TALLINN UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF SOFTWARE SCIENCE

Filmography website for the Ministry of Culture of Estonia

Testing and verification activities (excerpt)

Lab 3 in subject "Software Quality and standards" (IDY0204)

Authors, student group

Maksym Avramenko (223884IVSM) Developer

Viktoriia Abakumova (223890IVSM) Tester

Fillip Molodtsov (223891IVSM) Maintainer

Azhar Kazakbaeva (223909IVSM) Acquirer (contact person)

Team ID: 2+2

Presented: 19.11.2022

Supervisor: Jaak Tepandi

TALLINN 2022

A7. Functional specification-based testing

7.1 Test cases were taken from the previous laboratory.

Based on previous laboratory and system components, Registration module was selected for testing. In our case, these is a controller that is responsible for the user operations. All tests cases are from Laboratory 1. Below are just two of the main ones.

Use Case ID	UC_001
Use Case Name	Successful registration of a system user
Primary Actor	System user
Preconditions	● User has access to the main page of the site ● User has access to the ‘sign up’ button ● User has a valid email ● User pressed the ‘sign up’ button
Postconditions	User account was created and saved in the system and user was redirected to main page
Main Success Scenario	1. User redirected to the signup page 2. User fills the form with a valid name, email and password 3. User submits the data to the form by clicking ‘sign up’ button

Use Case ID	UC_003
Use Case Name	Successful log in of the user
Primary Actor	System user
Preconditions	● User has access to the main page of the site ● User has access to the ‘log in’ button ● User has a valid username and password ● User pressed the ‘login’ button
Postconditions	The user is logged-in and redirected to the main page
Main Success Scenario	1. User is redirected to the login page 2. User fills the form with username and password 3. User submits the data to the form by clicking ‘log in’ button

	4. The user is refirected to the main page

7.2. Designing functional tests.

Checklist of functional tests for User Registration:

Equivalence classes	NAME FIELD	EMAIL FIELD	PASSWORD FIELD
Positive case VALID	**Class_1_Length**: 3 to 255 symbols; Name field accepts Uppercase and lowercase English letters (a–z, A–Z) and Digits 0 to 9	**Class_1_Email_length**: 1-254 symbols; Email accepts Uppercase and lowercase English letters (a–z, A–Z) and Digits 0 to 9	**Class_1_Length**: 8 to 255 symbols
		Class_1_Domain_name is present and valid	
Negative case INVALID	**Class_2_Length**: 0 to 2, > = 256	**Class_2_Email_length**: 0, > 255	**Class_2_Length**: 0 to 7, > = 256
		Class_2_Domain_name is missed	

We used the equivalence classes technique for this table.

'Test Data' table

NAME FIELD	EMAIL FIELD	PASSWORD FIELD
0 symbol: "" **2 symbols:** ty **3 symbols:** qwe **25 symbols:** Atmj47sjpd4RSOsCT3Tx6uVrT **255 symbols:** nlCCdxDbB7UW2qbLVt4dCFiMb2MXlgpuFkCdgHseywA8Tty3otUsSITXzoJbxxWf2kATVSiGYitsiEslMsBuKcU8D7n5WVeil7gcHyERXZJApQbFno77GjGTUWfvSBR4gduzsMj7U9EThqRixawnp8Aj6Mj2Uy0zhdO7zhRK8gpeorRLKqttPlfxAKNVPqh7d2T3jbmHDgrVn75u950A2ug9IMMRvyqzijYB7rYKuBqUAn5RHDhQQhHhsV2QuKo **256 symbols:** KiekqemWRIadt55DvlkOJ6c9w1NnkRoa9Xkucumq48YNBuKJ331LSP8h7eGD5NsWLaqMLR8k4rpHW81BtY5bQJy0Rf5awJ1tow0ZZBP3iuVoMJCnesg1uC0XZCyYjK4FXj9NequXVysnlYg4GPYtxo4EdC0AjpoKppVvPzW9CSSbOQimiOGmB8ShYffv4lzQL78zLOd9zxvglJwdAkl6zTKRQvN2me	**0 symbol: ""** **1 symbol:** a@test.com **4 symbols:** qwer@test.com **w/o domain:** test	**0 symbol: ""** **2 symbols:** ty **7 symbols**: qwertyu **8 symbols:** qwertyui **25 symbols:** Atmj47sjpd4RSOsCT3Tx6uVrT
	264 = 255 + 9 IUDuPRGTDpNX9U7vWpgMIqc4dmqCIZfPd7U7VLJjpnlOmEuiGAfVky0OIIcgViUHJQi8YEdprlUte55jbybotJOEJZCeNZDSVVLZnn8zhrGMSJUiXfBxQc1nxwuUt3dfm6v3JHtAFM6ZzmmPvGDWaDgDkz7xQXJJIXNW1PuMsVchq0B9eK4gRPepiHYOY0C22U55yfn1o1nqwxHPTwNhOP03p2jOOG2LGqYWYsfg8rejmXzSb8Z7xnXTBmILzPi**@test.com**	**255 symbols** UtWPk8PZg7fthMAkrV7lo0u7CFLMTz9NSmXeUPh4usQ1lIInosSjOuocRp8DDQWsIpcV2zsiPfPDad580jsd2k8fn7NP4D6MSOuSW5vm4TCMsNfJMFV3YGhjyIIAwVtrLEHmlPg7J79fmcHDl1Wl5jvAiUnnjW8pN5wnlzTuTnX7h1DU1KGZi6ptU4EAQFkXogrJGD5GBc82KtcYAaKVVPO6R3INfdMzGCNAdVwHWARMesm653GK8jOAQ4FNDRo
	309 = 300 + 9 oIvRCobi2TFezwelLXCpD7E0XSPDV28vGC3X15brvjbfQGfzqbJQO9JRx3mgMdmyS95TzLDSBHD2g351PdgqI9Ilax3d1YyiJyH1tvWTVgrNuO0WASCj8xl3oYFjUKFPvfSqPZynEFWVhQoVrMbYokwcLoXu1ANqiLgpB44Xf6vuX8rOesEM3TuRUTHd9O8JADZqsqfqryBhhVMPTM68Paat3b39vl46ZwCaZJKwKsyC7Z7f9CWIK5Qh3na5xZ657lWAPLSodpttxUVyFEbQzXbumSbICq7JwOubkkuiHS3Y**@test.com**	**256 symbols** jy6eDYfEM0CpnQlFu7stBYEXpvJKu1CeQYVCNaEDZosnxi0bJxFX29j7AEf2gf7A1klVasH0G4mSoJ6La8ylAdhnop2gIa7ELbHqU6nlX7e5qzfXVoRzzpra0OO9Y5iKH9wg33wTTY8V5gOSUewAh0y6enmvIIGv7N206XTcktN7Dkt5byag2q39RV3Ul57oLW0aYL1gkrkC9JHngEDDH0RsTtSVkiamxT6RPDSZUvbCkCph5jnZvUOW86TJdgK5

