Security report, Module 4, Team 41, Topicus Bank Transactions

Types of techniques of testing	Used?(Yes/No)
Unit tests	Yes
Integration tests	Yes
User test	Yes

JUNIT Testing

In the project Junit version 5.6.1 has been used in order to test the back-end functionality. 3 classes have been created ; AnalyticsTest, ConnectionHandlerTest and ParserTest respectively. In all the methods, annotation @Test has been used. Thanks to assert keywords it is possible to check whether the expected value matches the actual one.

Analytics Test:

In this Test class a new Analytics object has been created with a “testAccount” as a parameter.
Firstly, the method setFirst() is checked. Using assert keywords it is checked whether date has been added and if the analytics object has the accountID which was assigned it in the AnalyticsTest constructor.

Later the checkAddAmount() method in AnalyticsTest checks if AddAmount() method works correctly. Again the setFirst() method is being used and later addAmount() with a “D” sign which stands for Debit. With assert annotations it checks whether the date has been added and if the amounts have been subtracted.

ConnectionHandlerTest

In ConnectionHandlerTest, the ConnectionHandler class is checked. The test is relatively simple since it contains only 4 methods which all of them make use of the assertNotNull statement. However, additionally @TestMethodOrder annotation is used in order to perform test methods in desired order checkConnection(), checkPassword(), checkUrl() and checkDBName() respectively.

ParserTest

In the ParserTest class the Parser class is checked. In order to perform the test a sample MT940 file has been added to the source folder. Thanks to it, a new file object is created inside the test class and later this file object is passed as a parameter to create a new MT940 object. In order to carry out the comprehensive and reliable test all necessary objects have been created inside the ParserTest constructor. Thanks to it, it is possible to check all methods of the Parser class.

Method checklFirst3Fields() tests whether the MT940 file has all mandatory fields (according to the documentation) using assertNotNull annotation.

Later, method checkField60() makes use of a balance object and checks whether statement_date and first_balance have been parsed and passed to balance object correctly.

In method checkField61() “for each loop” which iterates for all the fields 61 which are in the MT940 file. In each iteration a new transaction is created by calling the method parseField61() from the parser class. The transaction is added to balance. At the end of the iteration assertNotNull annotation is used in order to check that the amount has been passed successfully to the transaction.

In method checkField62F() the balance is updated by calling the method parseField62F() from the parser class. Later assertNotNull annotation is called to check that the booked_balance has been passed to the balance successfully.

In method checkField64() the balance is updated by calling the method parseField64() from the parser class. Later assertNotNull is called in order to check that the final_balance has been passed successfully and assertEquals in order to check that the closing_date from the file matches the one which was parsed by the parser class.

User Tests

When it comes to user Testing we did not take any specific approach. Our team has asked a few people to take a look at the webapp and tell us what improvements can be made.
The interesting fact which our team found out during user testing was to add an analysis button to each row in the history table. Before that our team planned to have just one

analysis button at the top of the website which would redirect the user to the website with analysis for all the accounts.
The users found it more convenient than just having one analysis button on top of the website since it easily clarifies on which account an analysis is performed. Therefore, after user tests and discussion among team members, we decided to add an analysis button in every row of the account ID’s.

Integration Tests

Our team has used the “Big-Bang” approach. The main principle of this approach is to wait for most of the modules to be developed.

We waited with combining the frontend which was done by one member of our team using Bootstrap and the backend which was done in Java.
Since Bootstrap creates a lot of HTML and CSS code by itself we occured problems with identifying in which place of the website our data should be displayed. Our team also struggled with finding the name of the classes and name of ID’s. We solved most of the problems by adding IDs to the fields in HTML classes in order to make use of the document.getElementById() function.
After merging the front-end with back-end we were also struggling with running the webapp. We resolved the problem by reloading the maven project and running it's lifecycle (such as cleaning, validating, testing exc).