

TALLINN UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF SOFTWARE SCIENCE

Filmography website for the Ministry of Culture of Estonia

Request for Supply (excerpts)

Lab 2 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: 22.10.2022

Supervisor: Jaak Tepandi

A4. Test-driven development	3
4.1. Requirements.	3
4.2. Tools.	4
4.3. Test-driven development cycle.	4
A5. Code coverage.	7
5.1. Present the program.	7
5.2. Coverage tests	8
5.3. Test, evaluate	9
A6. Software for further testing.	11
6.1. Software information and installation guide	11
6.2. Architecture	11

A4. Test-driven development

4.1. Requirements.

Test cases were taken from the previous laboratory.

Use Case ID	UC_007
Use Case Name	Editing a comment
Primary Actor	System user
Preconditions	<ul style="list-style-type: none">• User has access to the comments section of the corresponding film• User has access to the comment text form• User has access to the “change a comment” button below comment that was created by this particular user
Postconditions	The comment is posted
Main Success Scenario	<ol style="list-style-type: none">1. User presses “change a comment” button2. User fills in the comment form3. User presses the “add a comment” button1. The comment is posted

Use Case ID	UC_008
Use Case Name	Successful deletion of the comment
Primary Actor	System user
Preconditions	<ul style="list-style-type: none">• User has access to the comments section of the corresponding film• User has access to the comment text form• User has access to the “delete a comment” button below the comment that was created by this particular user
Postconditions	The comment is deleted
Main Success Scenario	<ol style="list-style-type: none">1. User presses the “delete a comment” button2. The comment is deleted

Use Case ID	UC_009
Use Case Name	Search films
Primary Actor	System user
Preconditions	<ul style="list-style-type: none"> User has access to the search field
Postconditions	The list of films filtered by the search text is presented
Main Success Scenario	<ol style="list-style-type: none"> User enters a text in the search field The films are filtered by the following condition: name of the film contains the search text

4.2. Tools.

As the main programming language for the back-end part was chosen Java 8, and for the front-end part - JavaScript. Unit test tool - JUnit 4. VSC - GitLab.

4.3. Test-driven development cycle.

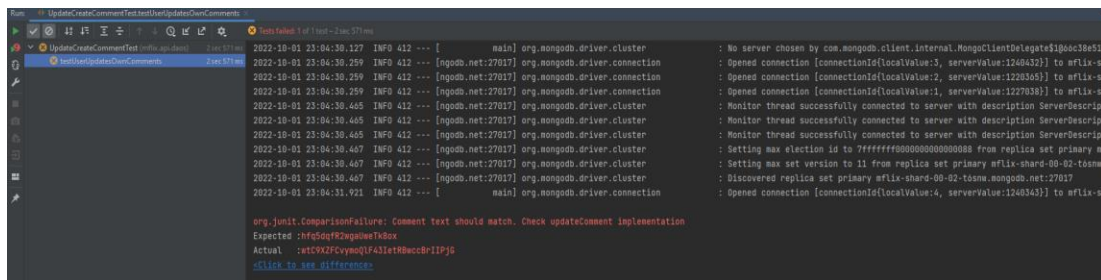
Step 1: Add a test for UC_007

```

81  @Test
82  public void testUserUpdatesOwnComments() {
83      Comment fakeComment = fakeCommentWithId();
84      dao.addComment(fakeComment);
85      String expectedCommentText = randomText(size: 20);
86
87      Assert.assertTrue(
88          message: "Should be able to update his own comments. Check updateComment implementation",
89          dao.updateComment(fakeComment.getId(), expectedCommentText, validEmail));
90
91      Document actualComment =
92          (Document)
93              commentsCollection() MongoCollection
94                  .find(new Document("_id", new ObjectId(fakeCommentId))) FindIterable
95                  .first();
96
97      Assert.assertEquals(
98          message: "Comment text should match. Check updateComment implementation",
99          expectedCommentText,
100         actualComment.getString(key: "text"));
101
102      Assert.assertEquals(message: "Commenter email should match the user email",
103                          validEmail, actualComment.getString(key: "email"));
104  }
105

```

Step 2: Run all tests and see if the new test fails



Failure of the test: The actual result is the comment that was created first, as expected. Result was to receive the updated comment, so the test failed.

