Sprawozdanie z Etapu 6: Testowanie Jednostkowe

Oleksandr Radionenko (274003), Yaroslav Perepilka (282279)

January 26, 2026

Contents

1 Wstęp

Niniejsze sprawozdanie zawiera kod testów jednostkowych oraz zestawów testów zorganizowany według zadań i przypadków użycia.

2 Zadanie 1: Testy jednostkowe bez mockowania

2.1 Przypadek użycia: Dodanie filmu do oferty

W tym zadaniu testujemy klasy bez symulacji zależności (bez mockowania).

2.1.1 model.TestFilm

Plik: `src/test/java/model/TestFilm.java`

```java
package model;

import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Testy jednostkowe dla klasy Film.
 * Testuje encję danych filmu - podstawową strukturę przechowującą informacje o
 * filmie.
 *
 * Przypadek użycia: Dodanie filmu do oferty
 * Warstwa: Encja (model)
 * Zadanie: 1 (testy bez mockowania)
 */
@DisplayName("Testy klasy Film - encja danych")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@Tag("encja")
@Tag("dodawanie")
class TestFilm {

    // Dane testowe przygotowywane przed każdym testem
    private Film film;
    private static final String TEST_ID = "F001";
    private static final String TEST_TYTUL = "Matrix";
    private static final String TEST_OPIS = "Cyberpunk thriller";
    private static final int TEST_CZAS = 136;
    private static final String TEST_GATUNEK = "SciFi";
    private static final double TEST_CENA = 28.5;

    @BeforeAll
    static void setUpBeforeClass() {
        // Przygotowanie przed wszystkimi testami
        System.out.println("Rozpoczęcie testów klasy Film");
    }