X5e2rltfk2LXwr98zQgqfLZrC0kxjOHyze **300 symbols:** gqmzuq2fa7nbinfOWjAgUsffoddjy1PFeX8SLxy2Vl7jFVsTbXya72bsrdeB4pyApEZ9JXvW2XpcGRb0zhTBXZMH2N7erZ17dh7jZrqc4iYVxezONdpKVp6I9nnEID5Dm9CPCmvmvPPf57ZHKjSQu7ZE2WRwbWMAY9QdFQDUmtfqPAVyjnlpceUmsYl8e8lsPn3k2ZsTS3QQKqXk4jl2T6W3DHjwf7HILqTUmrW7s20RE1nuYy9engiooKC1Kql7eSDhCEZbSlVPKD12P3S9gntvRbSNBDFXE2bNJbjOwuZA		**300 symbols** fgFXUk5aDxdb6e4NzxwFnbYzbxsz2D0v2FV9hH0fg7zfBe3zNqO6mQNJ7Zd5YQFxHe8u0i7qGoRcfygMMCAXCWjbxjQZR9KTwm2TYabpvPlVtxwT6Ad9ATUQJiawBVgZvMZZJAexeRbKuf6tTEyOnTlwGZ7FcLC0I6Q5R4pbzVg70pwQVk6J1y0XSpDBb35FVZuSOaypz0ajKENRO4TvM47EDBpTB1fynKwn3KQ49IXec4S0wOeHItvDMoAe6JLjNsMsfZOuxSzrdJsj1FrwhOsi0r2z7HptVtqalUjAHlmS

We wrote tests cases to cover most of common input validations for registration form.
All test data is used from the table ‘Test Data’.
To prepare ‘Test Cases’ table we used also boundary value technique.
Used designations:
8s – 8 symbols; number(s) – number of symbols
v/e – valid email; 1 + d – 1 symbol + valid domain

‘Test cases’ table

Id	Description	Test case			Expected result
		Name	Email	Password	
TC1	User fills in Name, Email and Password next information	Empty	Empty	Empty	Negative
TC2	User fills in Name, Email and Password next information	Empty	v/e	8s	Negative
TC3	User fills in Name, Email and Password next information	2s	v/e	8s	Negative
TC4	User fills in Name, Email and Password next information	3s	v/e	8s	Positive
TC5	User fills in Name, Email and Password next information	25s	v/e	8s	Positive
TC6	User fills in Name, Email and Password next information	255s	v/e	8s	Positive
TC7	User fills in Name, Email and Password next information	256s	v/e	8s	Negative
TC8	User fills in Name, Email and Password next information	300s	v/e	8s	Negative
TC9	User fills in Name, Email and Password next information	3s	Empty	8s	Negative
TC10	User fills in Name, Email and Password next information	3s	1 + d	8s	Positive
TC11	User fills in Name, Email and Password next information	3s	4 + d	8s	Positive
TC12	User fills in Name, Email and Password next information	3s	255 + d	8s	Negative
TC13	User fills in Name, Email and Password next information	3s	300 + d	8s	Negative
TC14	User fills in Name, Email and Password next information	3s	4 wd	8s	Negative
TC15	User fills in Name, Email and Password next information	3s	v/e	Empty	Negative
TC16	User fills in Name, Email and Password next information	3s	v/e	2s	Negative
TC17	User fills in Name, Email and Password next information	3s	v/e	7s	Negative
TC18	User fills in Name, Email and Password next information	3s	v/e	25s	Positive
TC19	User fills in Name, Email and Password next information	3s	v/e	255s	Positive
TC20	User fills in Name, Email and Password next information	3s	v/e	256s	Negative
TC21	User fills in Name, Email and Password next information	3s	v/e	300s	Negative
TC22	User fills in Name, Email and Password next information	3s	v/e	8s	Positive

7.3. Saving and running the tests using a test automation tool.

As a stack we used: Java, Rest Assured, JUnit.

While working on writing autotests we saw that for Positive case we have the same parameters (Name, Email, Password) as input and the response is also the same – one difference is in these parameters, so we decided to improve the approach and use DDT. Rest Assured frameworks uses BDD approach (so we have given-when-then block).