Step 3: Write the code The functionality is added so the test is correctly performed

```
public boolean updateComment(String commentId, String text, String email) {
    Bson commentFilter = Filters.and(Filters.eq( fieldName: "_id", new ObjectId(commentId)), Filters.eq( fieldName: "email", email));
    Optional<Comment> comment = Optional.ofNullable(commentCollection.find(commentFilter).first());
    if (comment.isPresent()) {
        UpdateResult updateResult = commentCollection.updateOne(commentFilter, set("text", text));
        return updateResult.wasAcknowledged();
    }
    return false;
}
```

Step 4: The test is passing



Step 1: Add a test for UC_008

```
@Test
public void testDeleteOfOwnedComment() {
    this.dao = new CommentDao(mongoClient, databaseName);

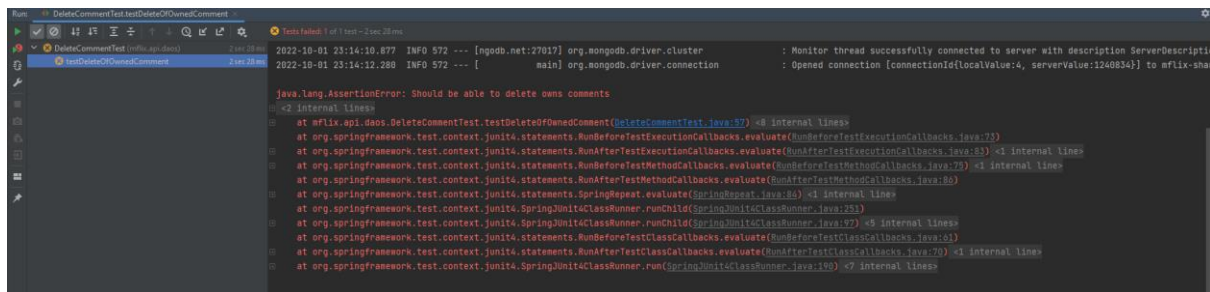
    Document commentDoc = new Document("email", ownerEmail);
    commentDoc.append("date", new Date());
    commentDoc.append("text", "some text");
    commentDoc.append("name", "user name");

    this.mongoClient.getDatabase(s: "mflix").getCollection(s: "comments").insertOne(commentDoc);

    commentId = commentDoc.getObjectId(key: "_id").toHexString();

    Assert.assertTrue(
        message: "Should be able to delete owns comments",
        dao.deleteComment(commentId, ownerEmail));
}
```

Step 2: Run all tests and see if the new test fails



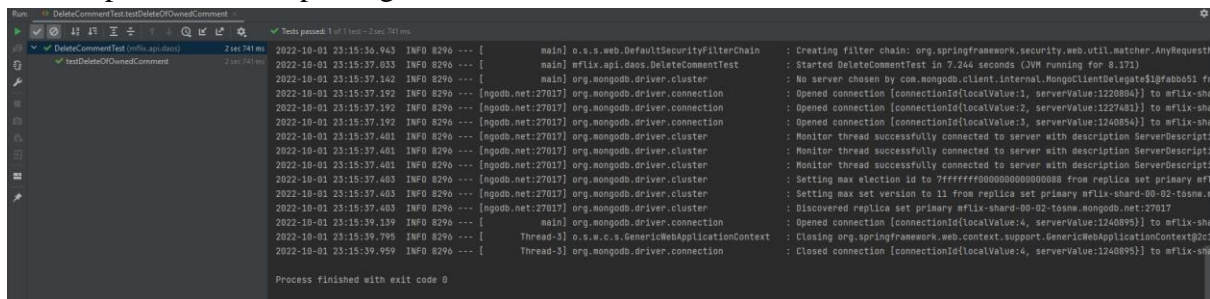
```
java.lang.AssertionError: Should be able to delete own comments
    at mflix.api.daos.DeleteCommentTest.testDeleteOwnedComment(DeleteCommentTest.java:57) <4 internal lines>
    at org.springframework.test.context.junit4.statements.RunBeforeTestExecutionCallbacks.evaluate(RunBeforeTestExecutionCallbacks.java:73)
    at org.springframework.test.context.junit4.statements.RunAfterTestExecutionCallbacks.evaluate(RunAfterTestExecutionCallbacks.java:63) <1 internal lines>
    at org.springframework.test.context.junit4.statements.RunBeforeTestMethodCallbacks.evaluate(RunBeforeTestMethodCallbacks.java:75) <1 internal lines>
    at org.springframework.test.context.junit4.statements.RunAfterTestMethodCallbacks.evaluate(RunAfterTestMethodCallbacks.java:86)
    at org.springframework.test.context.junit4.statements.SpringRepeat.evaluate(SpringRepeat.java:84) <1 internal lines>
    at org.springframework.test.runner.SpringJUnit4ClassRunner.runChild(SpringJUnit4ClassRunner.java:151)
    at org.springframework.test.context.junit4.SpringJUnit4ClassRunner.runChild(SpringJUnit4ClassRunner.java:97) <5 internal lines>
    at org.springframework.test.context.junit4.statements.RunBeforeTestClassCallbacks.evaluate(RunBeforeTestClassCallbacks.java:61)
    at org.springframework.test.context.junit4.statements.RunAfterTestClassCallbacks.evaluate(RunAfterTestClassCallbacks.java:70) <1 internal lines>
    at org.springframework.test.context.junit4.SpringJUnit4ClassRunner.run(SpringJUnit4ClassRunner.java:190) <7 internal lines>
```