```

```
41      @BeforeEach
42      void setUp() {
43          // Jeśli: Przygotowanie danych testowych przed każdym testem
44          // Tworzymy nowy obiekt Film z określonymi danymi
45          film = new Film(TEST_ID, TEST_TYTUL, TEST_OPIS, TEST_CZAS,
                TEST_GATUNEK, TEST_CENA);
46      }
47
48      @AfterEach
49      void tearDown() {
50          // Sprzątanie po każdym teście
51          film = null;
52      }
53
54      @AfterAll
55      static void tearDownAfterClass() {
56          // Sprzątanie po wszystkich testach
57          System.out.println("Zakończenie testów klasy Film");
58      }
59
60      // ========== TESTY KONSTRUKTORA ==========
61
62      @Test
63      @Order(1)
64      @DisplayName("Test tworzenia filmu przez konstruktor")
65      void testTworzenieFilmu() {
66          // Jeśli: Dane do utworzenia filmu zostały przygotowane w
                setUp()
67
68          // Gdy: Film został utworzony w setUp()
69
70          // Wtedy: Obiekt filmu nie powinien być null i powinien mieć
                poprawne dane
71          assertNotNull(film, "Film nie powinien być null po utworzeniu");
72          assertInstanceOf(Film.class, film, "Obiekt powinien być
                instancją klasy Film");
73          assertTrue(film instanceof IFilm, "Film powinien implementować
                interfejs IFilm");
74      }
75
76      // ========== TESTY GETTERÓW ==========
77
78      @Test
79      @Order(2)
80      @DisplayName("Test metody dajId() - zwracanie identyfikatora")
81      void testDajId() {
82          // Jeśli: Film został utworzony z ID = "F001"
83
84          // Gdy: Pobieramy ID filmu
85          String id = film.dajId();
86
87          // Wtedy: ID powinno być równe wartości podanej w konstruktorze
88          assertNotNull(id, "ID nie powinno być null");
89          assertEquals(TEST_ID, id, "ID powinno być równe 'F001'");
90          assertTrue(id.startsWith("F"), "ID powinno zaczynać się od
                'F'");
91      }
92
```

```
93      @Test
94      @Order(3)
95      @DisplayName("Test metody dajTytul() - zwracanie tytułu")
96      void testDajTytul() {
97          // Jeśli: Film został utworzony z tytułem "Matrix"
98
99          // Gdy: Pobieramy tytuł filmu
100         String tytul = film.dajTytul();
101
102         // Wtedy: Tytuł powinien być równy wartości podanej w
                konstruktorze
103         assertNotNull(tytul, "Tytuł nie powinien być null");
104         assertEquals(TEST_TYTUL, tytul, "Tytuł powinien być równy
                'Matrix'");
105         assertFalse(tytul.isEmpty(), "Tytuł nie powinien być pusty");
106     }
107
108     @Test
109     @Order(4)
110     @DisplayName("Test metody dajOpis() - zwracanie opisu")
111     void testDajOpis() {
112         // Jeśli: Film został utworzony z opisem "Cyberpunk thriller"
113
114         // Gdy: Pobieramy opis filmu
115         String opis = film.dajOpis();
116
117         // Wtedy: Opis powinien być równy wartości podanej w
                konstruktorze
118         assertNotNull(opis, "Opis nie powinien być null");
119         assertEquals(TEST_OPIS, opis, "Opis powinien być równy wartości
                testowej");
120     }
121
122     @Test
123     @Order(5)
124     @DisplayName("Test metody dajCzasTrwania() - zwracanie czasu
            trwania")
125     void testDajCzasTrwania() {
126         // Jeśli: Film został utworzony z czasem trwania 136 minut
127
128         // Gdy: Pobieramy czas trwania
129         int czas = film.dajCzasTrwania();
130
131         // Wtedy: Czas powinien być równy wartości podanej w
                konstruktorze
132         assertEquals(TEST_CZAS, czas, "Czas trwania powinien być równy
                136");
133         assertTrue(czas > 0, "Czas trwania powinien być dodatni");
134     }
135
136     @Test
137     @Order(6)
138     @DisplayName("Test metody dajGatunek() - zwracanie gatunku")
139     void testDajGatunek() {
140         // Jeśli: Film został utworzony z gatunkiem "SciFi"
141
142         // Gdy: Pobieramy gatunek filmu
143         String gatunek = film.dajGatunek();
```

```
144
145         // Wtedy: Gatunek powinien być równy wartości podanej w
                konstruktorze
146         assertNotNull(gatunek, "Gatunek nie powinien być null");
147         assertEquals(TEST_GATUNEK, gatunek, "Gatunek powinien być równy
                'SciFi'");
148     }
149
150     @Test
151     @Order(7)
152     @DisplayName("Test metody dajCeneSeansow() - zwracanie ceny")
153     void testDajCeneSeansow() {
154         // Jeśli: Film został utworzony z ceną 28.5 PLN
155
156         // Gdy: Pobieramy cenę seansów
157         double cena = film.dajCeneSeansow();
158
159         // Wtedy: Cena powinna być równa wartości podanej w
                konstruktorze
160         assertEquals(TEST_CENA, cena, 0.01, "Cena powinna być równa
                28.5");
161         assertTrue(cena > 0, "Cena powinna być dodatnia");
162     }
163
164     // ========== TESTY PARAMETRYZOWANE - @CsvSource ==========
165
166     @ParameterizedTest
167     @Order(8)