We prepared excel file (TestData.xlsx) for this: (Sheet 1 is used for Positive test cases)

	A	B	C	D
1	Test Case Name	Name	Email	Password
2	TC4	qwe	test1@test.com	qwertyui
3	TC5	\tmj47sjpd4RSOsCT3Tx6uVr	test2@test.com	qwertyui
4	TC6	jGTUWfvSBR4gduzsMj7U9ET	test3@test.com	qwertyui
5	TC10	qwe	a@test.com	qwertyui
6	TC11	qwe	qwer@test.com	qwertyui
7	TC18	qwe	test4@test.com	ıj47sjpd4RSOsCT3Tx6uVr
8	TC19	qwe	test5@test.com	jyIIAwVtrLEHmlPg7J79fm
9	TC22	qwe	test6@test.com	qwertyui

Sheet 2 with invalid data:

	B	C	D	E
1	Test Case Name	Name	Email	Password
2	TC1			
3	TC2		test1@tes	qwertyui
4	TC3	ty	test2@tes	qwertyui
5	TC7	KiekqemWRIadt55DvlkOJ6	test3@tes	qwertyui
6	TC8	gqmzuq2fa7nbinfOWjAgU:	test4@tes	qwertyui
7	TC9	qwe		qwertyui
8	TC12	qwe	IUDuPRGT	qwertyui
9	TC13	qwe	oIvRCobi2	qwertyui
10	TC14	qwe	test	qwertyui
11	TC15	qwe	test5@test.com	
12	TC16	qwe	test6@tes	ty
13	TC17	qwe	test7@tes	qwertyu
14	TC20	qwe	test8@tes	jy6eDYfEM0
15	TC21	qwe	test9@tes	fgFXUk5aDx

We also prepared additional class for working with Excel file (see Appendice A code for A7.3).

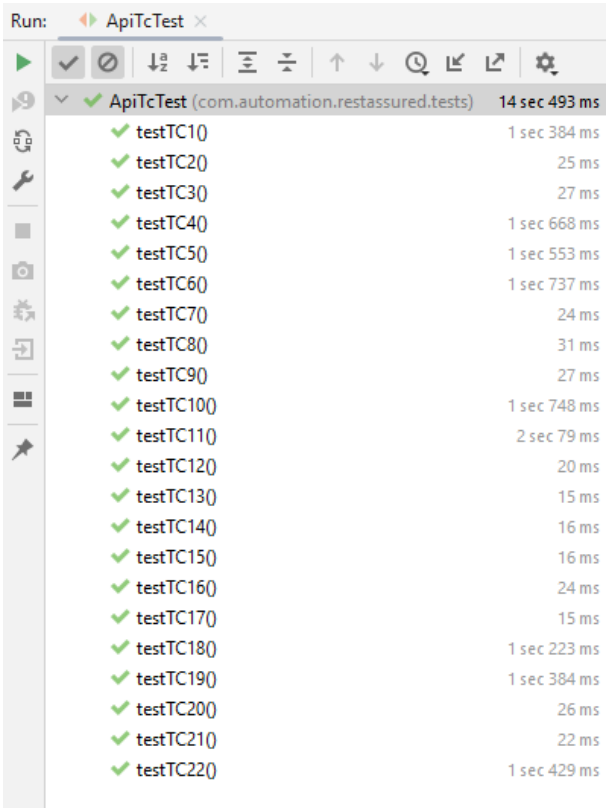
Then we prepared test for verifying registration flow with valid data.

```
@BeforeMethod
public static void before() { baseURI = "http://localhost:5000/"; }

1 usage
@DataProvider
public Object[][] dataValid() {
    return (ExcelUtils.getTableArray( filePath: "src/test/resources/TestData.xlsx",  sheetName: "Sheet1"));
}

@Test(dataProvider = "dataValid")
public void testValidRegistrationData(String name, String email, String password) {
    final String json = String.format("{\"name\": %s, \"email\": %s, \"password\": %s}", name, email, password);

    given() RequestSpecification
            .contentType( s: "application/json")
            .body(json)
            .when()
            .post( s: "api/v1/user/register") Response
            .then() ValidatableResponse
            .statusCode( i: 200)
            .contentType( s: "application/json")
            .body( s: "auth_token", notNullValue())
            .body( s: "info.name", equalTo(name))
            .body( s: "info.email", equalTo(email));
}
```

As the situation with registration flow with invalid data is different there we wrote separate tests because each field has its own validation message and in different combinations we expect one or more messages and it's not the same for different test data. All of them have test case name, so in result of testing we can see which test has failed or passed. Examples of a few more negative tests could be found in Appendice A Negative autotests.

```
@Test
public void testTC1() {
    final String json = ("{\"name\": \"\", \"email\": \"\", \"password\": \"\"}");

    given() RequestSpecification
            .contentType( s: "application/json")
            .body(json)
            .when()
            .post( s: "api/v1/user/register") Response
            .then() ValidatableResponse
            .statusCode( i: 400)
            .contentType( s: "application/json")
            .body( s: "error", hasItem("`name` field is mandatory"))
            .body( s: "error", hasItem("`email` field is mandatory"))
            .body( s: "error", hasItem("`password` field is mandatory"));
}
```

Test run:



As a result of this part: we assume that all tests passed so we covered registration flow fully with autotests using different techniques for test-design and test automation.

A8. Risk assessment and designing acceptance tests.