Failure of the test: the user is not able to delete own comment.

Step 3: Write the code The functionality is added to the functional works as expected

```
public boolean deleteComment(String commentId, String email) {
    Bson commentFilter = Filters.and(Filters.eq( fieldName: "_id", new ObjectId(commentId)), Filters.eq( fieldName: "email", email));
    Optional<Comment> comment = Optional.ofNullable(commentCollection.find(commentFilter).first());
    if (comment.isPresent()) {
        commentCollection.deleteOne(commentFilter);
        return true;
    }
    return false;
}
```

Step 4: The test is passing



```
Tests passed: 3 of 1 test - 2sec 741ms
Process finished with exit code 0
```

Step 1: Add a test for UC_009

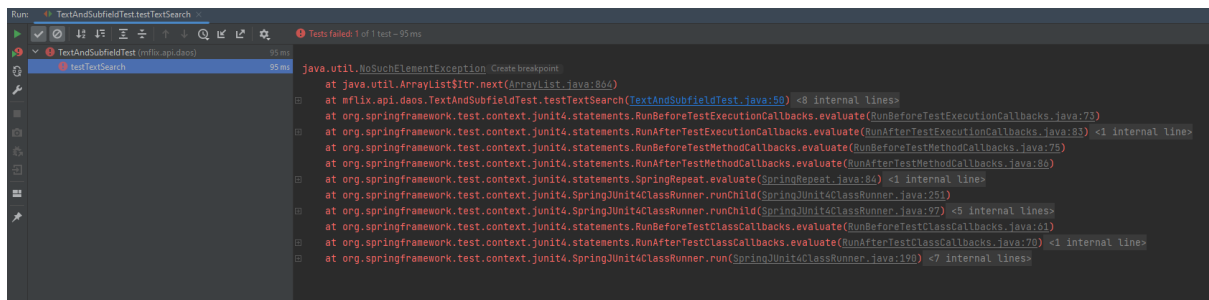
```
@Test
public void testTextSearch() {
    String keywords = "dress";
    int skip = 0;
    int limit = 20;

