Tesztelés jegyzőkönyv

Tesztelés során felhasznált eszközök

Az egységtesztelést a JUnit egységtesztelő keretrendszer segítségével végeztem. A JUnit az xUnit egységtesztelő keretrendszerek családjába tartozik. A nevében a „J” utal arra, hogy Java programozási nyelven íródott forráskódok teszteléséhez készült, és a teszteket szintén Java programozási nyelven kell implementálni. A JUnit támogatja az iteratív és inkrementális tesztelési és fejlesztési irányelveket. Ilyen például a teszt vezérelt fejlesztést (TDD – Test Driven Development), amely azt jelenti, hogy az új funkció lefejlesztése előtt, először a kódot tesztelő osztályokat implementáljuk, és csak ezután fejlesztjük le a tényleges funkciót. A JUnit-nak már több verziója is elérhető. A tesztelést a legújabb verzióval (JUnit 5) valósítottam meg.

A tesztek determinisztikus lefutását Mockito segítségével biztosítottam. Mockito egy open source tesztelő keretrendszer, amely lehetővé teszi a Java programozási nyelven íródott egységtesztekhez szükséges objektumok mock megfelelőinek létrehozását. Ezen kívül Mockito használatával előre meg tudjuk határozni az egyes külső, független metódusok viselkedését. Ezáltal az automatizált teszt minden lefutás során ugyanazt az eredményt produkálja.

Tesztelt osztályok

org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFrom.java: Az általam implementált „*SELECT * FROM <table_name>*” funkció egy fontos része az *SQLQueryTransformerSelectAllFrom* osztály, amely a megadott tábla lekérdezését végzi. Az osztály a betöltött szerepét tekintve egy rendkívül kritikus része a megvalósított funkciónak, ezért teszteltem külön.

org.jkiss.dbeaver.ui.editors.sql.util.SQLWordFinder.java: Az általam implementált „*SELECT * FROM <table_name>*” funkció egy másik, hasonlóan fontos része az *SQLWordFinder* osztály, amely a megadott tábla nevének azonosítását végzi. Ez azért egy kritikus része a funkciónak, hiszen a szövegszerkesztő kontextusából kell kinyerni egy létező adattábla nevet. Az osztályt a betöltött szerepe miatt teszteltem külön.

Létrehozott tesztcsomag bemutatása

org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java: Az *SQLQueryTransformerSelectAllFrom* osztályon belüli *transformQuery()* metódus tesztelését végzi. Ezen metódus állítja össze a megadott tábla lekérdezéséhez szükséges SQL lekérdezést. Két esetet teszteltem. Az egyik az ideális futás, ahol ellenőriztem, hogy a megadott táblanév esetén helyesen jön-e létre az SQL lekérdezés. A másik esetben nem adtam meg a táblanevet és egy kivétel képződését vizsgáltam.

org.jkiss.dbeaver.ui.editors.sql.util.SQLWordFinderTest.java: Az *SQLWordFinder* osztályon belüli *findWord()*, *getWordStartOffset()* és *getWordEndOffset()* metódusok tesztelését végzi. A *getWordStartOffset()* metódus a megadott szöveg egy tetszőleges pontjából indulva az első lehetséges szó kezdetének pozícióját adja vissza. A tesztelés során külön vizsgáltam a metódus viselkedését szöveg megadása nélkül, a megadott szöveg vizsgálatát nem megfelelő pozícióból indítva, valamint szóközt tartalmazó és szóközt nem tartalmazó szövegek megadása esetén. A *getWordEndOffset()* a megadott szöveg egy tetszőleges pontjából indulva az első lehetséges szó végének pozícióját adja vissza. A tesztelés során külön vizsgáltam a metódus viselkedését szóközt tartalmazó, valamint szóközt nem tartalmazó szövegek megadása esetén. A *findWord()* metódus a megtalált szó régióját határozza meg, amely tartalmazza az adott szó elejének és végének pozícióit. Ez a metódus használja az előző kettőt, ezért külön hangsúlyt kellett fektetni a determinisztikus futásra. A tesztelés során külön vizsgáltam a metódus viselkedését szöveg megadása nélkül, valamint amikor a kezdeti és a végső pozíciók megegyeznek, és amikor nem.