8.1. Risk assessment.

In this section, the main risks outlined in lab 1 are discussed and ranked based on their potential impact and frequency, using the 5X5 risk matrix. Each risk is assigned an identifier that allows it to be referred to. The likelihood of incident, risk assessment matrix and main risks are described as follows:

Likelihood of Incident:

Very Low	Very unlikely to happen
Low	Once every 182 days
Medium	Once every 30 days
High	Once 7 days
Very High	Once an hour

Risk assessment matrix

		Likelihood of Incidence				
		Very Low	Low	Medium	High	Very High
Impact	Very Low	1	2	3	4	5
	Low	2	3	4	5	6
	Medium	3	4	5	6	7
	High	4	5	6	7	8
	Very High	5	6	7	8	9

Assessment of system usage risks:

Risk ID	Description	Likelihood	Impact	Risk rating	Priority
R-001	Incompatibility of the in-house servers with the development servers	Low	High	4	4
R-002	Restriction of the databases access, of the system access	Low	High	4	4
R-003	Overloaded by users system and time-outed responses	Medium	High	6	2
R-004	Cyber-security of vulnerability issues	Medium	Very High	7	1
R-005	Insufficient technical infrastructure	Low	Medium	4	4
R-006	Unexpected shrink(change) of the development team	Medium	Medium	5	3
R-007	Delayed budget payments	Low	Medium	4	4
R-008	Non-compliance to the GDPR	Very Low	Very High	5	3
R-009	Breakage on new browser updates from the supported set of browsers	Medium	Very High	7	1
R-010	Delay for at least one sprint period from the scheduled plan	Medium	High	6	2

8.2. Risk-based acceptance tests.

From the previously defined risks and requirements, 20 risk-based acceptance tests are outlined below in the order of risk priority. 12 of these tests focus on functional requirements. The other eight requirements focus on non-functional requirements (with two of them being load tests)

Test ID	Risk ID	Req ID	Input Description	Expected Description
T1	R-004	UC_004	A user tries to log in with invalid credentials	The system denies user access
T2	R-003	UC_001	A user fills in valid information for creating a profile	The system successfully adds a user
T3	R-003	UC_003	A user tries to log in with valid data	The system validates data and redirects to the main page
T4	R-003	UC_006	A user logs in, chooses film and adds the comment	The system saves the comment and displays it in the comment section
T5	R-003	UC_007	A user logs in, chooses film and modifies previous comment	The system updates previous comment and displays it in the comment section with "(Updated)" remark
T6	R-003	UC_008	A user logs in, chooses film and deletes previous comment	The system deletes the comment and it does not visible in the comment section
T7	R-003	UC_009	A user searches a film using some text (valid search)	The system shows the result that contains the search text
T8	R-003	UC_009	A user searches a film using some text (invalid search)	The system shows the message that nothing was found
T9	R-003	UC_010	User can rate the film	The system saves the rate and shows it in the rating section
T10	R-008	UC_005	User can delete the profile	The system deletes whole information about the user
T11	R-003	UC_002	User tries to register with empty name	The system validates data and returns the error
T12	R-003	UC_002	User tries to register with existing email	The system validates data and returns the error

T13	R-005	UC_011, UC_020	Due to unexpected behavior some schemas in the database is deleted	The system has backup of users and films and is able to restore data
T14	R-003	UC_017	The user enters new information into the system	The system responds with message within 2500ms
T15	R-005	UC_190	User can translate the page with Google Translate extension	The system is compatible with Google Translate extension
T16	R-009	UC_016	The project is opened in the latest versions of Chrome, Opera, Firefox and Safari	The system is available on all aforementioned browsers
T17	R-009	UC_016	The project is opened in the latest versions of Chrome, Opera, Firefox and Safari	The project has the same view in all aforementioned browsers
T18	R-009	UC_016	The user attempts to fill any form without the required fields	The system notifies the user of missing fields
T19	R-003	UC_014 (load)	1000 users create profile at the same time	The system does not become slow or crash
T20	R-003	UC_014 (load)	1000 users search films	The system does not become slow or crash

8.3. Acceptance criteria

The fundamental acceptance criterion is that the system is used for its intended purpose and that it meets all requirements. That being said, acceptance of the software is based on the acceptance tests stated above. All tests that assess whether critically significant requirements are met must be passed. For the software to be accepted, 98 percent of all other tests must pass. As described in Laboratory 01, the table below summarizes acceptance of key aspects of the system development, testing and maintenance phase:

Action	Deliverable	Time
Requirements definition: Functional and Non-Functional requirements are defined.	The documentation provided by the procurer	Performed before procurement. The detailed requirements shall be specified in the following stages

Development (optional) Developers have to work according to Agile framework and deliver new features every sprint	Source code and documentation meets procurer's requirements	(optional) Two months
Testing the system performance: 1. Installation and deployment 2. Testing the system on at least 2 environments (QA and Stage) performed by the development team	Report and demonstration provided by the development team to the procurer	One month
Maintenance The development team will be responsible for maintenance. The development team must ensure that the system operates properly during maintenance. Additionally, they must delete obsolete files from storage and guarantee sufficient backup security.	Maintenance report provided by the development team to the procurer.	Post-contractual requirements to perform quarterly maintenance. The project's budget covers maintenance for a year.

A9. Acceptance testing, its documentation in the Lab 3, and preliminary system evaluation