    Iterable<Document> cursor = dao.getMoviesByText(limit, skip, keywords);
    Document firstMovie = cursor.iterator().next();
    Assert.assertEquals( message: "Movie title does not match expected.", expected: "The Dress", firstMovie.getString( key: "title"));

    int actualMoviesCount = 0;
    for (Document doc : cursor) {
        System.out.println(doc);
        actualMoviesCount++;
    }

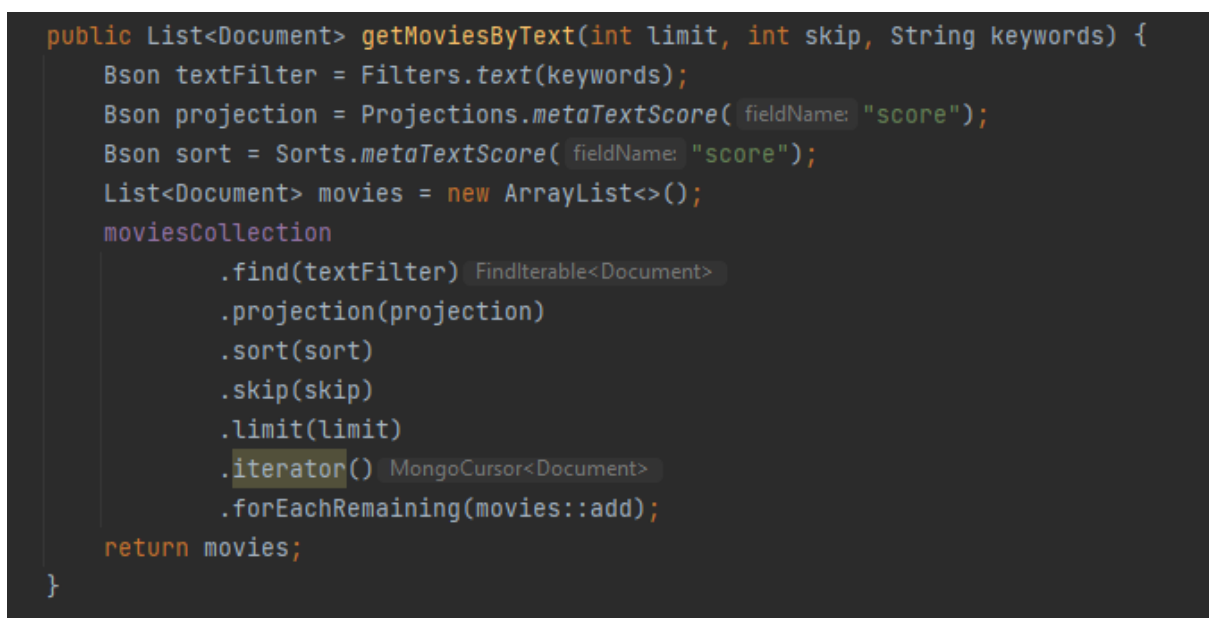
    Assert.assertEquals( message: "The expect number of movies does not match.", limit, actualMoviesCount);
}
```

Step 2: Run all tests and see if the new test fails

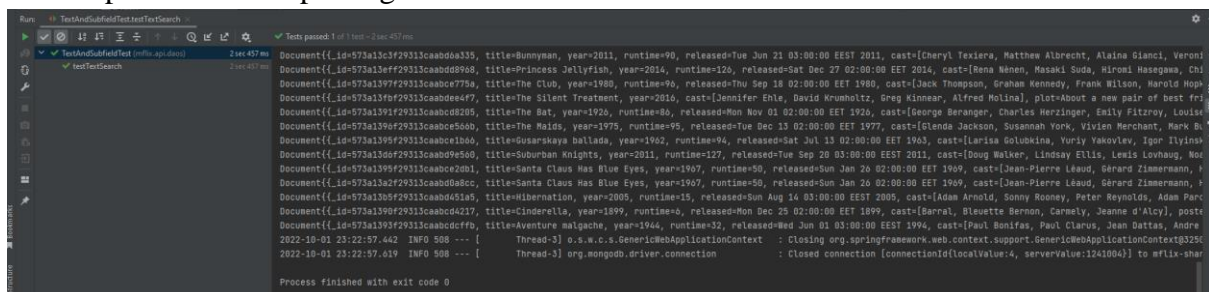


Failure of the test: The query returned no data for the given search pattern, so the test failed.

Step 3: Write the code The functionality is added to the query is correctly performed



Step 4: The test is passing



A5. Code coverage.

