HL9X35
Advanced OOP
24/25

Test Documentation

Kamil Milej
2264629

Contents

1) Test Plan

Test Strategy

The purpose of this document is to present automated tests. In the project, JUnit tests for Java were used.
One class was created following the TDD (Test-Driven Development) approach, meaning that the test was written first before the creation of the LoginController class, which contains the authenticate method that verifies the login and password. The test was designed to fail initially because the LoginController class did not yet exist.
After the class was implemented, the test passed, confirming that the test was correctly written and aligned with the test plan. The remaining tests were written after the corresponding classes had been implemented, as it was initially unclear which classes would require testing and what the final structure of the project would look like. The tests are not focused on integration or manual testing. Integration testing would have required a separate JSON file, which was considered unnecessary for the scope of this project. Additionally, manual tests were not included, as they were not part of the project requirements.

Tools and Technologies

In this project was used following tools:

JUnit 5 - for writing and executing automated unit tests in Java.
Gson - a library used for handling JSON file operations such as saving and loading player data.
Eclipse - used as the main development and testing environment.

Test Objectives

- ✓ To verify that key functionalities registration, login, score handling, and session control work as expected.
- ✓ To confirm that the code behaves correctly with valid and invalid input.
- ✓ To ensure that the application meets the functional requirements defined during the development phase.
- ✓ To detect and correct any logic or data processing errors.
- ✓ To validate the correctness of data saving and loading using JSON files.

2) Test Documentation

Class Name : PlayerTest			Test Name : shouldAddScoreCorrectly	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test01	Add score to player	+50, then -30	Score updated (count) correctly	Pass
Comment	Used assertEquals to compare the new score value with the expected result, ensuring that the score is updated correctly after calling addScore().			
Class Name : PlayerTest			Test Name : shouldReturnCorrectName	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test02	Return formatted player name	First name, last name, usernam	"Jan Kowalski username: janek"	pass
Comment	Used assertEquals to verify that the player's name is correctly formatted and returned by the method getPlayerName().			

Class Name :	SessionTest		Test Name : shouldLoginAndLogoutCorrectly	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test03	Session login and logout	Player object	Logged in, then logged out; session cleared	pass
Comment	Used assertTrue, assertFalse, and assertNull to validate session state after login and logout actions.			
Class Name :	LoginControllerTest		Test Name : shouldAuthenticateValidUser	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test04	Correct login and password	"brakney", "123qwe" (existing test account)	Player object returned	pass
Comment	Used assertNotNull and assertEquals to verify that the user is authenticated and returned correctly.			
Class Name :	LoginControllerTest		Test Name : shouldFailForInvalidPassword	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test05	Invalid password for valid login	"brakney", "wrongPassword"	Null (authentication fails-password wrong).	pass
Comment	Used assertNull to confirm that authentication fails when the password is incorrect.			
Class Name :	RegisterControllerTest		Test Name : shouldNotRegisterDuplicateLoginOrUsername	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test06	Duplicate login or username registration	("kmilej", "kmilej", "kmilej", "123qwe", "123qwe", "123qwe")	Null (registration fails)	pass
Comment	Used assertNull to verify that registration fails when using already existing login or username.			
Class Name :	LoginControllerTest		Test Name : shouldFailForEmptyCredentials	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test07	Empty login and password	("", "");	Null (authentication fails)	pass
Comment	Used assertNull to validate that empty credentials are rejected.			
Class Name :	PlayerTest		Test Name : testGamePlayedGetter	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test08	Get and update number of games played	Initially 0 → setGameplayed(+1)	Value becomes 1	pass
Comment	Used assertEquals to verify that the getter returns 0 initially and 1 after incrementing via setGameplayed(), confirming proper behavior of the getter/setter methods.			
Class Name :	RegisterControllerTest		Test Name : shouldNotRegisterDuplicateLoginOrUsernameEmpty	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test09	Reject registration with empty login and username	All identifying fields are empty strings	Registration fails (null)	pass
Comment	Used assertNull to confirm that the system correctly rejects registration attempts with empty login and username fields.			
Class Name :	LoginControllerTest		Test Name : shouldRejectInvalidLogin	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test10	Reject login with invalid credentials	"wrongLogin", "wrongPass"	Authentication fails (null)	pass
Comment	Used assertNull to ensure that authentication fails when the login is correct but the password is invalid.			

Class Name : LoginControllerTest			Test Name : shouldReturnNullWhenPasswordWrong	
Test Case ID	Test Description	Input Data	Expected Result	Actual Result & Status
Test11	Reject correct login with wrong password	"kmilej", "zlehaslo"	Authentication fails (null)	pass
Comment	Used assertNull to ensure that authentication fails when the login is correct but the password is invalid.			

3)Results of Test Runs

Summary of Test Results

Total number of test cases: 11
Passed: 11
Failed: 0
Retested: 0

Example Test Output

Below is a list of test made:

PlayerTest:
- shouldAddScoreCorrectly
- shouldReturnCorrectName
- testGamePlayedGetter

SessionTest:
- shouldLoginAndLogoutCorrectly

LoginControllerTest:
- shouldAuthenticateValidUser
- shouldFailForInvalidPassword
- shouldFailForEmptyCredentials
- shouldRejectInvalidLogin
- shouldReturnNullWhenPasswordWrong

RegisterControllerTest:
- shouldNotRegisterDuplicateLoginOrUsername
- shouldNotRegisterDuplicateLoginOrUsernameEmpty

4)Evaluation of Test Results

Analysis

All tests passed successfully, except for the ones written at the beginning using the TDD approach, which initially failed as expected. This indicates that the main assumptions of the application have been fulfilled and all core functionalities are working correctly. No errors occurred during the current phase of implementation.

Final Conclusion

The testing process confirmed that the system meets the functional and technical requirements. The code is stable, performs correctly, and is ready for further development, integration, or deployment.