168     @DisplayName("Test tworzenia filmów z różnymi danymi - @CsvSource")
169     @CsvSource({
170             "F001, Inception, Thriller psychologiczny, 148, Thriller,
                    32.0",
171             "F002, Avatar, Fantasy SciFi, 162, SciFi, 35.0",
172             "F003, Titanic, Romans epicki, 195, Dramat, 25.0",
173             "F004, Joker, Studium postaci, 122, Dramat, 30.0"
174     })
175     void testTworzenieFilmowZRoznymiDanymi(String id, String tytul,
            String opis,
176             int czas, String gatunek, double cena) {
177         // Jeśli: Dane filmu z parametrów CSV
178
179         // Gdy: Tworzymy film z tymi danymi
180         Film testFilm = new Film(id, tytul, opis, czas, gatunek, cena);
181
182         // Wtedy: Wszystkie pola powinny być poprawnie ustawione
183         assertEquals(id, testFilm.dajId(), "ID powinno być poprawne");
184         assertEquals(tytul, testFilm.dajTytul(), "Tytuł powinien być
                poprawny");
185         assertEquals(opis, testFilm.dajOpis(), "Opis powinien być
                poprawny");
186         assertEquals(czas, testFilm.dajCzasTrwania(), "Czas powinien
                być poprawny");
187         assertEquals(gatunek, testFilm.dajGatunek(), "Gatunek powinien
                być poprawny");
188         assertEquals(cena, testFilm.dajCeneSeansow(), 0.01, "Cena
                powinna być poprawna");
189     }
190
```

```
191     // ========== TESTY PARAMETRYZOWANE - @ValueSource ==========
192
193     @ParameterizedTest
194     @Order(9)
195     @DisplayName("Test filmów z różnymi czasami trwania - @ValueSource")
196     @ValueSource(ints = { 60, 90, 120, 150, 180, 240 })
197     void testFilmyZRoznymCzasemTrwania(int czas) {
198         // Jeśli: Różne czasy trwania filmu
199
200         // Gdy: Tworzymy film z określonym czasem
201         Film testFilm = new Film("FT", "Test", "Opis", czas, "Gatunek",
                20.0);
202
203         // Wtedy: Czas trwania powinien być poprawnie zapisany
204         assertEquals(czas, testFilm.dajCzasTrwania(),
205                 "Czas trwania powinien być równy " + czas);
206         assertTrue(testFilm.dajCzasTrwania() > 0, "Czas powinien być
                dodatni");
207     }
208
209     @ParameterizedTest
210     @Order(10)
211     @DisplayName("Test filmów z różnymi cenami - @ValueSource")
212     @ValueSource(doubles = { 10.0, 15.5, 20.0, 25.99, 30.0, 35.5, 50.0
            })
213     void testFilmyZRoznaCena(double cena) {
214         // Jeśli: Różne ceny seansów
215
216         // Gdy: Tworzymy film z określoną ceną
217         Film testFilm = new Film("FT", "Test", "Opis", 120, "Gatunek",
                cena);
218
219         // Wtedy: Cena powinna być poprawnie zapisana
220         assertEquals(cena, testFilm.dajCeneSeansow(), 0.01,
221                 "Cena powinna być równa " + cena);
222         assertTrue(testFilm.dajCeneSeansow() >= 10.0, "Cena powinna być
                >= 10.0");
223     }
224
225     // ========== TESTY WARTOŚCI BRZEGOWYCH ==========
226
227     @Test
228     @Order(11)
229     @DisplayName("Test filmu z minimalnym czasem trwania")
230     void testFilmMinimalnyCzas() {
231         // Jeśli: Film ma minimalny czas trwania (1 minuta)
232
233         // Gdy: Tworzymy film z czasem 1 minuty
234         Film krotki = new Film("FK", "Krótki", "Opis", 1, "Short", 5.0);
235
236         // Wtedy: Film powinien być poprawnie utworzony
237         assertEquals(1, krotki.dajCzasTrwania());
238         assertNotNull(krotki.dajTytul());
239     }
240
241     @Test
242     @Order(12)
243     @DisplayName("Test filmu z bardzo długim czasem trwania")
```

```
244     void testFilmDlugiCzas() {
245         // Jeśli: Film ma bardzo długi czas trwania
246 
247         // Gdy: Tworzymy film z czasem 300 minut
248         Film dlugi = new Film("FD", "Długi", "Opis", 300, "Epic", 40.0);
249 
250         // Wtedy: Film powinien być poprawnie utworzony
251         assertEquals(300, dlugi.dajCzasTrwania());
252         assertTrue(dlugi.dajCzasTrwania() > 180, "Film powinien być
                dłuższy niż 3 godziny");
253     }
254 
255     // ========== TESTY NIEZMIENNICZOŚCI ==========
256 
257     @Test
258     @Order(13)
259     @DisplayName("Test niezmienniczości danych filmu")
260     void testNiezmiennoscDanych() {
261         // Jeśli: Film został utworzony z określonymi danymi
262         String originalId = film.dajId();
263         String originalTytul = film.dajTytul();
264         double originalCena = film.dajCeneSeansow();
265 
266         // Gdy: Pobieramy dane wielokrotnie
267         String id2 = film.dajId();
268         String tytul2 = film.dajTytul();
269         double cena2 = film.dajCeneSeansow();
270 
271         // Wtedy: Dane powinny pozostać niezmienione
272         assertSame(originalId, id2, "ID nie powinno się zmienić");
273         assertSame(originalTytul, tytul2, "Tytuł nie powinien się
                zmienić");
274         assertEquals(originalCena, cena2, 0.001, "Cena nie powinna się
                zmienić");
275     }
276 }
```