9.1. Functional acceptance testing.

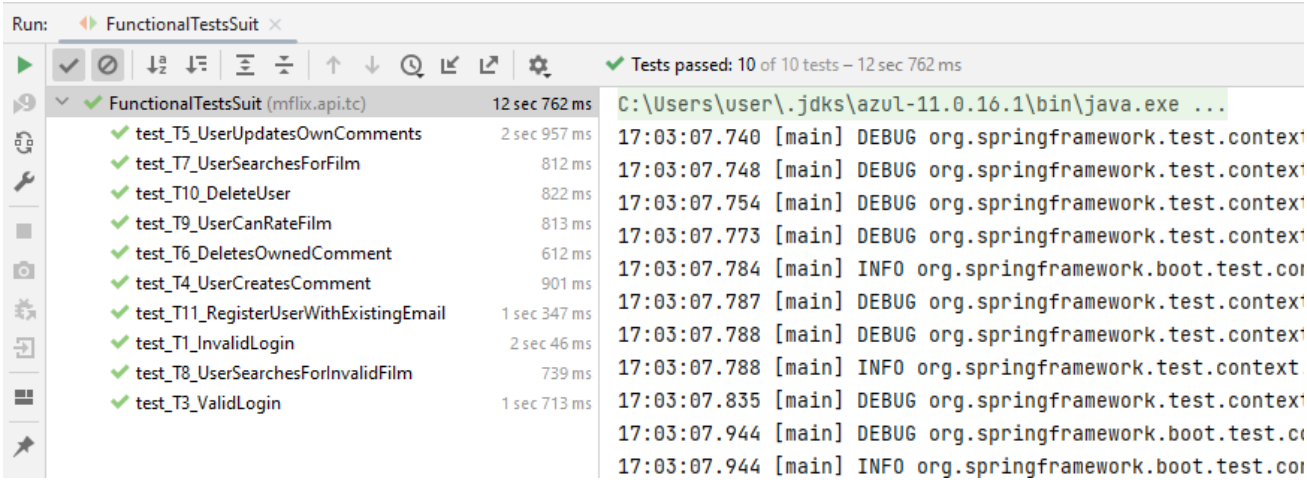
The results of 12 functional acceptance tests specified above have been covered with autotests and were executed using appropriate tools. All autotests for this part are in Appendice B.

Test ID	Test Description	Auto-test name	Acceptance Criteria	Status
T1	A user tries to log in with invalid credentials	test_T1_InvalidLogin()	Test Passes	Pass

T2	A user fills in valid information for creating a profile	testValidRegistrationData() re-use tests from A7 7.3 (they are located in different class test but they are part of test suit)	Tests Pass	Pass
T3	A user tries to log in with valid data	test_T3_ValidLogin()	Test Passes	Pass
T4	A user logs in, chooses film and adds the comment	test_T4_UserCreatesComment()	Test Passes	Pass
T5	A user logs in, chooses film and modifies previous comment	test_T5_UserUpdatesOwnComments	Test Passes	Pass
T6	A user logs in, chooses film and deletes previous comment	test_T6_DeletesOwnedComment()	Test Passes	Pass
T7	A user searches a film using some text (valid search)	test_T7_UserSearchesForFilm()	Test Passes	Pass
T8	A user searches a film using some text (invalid search)	test_T8_UserSearchesForInvalidFilm()	Test Passes	Pass
T9	User can rate the film	test_T9_UserCanRateFilm()	Test Passes	Pass
T10	User can delete the profile	test_T10_DeleteUser()	Test Passes	Pass

T11	User tries to register with empty name	testTC2() re-use tests from A7 7.3 (they are located in different class test but they are part of test suit)	Test Passes	Pass
T12	User tries to register with existing email	test_T11_RegisterUserWithExistingEmail	Test Passes	Pass

Results:



9.2. Non-functional acceptance testing

Non-functional acceptance testing were done manually.

Test ID	Test Description	Steps	Acceptance Criteria	Status
T13	Verify possibility to restore data from back-up	1. Delete schema 2. Restore it from back-up	Data is restored	Pass
T14	Verify response time of requests	1. Turn on Charles Proxy 2. Make requests	Time does not exceed 2500 ms	Pass
T15	Verify translation of pages	1. Go to the main page 2. Use Google Translate extension	Page is translated	Pass
T16	Verify compatibility	1. Open main page in Chrome 2. Open Opera 3. Open Firefox	User can view main page in	Pass

		4. Open Safari	different browsers	
T17	Verify that pages are the same in different browsers	1. Open main page in Chrome 2. Open Opera 3. Open Firefox 4. Open Safari	User can view main page in different browsers	Pass
T18	Verify that user can see validation messages	1. Open main page 2. Click on Sign Up button 3. Click on Submit button	The system notifies the user of missing fields	Pass

For performance testing we use Gatling (Scala language). Scripts for testing could be found in Appendice C.

Test ID	Test Description	Acceptance Criteria	Status
T19	1000 users create profile at the same time	Test Passes	Pass
T20	1000 users search films	Test Passes	Pass

Result:

```
================================================================================
2022-11-12 01:36:57                                70s elapsed
---- Requests ------------------------------------------------------------------
> Global                                          (OK=4197   KO=0)
> Search Page                                     (OK=4197   KO=0)
> User Sign Up Action                             (OK=4197   KO=0 )


---- Global Information --------------------------------------------------------
> request count                                   4197 (OK=4197   KO=0    )
> min response time                               561 (OK=561    KO=-    )
> max response time                               21101 (OK=21101  KO=-    )
> mean response time                              17188 (OK=17188  KO=-    )
> std deviation                                   4720 (OK=4720   KO=-    )
> response time 50th percentile                   19853 (OK=19853  KO=-    )
> response time 75th percentile                   19999 (OK=19999  KO=-    )
```

```
> response time 95th percentile                 20077 (OK=20077  KO=-     )
> response time 99th percentile                 20216 (OK=20216  KO=-     )
> mean requests/sec                           48.241 (OK=48.241 KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                    1 (  0%)
> 800 ms < t < 1200 ms                          0 (  0%)
> t > 1200 ms                                4196 (100%)
> failed                                     0 (  0%)
================================================================================
```

9.3. Summarising the results of acceptance testing.

As seen in the summary for functional requirements testing, 100% of the tests pass. The tests cover around 95% of the criteria. Results of performance tests are rather acceptable. According to received results we could say that the system is prepared and could be deployed for beta testing. For further steps we need to discuss with the customer new risks and add additional acceptance tests.

9.4. System evaluation, risk analysis, acceptance.

Given all of the previous results and the acceptance criteria, it is concluded that the system is ready for use at beta form. So, the system has main functionality done, all necessary documentation, tests (manual, auto-tests, performance tests). Main risks are covered with tests.