Létrehozott tesztek pontos megadása

org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java:

1. *testTransformQueryWhenIdentifierGiven()*:
    - Arrange: 16-19. sorok
    - Act: 20. sor
    - Assert: 21-20. sorok

2. *testTransformQueryWhenNoSpecifiedIdentifier()*:
    - Arrange: 27-30. sorok
    - Act: 31. sor
    - Assert: 32-33. sorok

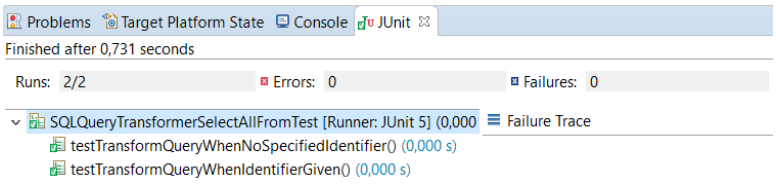
org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java:

1. *findWordWhenDocumentIsEmpty()*:
    - Arrange: 15-16. sorok
    - Act: 17. sor
    - Assert: 18. sor
2. *findWordWhenStartAndEndIsEquals()*:
    - Arrange: 23-24. sorok
    - Act: 25. sor
    - Assert: 26-27. sorok
3. *findWordWhenStartAndOffsetIsNotEquals()*:
    - Arrange: 23-33. sorok
    - Act: 34. sor
    - Assert: 35-36. sorok
4. *testGetWordStartOffsetWhenTextIsNull()*:
    - Arrange: 41. sor
    - Act: 41. sor
    - Assert: 42-43. sorok
5. *testGetWordStartOffsetWhenStartIndexIsInvalid()*:
    - Arrange: 48. sor
    - Act: 48. sor
    - Assert: 49-50. sorok
6. *testGetWordStartOffsetWhenTextHasWhiteSpace()*:
    - Arrange: 55. sor
    - Act: 55. sor
    - Assert: 56-57. sorok

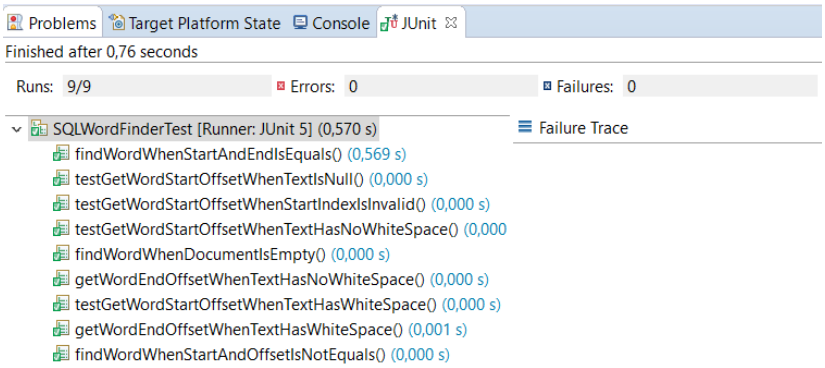
7. *testGetWordStartOffsetWhenTextHasNoWhiteSpace()*
    - Arrange: 62. sor
    - Act: 62. sor
    - Assert: 63-64. sorok
8. *getWordEndOffsetWhenTextHasWhiteSpace()*:
    - Arrange: 69. sor
    - Act: 69. sor
    - Assert: 70-71. sorok
9. *getWordEndOffsetWhenTextHasNoWhiteSpace()*:
    - Arrange: 76. sor
    - Act: 76. sor
    - Assert: 77-78. sorok

Végrehajtott tesztek sikerességének megállapítása

org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java: Mind a két teszt sikeresen átment.



org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java: Mind a kilenc teszt sikeresen átment.