2.1.2 model.TestDAO

Plik: src/test/java/model/TestDAO.java

```
1  package model;
2
3  import org.junit.jupiter.api.*;
4  import org.junit.jupiter.api.Tag;
5  import org.junit.jupiter.params.ParameterizedTest;
6  import org.junit.jupiter.params.provider.CsvSource;
7  import org.junit.jupiter.params.provider.ValueSource;
8
9  import static org.junit.jupiter.api.Assertions.*;
10
11 /**
12  * Testy jednostkowe dla klasy DAO.
13  * Testuje operacje dodawania i wyszukiwania danych.
14  */
15 @DisplayName("Testy klasy DAO")
16 @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
17 @Tag("encja")
18 @Tag("dodawanie")
19 class TestDAO {
20
21     private DAO dao;
22
23     @BeforeAll
24     static void setUpBeforeClass() {
25         // Przygotowanie przed wszystkimi testami
26         System.out.println("Rozpoczęcie testów DAO");
27     }
28
29     @BeforeEach
30     void setUp() {
31         // Jeśli: Utworzenie nowego DAO przed każdym testem
32         dao = new DAO();
33     }
34
35     @AfterEach
36     void tearDown() {
37         // Sprzątanie po każdym teście
38         dao = null;
39     }
40
41     @AfterAll
42     static void tearDownAfterClass() {
43         // Sprzątanie po wszystkich testach
44         System.out.println("Zakończenie testów DAO");
45     }
46
47     @Test
48     @Order(1)
49     @DisplayName("Test dodawania filmu i generowania ID")
50     void testDodajFilm() {
51         // Jeśli: Dane filmu do zapisania
52         String daneFilmu =
               "F001;Avengers;Superbohaterowie;150;Akcja;30.0";
53
54         // Gdy: Dodajemy film do DAO
```

```
55          String id = dao.dodajFilm(daneFilmu);
56
57          // Wtedy: ID powinno być wygenerowane i film powinien być
                zapisany
58          assertNotNull(id, "ID nie powinno być null");
59          assertTrue(id.startsWith("F"), "ID powinno zaczynać się od F");
60          assertEquals("F001", id, "ID powinno być F001");
61      }
62
63      @Test
64      @Order(2)
65      @DisplayName("Test znajdowania filmu po ID")
66      void testZnajdzFilm() {
67          // Jeśli: Film został dodany do DAO
68          String daneFilmu = "F1;Matrix;SciFi;136;Akcja;28.0";
69          String id = dao.dodajFilm(daneFilmu);
70
71          // Gdy: Szukamy filmu po ID
72          String znalezionyFilm = dao.znajdzFilm(id);
73
74          // Wtedy: Powinniśmy znaleźć ten sam film
75          assertNotNull(znalezionyFilm, "Film powinien zostać
                znaleziony");
76          assertEquals(daneFilmu, znalezionyFilm, "Dane filmu powinny być
                identyczne");
77      }
78
79      @Test
80      @Order(3)
81      @DisplayName("Test szukania nieistniejącego filmu")
82      void testZnajdzFilmNieistniejacy() {
83          // Jeśli: DAO jest puste
84
85          // Gdy: Szukamy nieistniejącego filmu
86          String znalezionyFilm = dao.znajdzFilm("F999");
87
88          // Wtedy: Powinniśmy otrzymać null
89          assertNull(znalezionyFilm, "Nieistniejący film powinien zwrócić
                null");
90      }
91
92      @ParameterizedTest
93      @Order(4)
94      @DisplayName("Test dodawania filmów z różnymi ID")
95      @CsvSource({
96              "F1;Film1;Opis1;90;Komedia;20.0,
                   F2;Film2;Opis2;120;Dramat;25.0,
                   F3;Film3;Opis3;110;Akcja;30.0",
97              "F10;KomediaX;OpisX;95;Komedia;18.0,
                   F11;DramatY;OpisY;125;Dramat;22.5,
                   F12;AkcjaZ;OpisZ;100;Akcja;27.0"
98      })
99      void testInkrementacjaIdFilmow(String film1, String film2, String
            film3) {
100         // Jeśli: Dane kilku filmów (parametryzowane)
101
102         // Gdy: Dodajemy filmy kolejno
103         String id1 = dao.dodajFilm(film1);
```

```
104         String id2 = dao.dodajFilm(film2);
105         String id3 = dao.dodajFilm(film3);
106 
107         // Wtedy: ID powinny być takie same jak w danych
108         String[] parts1 = film1.split(";");
109         String[] parts2 = film2.split(";");
110         String[] parts3 = film3.split(";");
111         assertEquals(parts1[0], id1);
112         assertEquals(parts2[0], id2);
113         assertEquals(parts3[0], id3);
114         assertNotEquals(id1, id2, "ID powinny być różne");
115     }
116 
117     @Test
118     @Order(5)
119     @DisplayName("Test dodawania wpisu do logu")
120     void testDodajWpisDoLogu() {
121         // Jeśli: Treść zdarzenia do zalogowania
122         String zdarzenie = "Test zdarzenia";
123 
124         // Gdy: Dodajemy wpis do logu
125         // Wtedy: Nie powinien wystąpić wyjątek
126         assertDoesNotThrow(() -> {
127             dao.dodajWpisDoLogu(zdarzenie);
128         }, "Dodawanie wpisu do logu nie powinno rzucić wyjątku");
129     }
130 
131     @ParameterizedTest
132     @Order(6)
133     @DisplayName("Test dodawania filmów o różnych cenach")
134     @ValueSource(doubles = { 10.0, 15.5, 20.0, 25.99, 30.0 })
135     void testDodajFilmyRozneCeny(double cena) {
136         // Jeśli: Dane filmu z różnymi cenami
137         String daneFilmu = "FX;Film;Opis;120;Gatunek;" + cena;
138 
139         // Gdy: Dodajemy film
140         String id = dao.dodajFilm(daneFilmu);
141 
142         // Wtedy: Film powinien być zapisany
143         assertNotNull(id);
144         String znalezionyFilm = dao.znajdzFilm(id);
145         assertTrue(znalezionyFilm.contains(String.valueOf(cena)),
146                 "Znaleziony film powinien zawierać cenę");
147     }
148 }
```