Appendice A

Code for A7.3

Class for work with Excel file for DDT:

```
package com.automation.ddt.utils;

import org.apache.poi.ss.usermodel.CellType;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.FileInputStream;
import java.io.IOException;

import static org.apache.poi.ss.usermodel.CellType.*;

public class ExcelUtils {

    private static XSSFSheet excelWSheet;
    private static XSSFWorkbook excelWBook;
    private static XSSFCell cell;

    public static Object[][] getTableArray(String filePath, String sheetName) {
        String[][] tabArray = null;
        try {
            FileInputStream excelFile = new FileInputStream(filePath);
            excelWBook = new XSSFWorkbook(excelFile);
            excelWSheet = excelWBook.getSheet(sheetName);
            int startRow = 1;
            int startCol = 1;
            int ci, cj;
            int totalRows = excelWSheet.getLastRowNum() - 1;
            System.out.println(totalRows);
            int totalCols = 2;
            tabArray = new String[totalRows][totalCols];
            ci = 0;
            for (int i = startRow; i <= totalRows; i++, ci++) {
                cj = 0;
                for (int j = startCol; j <= totalCols; j++, cj++) {
                    tabArray[ci][cj] = getCellData(i, j);
                    System.out.println(tabArray[ci][cj]);
                }
            }
        } catch (IOException e) {
            System.out.println("Could not read the Excel sheet");
            e.printStackTrace();
        }
        return (tabArray);
    }

    private static String getCellData(int rowNum, int colNum) {
        try {
            cell = excelWSheet.getRow(rowNum).getCell(colNum);
            CellType dataType = cell.getCellType();
            if (dataType == _NONE) {
                return "";
            } else {
                return cell.getStringCellValue();
```

```
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
            throw (e);
        }
    }
}
```

Negative autotests:

```
@Test
public void testTC2() {
    final String json = ("{\"name\": \"\", \"email\": \"test1@test.com\",
\"password\": \"qwertyui \"}");

    given()
            .contentType("application/json")
            .body(json)
            .when()
            .post("api/v1/user/register")
            .then()
            .statusCode(400)
            .contentType("application/json")
            .body("error", hasItem("`name` must be at least 3 characters long"));
}


@Test
public void testTC14() {
    final String json = ("{\"name\": \"qwe\", \"email\": \"test\", \"password\":
\"qwertyui\"}");

    given()
            .contentType("application/json")
            .body(json)
            .when()
            .post("api/v1/user/register")
            .then()
            .statusCode(400)
            .contentType("application/json")
            .body("error", hasItem("`email` must be an well-formed email
address"));
}


@Test
public void testTC15() {
    final String json = ("{\"name\": \"qwe\", \"email\": \"test5@test.com\",
\"password\": \"\"}");

    given()
            .contentType("application/json")
            .body(json)
            .when()
            .post("api/v1/user/register")
            .then()
            .statusCode(400)
            .contentType("application/json")
            .body("error", hasItem("`password` field is mandatory"));
}
```

```
@Test
public void testTC17() {
    final String json = ("{\"name\": \"qwe\", \"email\": \"test7@test.com\",
\"password\": \"qwertyu\"}");

    given()
            .contentType("application/json")
            .body(json)
            .when()
            .post("api/v1/user/register")
            .then()
            .statusCode(400)
            .contentType("application/json")
            .body("error", hasItem("`password` must be at least 8 characters
long"));
}


@Test
public void testTC20() {
    final String json = ("{\"name\": \"qwe\", \"email\": \"test8@test.com\",
\"password\":
\"jy6eDYfEM0CpnQlFu7stBYEXpvJKu1CeQYVCNaEDZosnxi0bJxFX29j7AEf2gf7A1klVasH0G4mSoJ6
La8ylAdhnop2gIa7ELbHqU6nlX7e5qzfXVoRzzpra0OO9Y5iKH9wg33wTTY8V5gOSUewAh0y6enmvIIGv
7N206XTcktN7Dkt5byag2q39RV3Ul57oLW0aYL1gkrkC9JHngEDDH0RsTtSVkiamxT6RPDSZUvbCkCph5
jnZvUOW86TJdgK5\"}");

    given()
            .contentType("application/json")
            .body(json)
            .when()
            .post("api/v1/user/register")
            .then()
            .statusCode(400)
            .contentType("application/json")
            .body("error", hasItem("`password` must be at max 255 characters
long"));
}
```