5.1. Present the program.

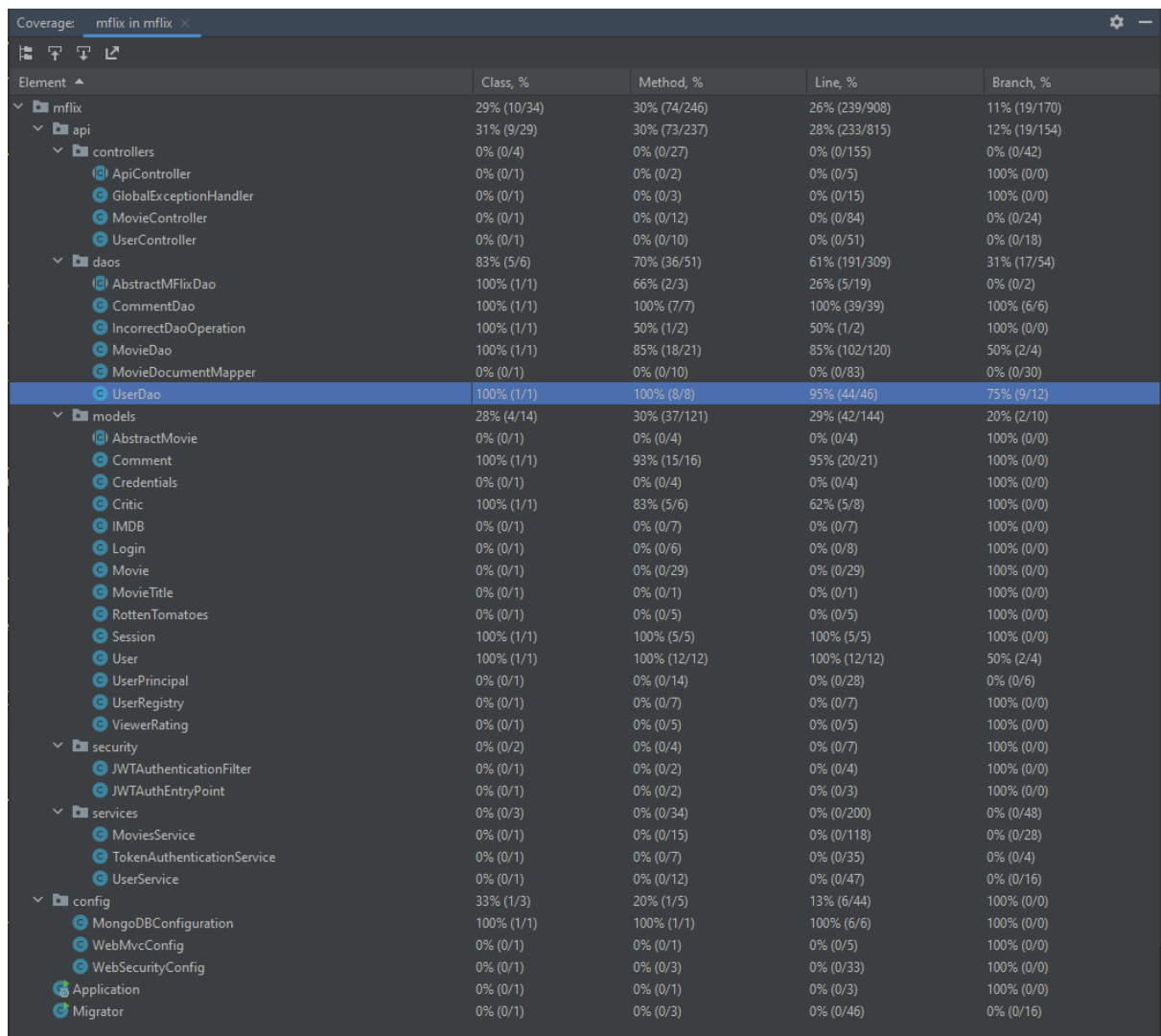
<https://gitlab.cs.ttu.ee/viabak/idy0204>

Our program - mflix - contains data about films. The project is built with microservices and has front-end and back-end parts. Also, it has unit tests.

5.2. Coverage tests

IntelliJ Idea is used as IDE for development and unit testing. This IDE makes it possible to run the test suit with different metrics (class coverage, method coverage, line coverage and branch coverage).