2.1.3 model.TestFabrykaStandardowegoFilmu

Plik: src/test/java/model/TestFabrykaStandardowegoFilmu.java

```java
package model;

import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.MethodSource;

import java.util.stream.Stream;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Testy jednostkowe dla klasy FabrykaStandardowegoFilmu.
 * Testuje tworzenie filmów z danych wejściowych.
 */
@DisplayName("Testy klasy FabrykaStandardowegoFilmu")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@Tag("encja")
@Tag("dodawanie")
class TestFabrykaStandardowegoFilmu {

    private FabrykaStandardowegoFilmu fabryka;

    @BeforeAll
    static void setUpBeforeClass() {
        // Przygotowanie przed wszystkimi testami
        System.out.println("Rozpoczęcie testów
            FabrykaStandardowegoFilmu");
    }

    @BeforeEach
    void setUp() {
        // Jeśli: Utworzenie fabryki przed każdym testem
        fabryka = new FabrykaStandardowegoFilmu();
    }

    @AfterEach
    void tearDown() {
        // Sprzątanie po każdym teście
        fabryka = null;
    }

    @AfterAll
    static void tearDownAfterClass() {
        // Sprzątanie po wszystkich testach
        System.out.println("Zakończenie testów
            FabrykaStandardowegoFilmu");
    }

    @Test
    @Order(1)
    @DisplayName("Test tworzenia filmu z poprawnych danych")
    void testUtworzFilmPoprawne() {
        // Jeśli: Dane filmu w formacie CSV
```

```
54          String daneFilmu =
                "F001;Avengers;Superbohaterowie;150;Akcja;30.0";
55
56          // Gdy: Tworzymy film używając fabryki
57          IFilm film = fabryka.utworzFilm(daneFilmu);
58
59          // Wtedy: Film powinien być utworzony z poprawnymi danymi
60          assertNotNull(film, "Film nie powinien być null");
61          assertEquals("Avengers", film.dajTytul());
62          assertEquals("Superbohaterowie", film.dajOpis());
63          assertEquals(150, film.dajCzasTrwania());
64          assertEquals(30.0, film.dajCeneSeansow(), 0.01);
65      }
66
67      @Test
68      @Order(2)
69      @DisplayName("Test tworzenia filmu z minimalnymi danymi")
70      void testUtworzFilmMinimalne() {
71          // Jeśli: Minimalne dane filmu
72          String daneFilmu = "F002;Film;Opis;60;Dramat;10.0";
73
74          // Gdy: Tworzymy film
75          IFilm film = fabryka.utworzFilm(daneFilmu);
76
77          // Wtedy: Film powinien być utworzony
78          assertNotNull(film);
79          assertTrue(film.dajCzasTrwania() > 0, "Czas trwania powinien
                być dodatni");
80          assertTrue(film.dajCeneSeansow() > 0, "Cena powinna być
                dodatnia");
81      }
82
83      @ParameterizedTest
84      @Order(3)
85      @DisplayName("Test tworzenia filmów z różnymi danymi wejściowymi")
86      @CsvSource({
87              "F001, Titanic, Romans na statku, 195, Dramat, 25.0",
88              "F002, Matrix, Cyberpunk, 136, SciFi, 28.5",
89              "F003, Joker, Psychologiczny, 122, Thriller, 32.0"
90      })
91      void testUtworzFilmParametryzowany(String id, String tytul, String
            opis,
92              int czas, String gatunek, double cena) {
93          // Jeśli: Dane filmu z różnych źródeł
94          String daneFilmu = id + ";" + tytul + ";" + opis + ";" + czas +
                ";" + gatunek + ";" + cena;
95
96          // Gdy: Tworzymy film
97          IFilm film = fabryka.utworzFilm(daneFilmu);
98
99          // Wtedy: Wszystkie pola powinny być poprawnie ustawione
100         assertEquals(id, film.dajId());
101         assertEquals(tytul, film.dajTytul());
102         assertEquals(opis, film.dajOpis());
103         assertEquals(czas, film.dajCzasTrwania());
104         assertEquals(cena, film.dajCeneSeansow(), 0.01);
105     }
106
```

```
107     @ParameterizedTest
108     @Order(4)
109     @DisplayName("Test tworzenia filmów z różnymi cenami")
110     @MethodSource("dostarczDaneFilmow")
111     void testUtworzFilmZMetody(String daneFilmu, double oczekiwanaCena)
            {
112         // Jeśli: Dane filmu dostarczone z metody
113
114         // Gdy: Tworzymy film
115         IFilm film = fabryka.utworzFilm(daneFilmu);
116
117         // Wtedy: Cena powinna odpowiadać oczekiwanej
118         assertNotNull(film);
119         assertEquals(oczekiwanaCena, film.dajCeneSeansow(), 0.01);
120         assertFalse(film.dajTytul().isEmpty(), "Tytuł nie powinien być
                pusty");
121     }
122
123     static Stream<org.junit.jupiter.params.provider.Arguments>
            dostarczDaneFilmow() {
124         return Stream.of(
125                 org.junit.jupiter.params.provider.Arguments.of("F001;Film1;Opis1;90;
                        15.0),
126                 org.junit.jupiter.params.provider.Arguments.of("F002;Film2;Opis2;120
                        22.5),
127                 org.junit.jupiter.params.provider.Arguments.of("F003;Film3;Opis3;100
                        18.99));
128     }
129
130     @Test
131     @Order(5)
132     @DisplayName("Test wyjątku przy niepoprawnych danych")
133     void testUtworzFilmNiepoprawne() {
134         // Jeśli: Niepoprawne dane wejściowe (brak wystarczającej
                liczby pól)
135         String daneFilmu = "F001;Film";
136
137         // Gdy/Wtedy: Powinien wystąpić wyjątek
138         assertThrows(Exception.class, () -> {
139             fabryka.utworzFilm(daneFilmu);
140         }, "Powinien wystąpić wyjątek przy niepoprawnych danych");
141     }
142 }
```

