İZMİR UNIVERSITY OF ECONOMICS

SE 318 – SOFTWARE VERIFICATION AND VALIDATION
SPRING 2018

MOVIE TİCKET BOOKING SYSTEM

BAŞARCAN CELEBCİ

ÖZER ÇEVİKASLAN

SİMGE ÖZCAN

UNIT TEST DOCUMENT

Version *3.0*

11/05/2018

VERSION HISTORY

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	*Özer Çevikaslan*	*31/03/18*	*Simge Özcan*	*01/04/2018*	Initial
2.0	*Özer Çevikaslan*	*21/04/18*	*Başarcan Celebci*	*22/04/2018*	Test Case draft
3.0	*Özer Çevikaslan*	*10/05/18*	*Simge Özcan*	*11/05/2018*	Finalize

INTRODUCTION

1.1 PURPOSE OF THE TEST CASE DOCUMENT

Purpose of this test case document is to verify and validate that our software behaves as expected with the set of inputs given. The Test Case document documents the functional requirements of the *Movie Ticket Booking Systems* test cases. The intended audience is the project manager, project team, and testing team. Some portions of this document may on occasion be shared with the client/user and other stakeholder whose input/approval into the testing process is needed.

1.2 CONSTRAINTS

The project is developed with the Java programming language. Therefore, JUnit test framework is used for unit tests. Furthermore, Eclipse is used as development environment. Eclipse IDE were preferred since it has supports the creation of the JUnit test cases.

2 UNIT TEST FRAMEWORK: JUNIT

JUnit is a unit testing framework for Java programming language which is used for writing and running tests. It is the crucial part of the test-driven development. It increases the productivity of the programmer and the stability of program code, which in turn reduces the stress on the programmer and the time spent on debugging.

- Provides annotations to identify test methods.
- Provides assertions for testing expected results.
- Provides test runners for running tests.
- JUnit tests allow you to write codes faster, which increases quality.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback.
- JUnit tests can be organized into test suites containing test cases and even other test suites.
- JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

3 TEST CASES

Test Case 1	
Test Definition	
Login to the system	
Input Value	
Username and password	
Expected Value	**Actual Value**
Login success	**Login success**
Result of Test Case	***successful***
Test Script	

```
        // Login Test Case
        @Test
        public void login() {

        assertTrue(dbHandler.logIn("ozeriko".toString(),
"123456".toString()));
        }
```

Test Case 2	
Test Definition	
Sign up to the system	
Input Value	
Username and password	
Expected Value	**Actual Value**
Sign up to database success	**Sign up to database success**
Result of Test Case	***successful***
Test Script	

```
        // Sign Up Test Case
        @Test
        public void signUp() {

        assertTrue(dbHandler.signUp("ozeriko",
"123456"));
        }
```

Test Case 3	
Test Definition	
Add movie to the system	
Input Value	
Movie name	
Expected Value	Actual Value
Added to database successfully	Added to database successfully
Result of Test Case	*successful*
Test Script	

```
        // Add Movie Test Case
        @Test
        public void addMovie() {

        assertTrue(dbHandler.addMovie("john
wick 2", 15));
        }
```

Test Case 4	
Test Definition	
Check if the user exists	
Input Value	
Given username	
Expected Value	Actual Value
Exists on database	Exists on database
Result of Test Case	*successful*
Test Script	

```
        @Test
        public void checkUser() {
                assertNotEquals("Existed",
dbHandler.isUsernameExists("ozeriko"));
        }
```

Test Case 5	
Test Definition	
Check if the movie exists	
Input Value	
Movie name	
Expected Value	**Actual Value**
Exists on database	Exists on database
Result of Test Case	*successful*
Test Script	

```
	// Testing of checking if given movie
exists on database.
	@Test
	public void checkMovie() {
		assertNotEquals("Existed",
dbHandler.isMovieExists("undefined movie"));
	}
```

Test Case 6	
Test Definition	
Check if removing user works	
Input Value	
Username to be removed	
Expected Value	**Actual Value**
Removed from database	Removed from database
Result of Test Case	*successful*
Test Script	

```
	@Test
	public void checkRemovingUser() {

	assertTrue(dbHandler.removeUser("ozer"));
	}
```

Test Case 7	
Test Definition	
Check if the removing movie works	
Input Value	
Movie name to be deleted	
Expected Value	**Actual Value**
Movie removed from database	**Movie removed from database**
Result of Test Case	***successful***
Test Script	

```
       @Test
       public void checkRemovingMovie() {

       assertTrue(dbHandler.removeMovie("The
Lord of the Rings"));
       }
```

Test Case 8	
Test Definition	
Check if the main page is loaded properly	
Input Value	
Main page	
Expected Value	**Actual Value**
Loaded main page	**Loaded main page**
Result of Test Case	***successful***
Test Script	

```
              @Test
              public void startUpMainPage() {
                     assertNotNull(mainpage.addComponents
ToMainFrame(mainpage.frame.getContentPane()));
              }
```