Appendice B

Code for A9.1

```
package mflix.api.tc;

import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import mflix.api.daos.CommentDao;
import mflix.api.daos.MovieDao;
import mflix.api.daos.RatingDao;
import mflix.api.daos.UserDao;
import mflix.api.models.Comment;
import mflix.api.models.Session;
import mflix.api.models.User;
import mflix.config.MongoDBConfiguration;
import org.apache.commons.lang3.RandomStringUtils;
import org.bson.Document;
import org.bson.types.ObjectId;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Objects;

import static org.junit.Assert.*;
import static org.junit.Assert.assertNull;

@SpringBootTest(classes = {MongoDBConfiguration.class})
@EnableConfigurationProperties
@EnableAutoConfiguration
@RunWith(SpringJUnit4ClassRunner.class)
public class FunctionalTestsSuit {

    private UserDao userDao;
    private MovieDao movieDao;
    private CommentDao commentDao;
    private RatingDao ratingDao;

    private static String email = "gryffindor@hogwarts.edu";
    private User testUser;
    private String jwt;

    private String notValidEmail;

    private String validEmail;
```

```
    private String fakeCommentId;

    @Autowired
    MongoClient mongoClient;

    @Value("${spring.mongodb.database}")
    String databaseName;

    @Before
    public void setup() {
        this.userDao = new UserDao(mongoClient, databaseName);
        this.testUser = new User();
        this.testUser.setName("Hermione Granger");
        this.testUser.setEmail(email);
        this.testUser.setHashedpw("somehashedpw");
        this.jwt = "somemagicjwt";
        mongoClient
                .getDatabase("mflix")
                .getCollection("users")
                .deleteOne(new Document("email", "log@out.com"));
    }

    @After
    public void tearDownClass() {
        MongoDatabase db = mongoClient.getDatabase("mflix");
        db.getCollection("users").deleteMany(new Document("email", email));
        db.getCollection("users").deleteMany(new Document("email",
"log@out.com"));
        db.getCollection("sessions").deleteMany(new Document("user_id",
"log@out.com"));
    }

    @Test
    public void test_T1_InvalidLogin() {
        userDao.addUser(testUser);
        boolean result = userDao.createUserSession(testUser.getEmail(), jwt);
        assertTrue("Should be able to create user session.", result);
        Session session = userDao.getUserSession(testUser.getEmail() +
"testWrong");
        assertTrue(Objects.isNull(session));
    }

    @Test
    public void test_T3_ValidLogin() {
        userDao.addUser(testUser);
        boolean result = userDao.createUserSession(testUser.getEmail(), jwt);
        assertTrue("Should be able to create user sesssion.", result);
        Session session = userDao.getUserSession(testUser.getEmail());
        assertEquals(
                "The user email needs to match the `session` user_id field",
                testUser.getEmail(),
                session.getUserId());
        assertEquals("jwt key needs to match the session `jwt`", jwt,
session.getJwt());
    }

    @Test
    public void test_T4_UserCreatesComment() {
        userDao.addUser(testUser);
        boolean result = userDao.createUserSession(testUser.getEmail(), jwt);
        assertTrue("Should be able to create user sesssion.", result);
```

```
        Session session = userDao.getUserSession(testUser.getEmail());
        assertEquals(
                "The user email needs to match the `session` user_id field",
                testUser.getEmail(),
                session.getUserId());
        assertEquals("jwt key needs to match the session `jwt`", jwt,
session.getJwt());
        Comment expectedComment = fakeCommentWithId();
        Assert.assertNotNull(
                "Comment should have been correctly added. Check your addComments
method",
                commentDao.addComment(expectedComment));

        Document actualComment =
                (Document) commentsCollection().find(Filters.eq("_id",
expectedComment.getOid())).first();

        Assert.assertNotNull("Comment should be found. Check your addComment
method", actualComment);

        Assert.assertEquals(
                "Comment email not matching. Check your addComment method",
                actualComment.getString("email"),
                expectedComment.getEmail());

        Assert.assertEquals(
                "Comment text not matching. Check your addComment method",
                actualComment.getString("text"),
                expectedComment.getText());

        Assert.assertEquals(
                "Comment date not matching. Check your addComment method",
                actualComment.getDate("date"),
                expectedComment.getDate());
    }

    @Test
    public void test_T5_UserUpdatesOwnComments() {
        userDao.addUser(testUser);
        boolean result = userDao.createUserSession(testUser.getEmail(), jwt);
        assertTrue("Should be able to create user session.", result);
        Comment fakeComment = fakeCommentWithId();
        commentDao.addComment(fakeComment);
        String expectedCommentText = randomText(20);

        Assert.assertTrue(
                "Should be able to update his own comments. Check updateComment
implementation",
                commentDao.updateComment(fakeComment.getId(),
expectedCommentText, validEmail));

        Document actualComment =
                (Document)
                        commentsCollection()
                                .find(new Document("_id", new
ObjectId(fakeCommentId)))
                                .first();

        Assert.assertEquals(
                "Comment text should match. Check updateComment implementation",
                expectedCommentText,
```

```
                actualComment.getString("text"));

        Assert.assertEquals("Commenter email should match the user email",
                validEmail, actualComment.getString("email"));
    }

    @Test
    public void test_T6_DeletesOwnedComment() {
        userDao.addUser(testUser);
        boolean result = userDao.createUserSession(testUser.getEmail(), jwt);
        assertTrue("Should be able to create user session.", result);
        Comment fakeComment = fakeCommentWithId();
        commentDao.addComment(fakeComment);
        String expectedCommentText = randomText(20);
        Assert.assertTrue(
                "Should be able to delete owns comments: Check your
deleteComment() method",
                commentDao.deleteComment(fakeComment.getId(),
testUser.getEmail()));
    }

    public void test_T7_UserSearchesForFilm() {
        userDao.addUser(testUser);
        boolean result = userDao.createUserSession(testUser.getEmail(), jwt);
        String cast = "Salma Hayek";
        List<Document> moviesInfo = movieDao.getMoviesCastFaceted(20, 0, cast);
        ArrayList<Document> allMovies = (ArrayList<Document>)
moviesInfo.get(0).get("movies");
        assertEquals(
                "Check your movies sub-pipeline on getMoviesFaceted() for
multiple cast in single cast",
                20,
                allMovies.size());
        ArrayList rating = (ArrayList<Document>) moviesInfo.get(0).get("rating");
        assertEquals(
                "Check your $bucket rating sub-pipeline on getMoviesFaceted() for
multiple cast in single cast",
                3,
                rating.size());
        ArrayList runtime = (ArrayList<Document>)
moviesInfo.get(0).get("runtime");
        assertEquals(
                "Check your $bucket runtime sub-pipeline on getMoviesFaceted()
for multiple cast in single cast",
                3,
                runtime.size());
    }

    public void test_T8_UserSearchesForInvalidFilm() {
        userDao.addUser(testUser);
        boolean result = userDao.createUserSession(testUser.getEmail(), jwt);
        String test = "qwygjjkgngk";
        List<Document> moviesInfo = movieDao.getMoviesCastFaceted(20, 0, test);
        ArrayList<Document> allMovies = (ArrayList<Document>)
moviesInfo.get(0).get("movies");
        assertEquals(
                0,
                allMovies.size());
    }

    public void test_T9_UserCanRateFilm() {
```

```
        userDao.addUser(testUser);
        boolean result = userDao.createUserSession(testUser.getEmail(), jwt);
        String rate = "1";
        String cast = "Salma Hayek";
        List<Document> moviesInfo = movieDao.getMoviesCastFaceted(20, 0, cast);
        ArrayList<Document> allMovies = (ArrayList<Document>)
moviesInfo.get(0).get("movies");
        result = ratingDao.setRating(allMovies.get(0), rate);
        assertTrue(result);
    }

    @Test
    public void test_T10_DeleteUser() {
        userDao.addUser(testUser);
        assertTrue(
                "You should be able to delete correctly the testDb user. Check
your delete filter",
                userDao.deleteUser(testUser.getEmail()));

        assertNull(
                "Should not find any sessions after deleting a user. deleteUser()
method needs to remove the user sessions data!",
                userDao.getUserSession(testUser.getEmail()));

        assertNull(
                "User data should not be found after user been deleted. Make sure
you delete data from users collection",
                userDao.getUser(testUser.getEmail()));
    }

    @Test
    public void test_T11_RegisterUserWithExistingEmail() {
        assertTrue(
                "Should have correctly created the user - check your write user
method",
                userDao.addUser(testUser)); // add string explanation

        User user = userDao.getUser(testUser.getEmail());
        Assert.assertEquals(testUser.getName(), user.getName());
        Assert.assertEquals(testUser.getEmail(), user.getEmail());
        Assert.assertEquals(testUser.getHashedpw(), user.getHashedpw());
        assertFalse(userDao.addUser(testUser));
    }


    private Comment fakeCommentWithId() {
        Comment comment = fakeCommentNoId();
        comment.setId(this.fakeCommentId);
        return comment;
    }

    private MongoCollection commentsCollection() {
        return
this.mongoClient.getDatabase("mflix").getCollection(CommentDao.COMMENT_COLLECTION
);
    }

    private Comment fakeCommentNoId() {
        String movieId = "573a1394f29313caabce0899";
        Comment comment = new Comment();
        comment.setEmail(validEmail);
```

```
        comment.setText(randomText(32));
        comment.setDate(new Date());
        comment.setName("some name");
        comment.setMovieId(movieId);
        return comment;
    }

    protected String randomText(int size) {
        return RandomStringUtils.random(size, true, true);
    }
}
```