2.1.4 model.TestModelDodawanieFilmu

Plik: src/test/java/model/TestModelDodawanieFilmu.java

```java
package model;

import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.MethodSource;

import java.util.stream.Stream;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Testy jednostkowe dla klasy Model - operacja dodawania filmu.
 * Testuje pełny przepływ dodawania filmu przez warstwę modelu.
 */
@DisplayName("Testy klasy Model - dodawanie filmu")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@Tag("encja")
@Tag("dodawanie")
class TestModelDodawanieFilmu {

    private Model model;
    private DAO dao;
    private Oferta oferta;

    @BeforeAll
    static void setUpBeforeClass() {
        // Przygotowanie przed wszystkimi testami
        System.out.println("Rozpoczęcie testów Model - dodawanie
            filmu");
    }

    @BeforeEach
    void setUp() {
        // Jeśli: Utworzenie systemu przed każdym testem
        dao = new DAO();
        oferta = new Oferta(dao);
        model = new Model(oferta, dao);
    }

    @AfterEach
    void tearDown() {
        // Sprzątanie po każdym teście
        model = null;
        oferta = null;
        dao = null;
    }

    @AfterAll
    static void tearDownAfterClass() {
        // Sprzątanie po wszystkich testach
        System.out.println("Zakończenie testów Model - dodawanie
            filmu");
    }
```

```
54
55      @Test
56      @Order(1)
57      @DisplayName("Test dodawania filmu przez Model")
58      void testDodajFilm() {
59          // Jeśli: Dane filmu w formacie CSV
60          String daneFilmu = "F001;Avengers;Superbohaterowie ratują ś
                wiat;150;Akcja;30.0";
61
62          // Gdy: Dodajemy film przez model
63          String wynik = model.dodajFilm(daneFilmu);
64
65          // Wtedy: Film powinien być dodany z komunikatem sukcesu
66          assertNotNull(wynik, "Wynik nie powinien być null");
67          assertTrue(wynik.contains("pomyslnie"), "Wynik powinien
                zawierać 'pomyslnie'");
68          assertTrue(wynik.contains("ID"), "Wynik powinien zawierać ID");
69          assertFalse(wynik.isEmpty(), "Wynik nie powinien być pusty");
70      }
71
72      @Test
73      @Order(2)
74      @DisplayName("Test dodawania filmu i weryfikacja w DAO")
75      void testDodajFilmWeryfikacjaDAO() {
76          // Jeśli: Dane filmu
77          String daneFilmu = "F001;Matrix;Cyberpunk
                thriller;136;SciFi;28.5";
78
79          // Gdy: Dodajemy film
80          String wynik = model.dodajFilm(daneFilmu);
81
82          // Wtedy: Film powinien istnieć w DAO
83          assertTrue(wynik.contains("F001"), "Wynik powinien zawierać ID
                F001");
84          String zapisanyFilm = dao.znajdzFilm("F001");
85          assertNotNull(zapisanyFilm, "Film powinien być zapisany w DAO");
86          assertTrue(zapisanyFilm.contains("Matrix"), "Zapisany film
                powinien zawierać tytuł");
87      }
88
89      @Test
90      @Order(3)
91      @DisplayName("Test formatu komunikatu zwrotnego")
92      void testFormatKomunikatu() {
93          // Jeśli: Dane filmu
94          String daneFilmu = "F001;Titanic;Romans;195;Dramat;25.0";
95
96          // Gdy: Dodajemy film
97          String wynik = model.dodajFilm(daneFilmu);
98
99          // Wtedy: Komunikat powinien mieć odpowiedni format
100         assertTrue(wynik.startsWith("Film dodany"), "Komunikat powinien
                zaczynać się od 'Film dodany'");
101         assertTrue(wynik.contains("ID:"), "Komunikat powinien zawierać
                'ID:'");
102     }
103
104     @Test
```

```
105     @Order(4)
106     @DisplayName("Test dodawania wielu filmów")
107     void testDodajWieleFilmow() {
108         // Jeśli: Dane kilku filmów
109         String film1 = "F001;Film1;Opis1;90;Komedia;20.0";
110         String film2 = "F002;Film2;Opis2;120;Dramat;25.0";
111         String film3 = "F003;Film3;Opis3;110;Akcja;30.0";
112 
113         // Gdy: Dodajemy filmy kolejno