Branch coverage is displayed in the right column for all classes in the developing system:



Element	Class, %	Method, %	Line, %	Branch, %
mflix	29% (10/34)	30% (74/246)	26% (239/908)	11% (19/170)
api	31% (9/29)	30% (73/237)	28% (233/815)	12% (19/154)
controllers	0% (0/4)	0% (0/27)	0% (0/155)	0% (0/42)
ApiController	0% (0/1)	0% (0/2)	0% (0/5)	100% (0/0)
GlobalExceptionHandler	0% (0/1)	0% (0/3)	0% (0/15)	100% (0/0)
MovieController	0% (0/1)	0% (0/12)	0% (0/84)	0% (0/24)
UserController	0% (0/1)	0% (0/10)	0% (0/51)	0% (0/18)
daos	83% (5/6)	70% (36/51)	61% (191/309)	31% (17/54)
AbstractMflixDao	100% (1/1)	66% (2/3)	26% (5/19)	0% (0/2)
CommentDao	100% (1/1)	100% (7/7)	100% (39/39)	100% (6/6)
IncorrectDaoOperation	100% (1/1)	50% (1/2)	50% (1/2)	100% (0/0)
MovieDao	100% (1/1)	85% (18/21)	85% (102/120)	50% (2/4)
MovieDocumentMapper	0% (0/1)	0% (0/10)	0% (0/83)	0% (0/30)
UserDao	100% (1/1)	100% (8/8)	95% (44/46)	75% (9/12)
models	28% (4/14)	30% (37/121)	29% (42/144)	20% (2/10)
AbstractMovie	0% (0/1)	0% (0/4)	0% (0/4)	100% (0/0)
Comment	100% (1/1)	93% (15/16)	95% (20/21)	100% (0/0)
Credentials	0% (0/1)	0% (0/4)	0% (0/4)	100% (0/0)
Critic	100% (1/1)	83% (5/6)	62% (5/8)	100% (0/0)
IMDB	0% (0/1)	0% (0/7)	0% (0/7)	100% (0/0)
Login	0% (0/1)	0% (0/6)	0% (0/8)	100% (0/0)
Movie	0% (0/1)	0% (0/29)	0% (0/29)	100% (0/0)
MovieTitle	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)
RottenTomatoes	0% (0/1)	0% (0/5)	0% (0/5)	100% (0/0)
Session	100% (1/1)	100% (5/5)	100% (5/5)	100% (0/0)
User	100% (1/1)	100% (12/12)	100% (12/12)	50% (2/4)
UserPrincipal	0% (0/1)	0% (0/14)	0% (0/28)	0% (0/6)
UserRegistry	0% (0/1)	0% (0/7)	0% (0/7)	100% (0/0)
ViewerRating	0% (0/1)	0% (0/5)	0% (0/5)	100% (0/0)
security	0% (0/2)	0% (0/4)	0% (0/7)	100% (0/0)
JWTAuthenticationFilter	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)
JWTAuthEntryPoint	0% (0/1)	0% (0/2)	0% (0/3)	100% (0/0)
services	0% (0/3)	0% (0/34)	0% (0/200)	0% (0/48)
MoviesService	0% (0/1)	0% (0/15)	0% (0/118)	0% (0/28)
TokenAuthenticationService	0% (0/1)	0% (0/7)	0% (0/35)	0% (0/4)
UserService	0% (0/1)	0% (0/12)	0% (0/47)	0% (0/16)
config	33% (1/3)	20% (1/5)	13% (6/44)	100% (0/0)
MongoDBConfiguration	100% (1/1)	100% (1/1)	100% (6/6)	100% (0/0)
WebMvcConfig	0% (0/1)	0% (0/1)	0% (0/5)	100% (0/0)
WebSecurityConfig	0% (0/1)	0% (0/3)	0% (0/33)	100% (0/0)
Application	0% (0/1)	0% (0/1)	0% (0/3)	100% (0/0)
Migrator	0% (0/1)	0% (0/3)	0% (0/46)	0% (0/16)

Analyzed data from the coverage report:

- 0% for such classes as MovieController, UserController, AbstractMflixDao, MovieDocumentMapper, UserPrincipal, Migrator, UserService, TokenAuthenticationService, MoviesService - in most cases these classes are utility classes or will be covered in further iterations (sprints).