Appendice C

Code for A9.2 Gatling

```
import io.gatling.core.Predef._
import io.gatling.http.Predef._

object User {

  val signUpFeeder = csv("data/signUpData.csv").circular

  def signUp = {
    feed(signUpFeeder)
      .exec { session => println(session); session }
      .exec(
        http("User Sign Up Action")
          .post("/api/v1/user/register")
          .formParam("name", "${name}")
          .formParam("email", "${email}")
          .formParam("password", "${password}")
          .check(status.is(200))
      )
      .exec(session => session.set("customerSignedIn", true))
      .exec { session => println(session); session }
  }
}
```

```
import io.gatling.core.Predef._
import io.gatling.http.Predef._

object SearchFilm {

  def searchFilm = {
    exec(
        http("Search Page")
          .get("/api/v1/movies/search")
          .check(status.is(200))
      )
  }

}
```

```
import io.gatling.core.Predef._
import io.gatling.http.Predef._

import scala.concurrent.duration.DurationInt
import scala.util.Random

import mflix.pageobjects._

class MflixSimulation extends Simulation {

  val domain = "localhost:5000"

  val httpProtocol = http
    .baseUrl("http://" + domain)
```

```
def userCount: Int = getProperty("USERS", "1000").toInt

def rampDuration: Int = getProperty("RAMP_DURATION", "10").toInt

def testDuration: Int = getProperty("DURATION", "60").toInt

val rnd = new Random()

def randomString(length: Int): String = {
  rnd.alphanumeric.filter(_.isLetter).take(length).mkString
}

before {
  println(s"Running test with ${userCount} users")
  println(s"Ramping users over ${rampDuration} seconds")
  println(s"Total test duration: ${testDuration} seconds")
}

after {
  println("Stress testing complete")
}

val initSession = exec(flushCookieJar)
  .exec { session => println(session); session }

object Scenarios {

  def default = scenario("Default Load Test")
    .during(testDuration.seconds) {
      randomSwitch(
        75d -> exec(UserJourneys.searchFilm),
        15d -> exec(UserJourneys.completeSignUp)
      )
    }

  def highLoad = scenario("High Purchase Load Test")
    .during(testDuration.seconds) {
      randomSwitch(
        25d -> exec(UserJourneys.searchFilm),
        25d -> exec(UserJourneys.completeSignUp)
      )
    }
}

object UserJourneys {

  def minPause = 100.milliseconds

  def maxPause = 500.milliseconds

  def searchFilm = {
    exec(initSession)
      .exec(SearchFilm.searchFilm)
  }

  def completeSignUp = {
    exec(initSession)
      .exec(User.signUp)
  }
}
```

```
  val scn = scenario("RecordedSimulation")
    .exec(initSession)
    .exec(User.signUp)
    .pause(2)
    .exec(SearchFilm.searchFilm)

  private def getProperty(propertyName: String, defaultValue: String) = {
    Option(System.getenv(propertyName))
      .orElse(Option(System.getProperty(propertyName)))
      .getOrElse(defaultValue)
  }

  //Parallel
  setUp(
    Scenarios.default
      .inject(rampUsers(userCount) during
(rampDuration.seconds)).protocols(httpProtocol)
  )
}
```