Lefedettségi jelentés

Lefedettség mérés során felhasznált eszközök

A kód lefedettséget SonarQoube segítségével mértem. A SonarQoube egy open source szoftver, amely a forráskód minőségének ellenőrzésére és követésére szolgál. A SonarQoube képes a Java programozási nyelven íródott forráskódok feldolgozására. Különböző grafikonokat készít a forráskód minőségi kritériumairól. Ilyen kritérium lehet például a duplikált kód mennyisége, a kód szabványok betartása, unit tesztek mennyisége és az általuk lefedett forráskód, a forráskód komplexitása, esetleges hibák száma, dokumentáltság.

Lefedettségi érték indoklása

Én az *org.jkiss.dbeaver.ui.editors.sql* csomag lefedettségét mértem. Mivel ezen csomaghoz egyetlen egységteszt sem tartozik, így nem nehéz megállapítani, hogy a lefedettsége 0%.

org.jkiss.dbeaver.ui.editors.sql csomag méret metrikái:

1. Lines of Code: 18,060
2. Lines: 25,362
3. Statements: 7,842
4. Functions: 1,233
5. Classes: 215
6. Files: 151
7. Comment Lines: 1,335
8. Comments (%): 6.9%

org.jkiss.dbeaver.ui.editors.sql csomag tényleges lefedettsége:

1. Coverage: 0.0%
2. Lines to Cover: 9,229
3. Uncovered Lines: 9,229
4. Lines Coverage: 0.0%

Ez az érték azért lehet probléma, mert az alkalmazás egy nagyon fontos funkciójának a magjáról beszélünk, ugyanis ez a csomag a szíve az SQL szerkesztő felületnek. A lefedettségi értékekből ítélve viszont megkérdőjelezhető ezen funkció tényleges működése.

Erősen és kevésbe lefedett kódelemek bemutatása

Az *SQLEditorBase.java* és az *SQLEditor.java* két kulcsfontosságú osztály. Ezen osztályok lefedése lényeges kódminőség javuláshoz vezethetne. Ez a két osztály része volt az általam fejlesztett „*SELECT * FROM <table_name>*” funkciónak. Sajnos mivel az egész csomag lefedettsége is aggasztó volt, így ezen két osztályé is.

org.jkiss.dbeaver.ui.editors.sql.SQLEditorBase.java méret metrikái:

1. Lines of Code: 674
2. Lines: 887
3. Statements: 316
4. Functions: 61
5. Classes: 3
6. Files: 1
7. Comment Lines: 75
8. Comments (%): 10.0%

org.jkiss.dbeaver.ui.editors.sql.SQLEditorBase.java tényleges lefedettsége:

1. Coverage: 0.0%
2. Lines to Cover: 381
3. Uncovered Lines: 381
4. Lines Coverage: 0.0%

org.jkiss.dbeaver.ui.editors.sql.SQLEditor.java méret metrikái:

9. Lines of Code: 3,013
10. Lines: 3,543
11. Statements: 1,631
12. Functions: 169
13. Classes: 15
14. Files: 1
15. Comment Lines: 146
16. Comments (%): 4.6%

org.jkiss.dbeaver.ui.editors.sql.SQLEditor.java tényleges lefedettsége:

5. Coverage: 0.0%
6. Lines to Cover: 1,817
7. Uncovered Lines: 1,817
8. Lines Coverage: 0.0%

Könnyen és nehezen tesztelhető funkciók és változások

A korábban említett két kulcsfontosságú osztály tesztelése sajnos nem valósulhatott meg, ugyanis a példányosításuk, mock-olásuk nem volt lehetséges. Ez a kódminőségnek tudható be, ugyanis ezen osztályokban számos kódrész szerepel, amelyeket nem lehet felülírni. Ezenkívül rengeteg függőséggel rendelkeznek, melyek mock-olása jóformán lehetetlen. Ezek és az ezekhez hasonló osztályok esetében először a kódminőség javítására kellene fókuszálni, hiszen anélkül nehéz lesz tesztelni és tesztelés által lefedettséget növelni. A lefedettségi érték tökéletesen tükrözi ezen gondokat.