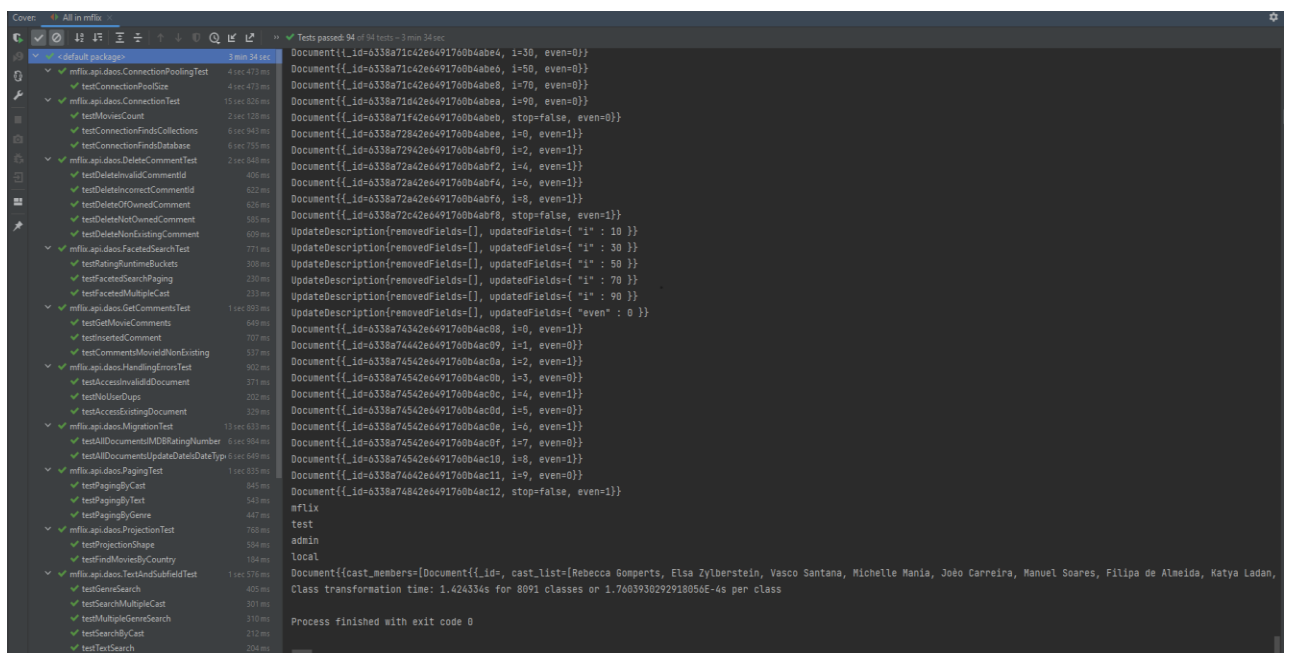
- 50-75% UserDao, MovieDao, User - these classes are partially covered, so we covered the most important functionality and expect further coverage in the next iterations

- 100% most of the classes - functionality in these classes is the most important. So to be sure that the system works as expected it was fully covered with tests.

5.3. Test, evaluate

To evaluate product quality unit tests were executed. Unit tests were written using JUnit4 tool which means that they could be easily run using IntelliJ Idea IDE. the tests using a unit test tool.

Furthermore, this IDE has built-in coverage tools. For example, it's possible to evaluate coverage for each class, method or line. So, tests were run with a coverage flag and such results were received.