Test Case 9	
Test Definition	
Check if the main page interface loads properly	
Input Value	
UI Components	
Expected Value	**Actual Value**
Loaded components	**Loaded components**
Result of Test Case	***successful***
Test Script	

```
		// Test case for if the UI components are loaded to the 
main page of the
		// application.
		@Test
		public void testMainPage() {
			
	mainpage.addComponentsToMainFrame(mainpage.frame.getContentPane());
			assertNotNull(mainpage);
		}
```

Test Case 10	
Test Definition	
Check if the login button loaded to login page	
Input Value	
Button component	
Expected Value	**Actual Value**
Loaded button	**Loaded button**
Result of Test Case	***successful***
Test Script	

```
		// Testing the login button 
UI component
		@Test
		public void loginButton() {
			
	lp.addComponentsToLoginPage(width, 
height);
			
	assertNotNull(lp.loginButton);
		}
```

Test Case 11	
Test Definition	
Check if the go back button is loaded properly	
Input Value	
Button component	
Expected Value	**Actual Value**
Loaded component	Loaded component
Result of Test Case	***successful***
Test Script	

```
		// Testing the backButton UI
component
		@Test
		public void
loginPagebackButton() {

	lp.addComponentsToLoginPage(width,
height);

	assertNotNull(lp.backButton);
		}
```

Test Case 12	
Test Definition	
Check if the page navigator works	
Input Value	
Navigator component	
Expected Value	**Actual Value**
Loaded component	Loaded component
Result of Test Case	***successful***
Test Script	

```
		// Testing the navigator
		@Test
		public void
loginPagenavigator() {

	lp.addComponentsToLoginPage(width,
height);

	assertNotNull(lp.navigator);
		}
```

Test Case 13	
Test Definition	
Check if the sign up button works properly	
Input Value	
Button component	
Expected Value	Actual Value
Loaded component	Loaded component
Result of Test Case	*successful*
Test Script	

```
                // Testing the sign up button
UI component
                @Test
                public void signUpButton() {

        sp.addComponentsToSignUpPage(width,
height);

        assertNotNull(sp.signUpButton);
                }
```

Test Case 14	
Test Definition	
Check if go back button in the sign up page works	
Input Value	
Button component	
Expected Value	Actual Value
Loaded component	Loaded component
Result of Test Case	*<successful OR fail>*
Test Script	

```
                // Testing the backButton UI
component
                @Test
                public void
signUpPagebackButton() {

        sp.addComponentsToSignUpPage(width,
height);

        assertNotNull(sp.backButton);
                }
```

Test Case 15	
Test Definition	
Testing the navigator	
Input Value	
Navigator	
Expected Value	**Actual Value**
Sign up page navigator	Sign up page navigator
Result of Test Case	*successful*
Test Script	

```
	// Testing the navigator
	@Test
	public void
signUpPagenavigator() {

	sp.addComponentsToSignUpPage(width,
height);

	assertNotNull(sp.navigator);
	}
```

Test Case 16	
Test Definition	
Check if adding movie button works	
Input Value	
Button component	
Expected Value	**Actual Value**
Loaded components	Loaded component
Result of Test Case	*successful*
Test Script	

```
	@Test
	public void addMovieButton() {

	addMoviePage.addComponentsToPanel();

	assertNotNull(addMoviePage.addMovieButton);
	}
```

Test Case 17	
Test Definition	
Check if movie name field works	
Input Value	
Text field component	
Expected Value	**Actual Value**
Loaded component	**Loaded component**
Result of Test Case	***successful***
Test Script	

```
          @Test
          public void addMovieTextInput() {

     addMoviePage.addComponentsToPanel();

     assertNotNull(addMoviePage.movieNameInput);
          }
```

Test Case 18	
Test Definition	
Check if ticket price input field works	
Input Value	
Text field component	
Expected Value	**Actual Value**
Loded component	**Loaded component**
Result of Test Case	***successful***
Test Script	

```
          @Test
          public void ticketPriceInput() {

     addMoviePage.addComponentsToPanel();

     assertNotNull(addMoviePage.movieTicketPrice);
          }
```

Test Case 19	
Test Definition	
Test the singleton pattern	
Input Value	
Navigator	
Expected Value	**Actual Value**
One instance of navigator	**One instance of navigator**
Result of Test Case	***successful***
Test Script	

```
		// Testing the getting the singleton
navigation stack object.
		@Test
		public void getNavigator() {
			assertNotNull(navigator =
NavigationStack.getInstance());
		}
```

Test Case 20	
Test Definition	
Test adding new page to the system	
Input Value	
New page	
Expected Value	**Actual Value**
Added page	**Added page**
Result of Test Case	***successful***
Test Script	

```
		// Testing of adding a panel as a new page to
the navigator stack.
		@Test
		public void addPageToNavigator() {
			JPanel testPanel = new JPanel();

	assertNotNull(navigator.addPageToNavigator(testPanel,
"Test Panel"));
		}
```

4 CONCLUSION

In conclusion, a movie ticket booking system were implemented with Java. Our verification and validation process has grown together with the implementation process. As project any component were implemented, their tests also implemented. So that we tested the functionalities as early as possible to prevent serious defects and faults.