace();
  }
}
 public static class Data{
  public static String ADMINUSER;
  public static String ADMINPASS;
  public static String NORMALUSER;
  public static String NORMALPASS;
 }
}
```

When these data is required in the tests, it can be achieved as:

```
GetUsers.Data.ADMINUSER; //this will return the value associated for the admin user
which is: mirimiramoii
```

Test data from DB