114         String wynik1 = model.dodajFilm(film1);
115         String wynik2 = model.dodajFilm(film2);
116         String wynik3 = model.dodajFilm(film3);
117 
118         // Wtedy: Wszystkie filmy powinny być dodane z różnymi ID
119         assertTrue(wynik1.contains("F001"));
120         assertTrue(wynik2.contains("F002"));
121         assertTrue(wynik3.contains("F003"));
122         assertNotEquals(wynik1, wynik2, "Wyniki powinny być różne");
123         assertNotEquals(wynik2, wynik3, "Wyniki powinny być różne");
124     }
125 
126     @ParameterizedTest
127     @Order(5)
128     @DisplayName("Test dodawania filmów z różnymi danymi")
129     @CsvSource({
130             "F001, Inception, Thriller psychologiczny, 148, Thriller,
                32.0",
131             "F002, Joker, Studium postaci, 122, Dramat, 30.0",
132             "F003, Parasite, Społeczny dramat, 132, Dramat, 28.0"
133     })
134     void testDodajFilmParametryzowany(String id, String tytul, String
        opis,
135             int czas, String gatunek, double cena) {
136         // Jeśli: Dane filmu z parametrów
137         String daneFilmu = id + ";" + tytul + ";" + opis + ";" + czas +
            ";" + gatunek + ";" + cena;
138 
139         // Gdy: Dodajemy film
140         String wynik = model.dodajFilm(daneFilmu);
141 
142         // Wtedy: Film powinien być dodany
143         assertNotNull(wynik);
144         assertTrue(wynik.contains("pomyslnie"), "Wynik powinien
            zawierać potwierdzenie");
145         assertTrue(wynik.contains("ID:"), "Wynik powinien zawierać ID");
146     }
147 
148     @ParameterizedTest
149     @Order(6)
150     @DisplayName("Test dodawania filmów z metodą źródłową")
151     @MethodSource("dostarczDaneFilmow")
152     void testDodajFilmZMetody(String daneFilmu, String oczekiwanyTytul)
        {
153         // Jeśli: Dane filmu z metody źródłowej
154 
155         // Gdy: Dodajemy film
156         String wynik = model.dodajFilm(daneFilmu);
157 
```

```
158         // Wtedy: Film powinien być dodany i dostępny w DAO
159         assertNotNull(wynik);
160         assertTrue(wynik.contains("pomyslnie"));
161 
162         // Weryfikacja w DAO
163         String zapisanyFilm = dao.znajdzFilm("F001");
164         assertNotNull(zapisanyFilm, "Film powinien być w DAO");
165         assertTrue(zapisanyFilm.contains(oczekiwanyTytul),
166                 "Film powinien zawierać oczekiwany tytuł");
167     }
168 
169     static Stream<org.junit.jupiter.params.provider.Arguments>
        dostarczDaneFilmow() {
170         return Stream.of(
171                 org.junit.jupiter.params.provider.Arguments.of(
172                         "F001;Avatar;Fantasy SciFi;162;SciFi;35.0",
                            "Avatar"),
173                 org.junit.jupiter.params.provider.Arguments.of(
174                         "F002;Gladiator;Historyczny;155;Akcja;27.0",
                            "Gladiator"),
175                 org.junit.jupiter.params.provider.Arguments.of(
176                         "F003;Interstellar;Kosmos;169;SciFi;33.0",
                            "Interstellar"));
177     }
178 
179     @Test
180     @Order(7)
181     @DisplayName("Test integracji fabryki z modelem")
182     void testIntegracjaFabryki() {
183         // Jeśli: Dane filmu wymagające przetworzenia przez fabrykę
184         String daneFilmu =
            "F001;TestFilm;TestOpis;100;TestGatunek;20.0";
185 
186         // Gdy: Dodajemy film (co używa fabryki wewnętrznie)
187         String wynik = model.dodajFilm(daneFilmu);
188 
189         // Wtedy: Film powinien być utworzony przez fabrykę i zapisany
190         assertNotNull(wynik);
191         assertTrue(wynik.contains("pomyslnie"));
192 
193         // Weryfikacja że dane zostały przetworzone
194         String zapisanyFilm = dao.znajdzFilm("F001");
195         assertNotNull(zapisanyFilm);
196         assertTrue(zapisanyFilm.contains("TestFilm"));
197         assertTrue(zapisanyFilm.contains("20.0"));
198     }
199 }
```