Coverage: All in mflix			
Element	Class, %	Method, %	Line, %
✓ mflix	29% (10/34)	30% (74/246)	26% (239/908)
api	31% (9/29)	30% (73/237)	28% (233/815)
controllers	0% (0/4)	0% (0/27)	0% (0/155)
ApiController	0% (0/1)	0% (0/2)	0% (0/5)
GlobalExceptionHandler	0% (0/1)	0% (0/3)	0% (0/15)
MovieController	0% (0/1)	0% (0/12)	0% (0/84)
UserController	0% (0/1)	0% (0/10)	0% (0/51)
daos	83% (5/6)	70% (36/51)	61% (191/309)
AbstractMFlixDao	100% (1/1)	66% (2/3)	26% (5/19)
CommentDao	100% (1/1)	100% (7/7)	100% (39/39)
IncorrectDaoOperation	100% (1/1)	50% (1/2)	50% (1/2)
MovieDao	100% (1/1)	85% (18/21)	85% (102/120)
MovieDocumentMapper	0% (0/1)	0% (0/10)	0% (0/83)
UserDao	100% (1/1)	100% (8/8)	95% (44/46)
models	28% (4/14)	30% (37/121)	29% (42/144)
AbstractMovie	0% (0/1)	0% (0/4)	0% (0/4)
Comment	100% (1/1)	93% (15/16)	95% (20/21)
Credentials	0% (0/1)	0% (0/4)	0% (0/4)
Critic	100% (1/1)	83% (5/6)	62% (5/8)
IMDB	0% (0/1)	0% (0/7)	0% (0/7)
Login	0% (0/1)	0% (0/6)	0% (0/8)
Movie	0% (0/1)	0% (0/29)	0% (0/29)
MovieTitle	0% (0/1)	0% (0/1)	0% (0/1)
RottenTomatoes	0% (0/1)	0% (0/5)	0% (0/5)
Session	100% (1/1)	100% (5/5)	100% (5/5)
User	100% (1/1)	100% (12/12)	100% (12/12)
UserPrincipal	0% (0/1)	0% (0/14)	0% (0/28)
UserRegistry	0% (0/1)	0% (0/7)	0% (0/7)
ViewerRating	0% (0/1)	0% (0/5)	0% (0/5)
security	0% (0/2)	0% (0/4)	0% (0/7)
JWTAuthenticationFilter	0% (0/1)	0% (0/2)	0% (0/4)
JWTAuthEntryPoint	0% (0/1)	0% (0/2)	0% (0/3)
services	0% (0/3)	0% (0/34)	0% (0/200)
MoviesService	0% (0/1)	0% (0/15)	0% (0/118)
TokenAuthenticationService	0% (0/1)	0% (0/7)	0% (0/35)
UserService	0% (0/1)	0% (0/12)	0% (0/47)
config	33% (1/3)	20% (1/5)	13% (6/44)
MongoDBConfiguration	100% (1/1)	100% (1/1)	100% (6/6)
WebMvcConfig	0% (0/1)	0% (0/1)	0% (0/5)
WebSecurityConfig	0% (0/1)	0% (0/3)	0% (0/33)
Application	0% (0/1)	0% (0/1)	0% (0/3)
Migrator	0% (0/1)	0% (0/3)	0% (0/46)

Evaluate the quality of the program and adequacy of testing:

Analyzed data from the coverage report:

- 0% for classes in such packages as models, security, and services -these classes are utility classes, and it is considered that they will be covered in further iterations (sprints).
- 0< and < 100% most of the classes - functionality in these classes is the most important. So to be sure that the system works as expected it was covered with tests.

A6. Software for further testing.

6.1. Software information and installation guide

Application information: the application is located in a public TalTech GitLab repository with the possibility to be hosted both on-premise and in the cloud. The software contains in itself secret credentials that are impossible to upload to a private repository.

Software name: MFLIX

Version: 0.01

Author: MongoDB University

Reference: no public original repository available

License: MIT License

Installation guide:

Prerequisites: access to the repository, Java 8+ installed in the environment, MongoDB environment available to connect to.

- Clone the GitLab repository
- Synchronize the Maven dependencies
- Add an application.properties file with the following secrets:
 - jwtSecret
 - jwtExpirationInMs
 - spring.mongodb.uri
 - spring.mongodb.database
 - server.port
- In order to create a jar the following block of code should be added to the pom.xml file:
 - `<packaging>jar</packaging>`
- Build the project via CLI with the command `mvn package` or `mvn install`.
- To launch the application please execute the following command: `java -jar mflix.jar`

Additional requirements:

For further customization, you can add the following properties in the application.properties file (reasonable working defaults are provided):

- `logging.level.api.controllers=DEBUG`
- `api.movies.movies_per_page=20`
- `spring.resources.static-locations=classpath:/META-INF/resources/,classpath:/resources/,classpath:/static/,classpath:/build`

6.2. Architecture

The architecture is based on the MVC pattern.

The following diagram describes further the software architecture.