2.1.5 controller.TestAdminControllerDodawanieFilmu

Plik: src/test/java/controller/TestAdminControllerDodawanieFilmu.java

```
1  package controller;
2
3  import model.*;
4  import org.junit.jupiter.api.*;
5  import org.junit.jupiter.api.Tag;
6  import org.junit.jupiter.params.ParameterizedTest;
7  import org.junit.jupiter.params.provider.CsvSource;
8  import org.junit.jupiter.params.provider.ValueSource;
9
10 import static org.junit.jupiter.api.Assertions.*;
11
12 /**
13  * Testy jednostkowe dla klasy AdminController - operacja dodawania
        filmu.
14  * Testuje pełny przepływ dodawania filmu przez kontroler
        administratora.
15  */
16 @DisplayName("Testy klasy AdminController - dodawanie filmu")
17 @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
18 @Tag("kontroler")
19 @Tag("dodawanie")
20 class TestAdminControllerDodawanieFilmu {
21
22     private AdminController adminController;
23     private Model model;
24     private DAO dao;
25     private Oferta oferta;
26
27     @BeforeAll
28     static void setUpBeforeClass() {
29         // Przygotowanie przed wszystkimi testami
30         System.out.println("Rozpoczęcie testów AdminController -
               dodawanie filmu");
31     }
32
33     @BeforeEach
34     void setUp() {
35         // Jeśli: Utworzenie pełnego systemu przed każdym testem
36         dao = new DAO();
37         oferta = new Oferta(dao);
38         model = new Model(oferta, dao);
39         adminController = new AdminController(model);
40     }
41
42     @AfterEach
43     void tearDown() {
44         // Sprzątanie po każdym teście
45         adminController = null;
46         model = null;
47         oferta = null;
48         dao = null;
49     }
50
51     @AfterAll
52     static void tearDownAfterClass() {
```

```
53          // Sprzątanie po wszystkich testach
54          System.out.println("Zakończenie testów AdminController -
                dodawanie filmu");
55      }
56
57      @Test
58      @Order(1)
59      @DisplayName("Test dodawania filmu przez AdminController")
60      void testDodajFilm() {
61          // Jeśli: Dane filmu w formacie CSV
62          String daneFilmu =
                "F001;Avengers;Superbohaterowie;150;Akcja;30.0";
63
64          // Gdy: Dodajemy film przez kontroler
65          String wynik = adminController.dodajFilm(daneFilmu);
66
67          // Wtedy: Film powinien być dodany pomyślnie
68          assertNotNull(wynik, "Wynik nie powinien być null");
69          assertTrue(wynik.contains("pomyslnie"), "Wynik powinien
                zawierać 'pomyslnie'");
70          assertTrue(wynik.contains("ID"), "Wynik powinien zawierać ID
                filmu");
71      }
72
73      @Test
74      @Order(2)
75      @DisplayName("Test dodawania filmu i weryfikacja zapisu")
76      void testDodajFilmWeryfikacja() {
77          // Jeśli: Dane filmu
78          String daneFilmu = "F001;Matrix;Cyberpunk;136;SciFi;28.5";
79
80          // Gdy: Dodajemy film
81          String wynik = adminController.dodajFilm(daneFilmu);
82
83          // Wtedy: Film powinien być w systemie
84          assertTrue(wynik.contains("F001"), "Wynik powinien zawierać
                ID");
85
86          // Weryfikacja w DAO
87          String zapisanyFilm = dao.znajdzFilm("F001");
88          assertNotNull(zapisanyFilm, "Film powinien być zapisany w
                bazie");
89          assertTrue(zapisanyFilm.contains("Matrix"), "Film powinien
                zawierać poprawny tytuł");
90      }
91
92      @Test
93      @Order(3)
94      @DisplayName("Test wzorca Strategy w dodawaniu filmu")
95      void testStrategiaEdycjiOferty() {
96          // Jeśli: Dane filmu do dodania
97          String daneFilmu = "F001;Inception;Thriller;148;Thriller;32.0";
98
99          // Gdy: Dodajemy film (używa strategii DodanieNowegoFilmu)
100         String wynik = adminController.dodajFilm(daneFilmu);
101
102         // Wtedy: Strategia powinna zostać poprawnie wykonana
103         assertNotNull(wynik);
```

```
104         assertTrue(wynik.contains("pomyslnie"), "Strategia powinna
                wykonać się pomyślnie");
105         assertFalse(wynik.isEmpty(), "Wynik nie powinien być pusty");
106     }
107
108     @Test
109     @Order(4)
110     @DisplayName("Test dodawania wielu filmów sekwencyjnie")
111     void testDodajWieleFilmow() {
112         // Jeśli: Dane kilku filmów
113         String film1 = "F001;Film1;Opis1;90;Komedia;20.0";
114         String film2 = "F002;Film2;Opis2;120;Dramat;25.0";
115         String film3 = "F003;Film3;Opis3;110;Akcja;30.0";
116
117         // Gdy: Dodajemy filmy kolejno
118         String wynik1 = adminController.dodajFilm(film1);
119         String wynik2 = adminController.dodajFilm(film2);
120         String wynik3 = adminController.dodajFilm(film3);
121
122         // Wtedy: Wszystkie filmy powinny być dodane z unikalnymi ID
123         assertTrue(wynik1.contains("F001"), "Pierwszy film powinien
                mieć ID F001");
124         assertTrue(wynik2.contains("F002"), "Drugi film powinien mieć
                ID F002");
125         assertTrue(wynik3.contains("F003"), "Trzeci film powinien mieć
                ID F003");
126
127         // Weryfikacja w DAO
128         assertNotNull(dao.znajdzFilm("F001"));
129         assertNotNull(dao.znajdzFilm("F002"));
130         assertNotNull(dao.znajdzFilm("F003"));
131     }
132
133     @Test
134     @Order(5)
135     @DisplayName("Test poprawności formatu komunikatu")
136     void testFormatKomunikatu() {
137         // Jeśli: Dane filmu
138         String daneFilmu = "F001;Titanic;Romans;195;Dramat;25.0";
139
140         // Gdy: Dodajemy film
141         String wynik = adminController.dodajFilm(daneFilmu);
142
143         // Wtedy: Komunikat powinien mieć określony format
144         assertTrue(wynik.startsWith("Film dodany"),
145                 "Komunikat powinien zaczynać się od 'Film dodany'");
146         assertTrue(wynik.contains("ID:"), "Komunikat powinien zawierać
                'ID:'");
147         assertTrue(wynik.contains("F"), "Komunikat powinien zawierać
                prefix ID");
148     }
149
150     @ParameterizedTest
151     @Order(6)
152     @DisplayName("Test dodawania filmów z różnymi danymi")
153     @CsvSource({
154             "F001, Parasite, Dramat społeczny, 132, Dramat, 28.0",
155             "F002, Joker, Psychologiczny, 122, Thriller, 30.0",
```

```
156             "F003, 1917, Wojenny, 119, Wojenny, 27.5",
157             "F004, Dune, Epicka SF, 155, SciFi, 35.0"
158     })
159     void testDodajFilmParametryzowany(String id, String tytul, String
            opis,
160             int czas, String gatunek, double cena) {
161         // Jeśli: Dane filmu z parametrów
162         String daneFilmu = id + ";" + tytul + ";" + opis + ";" + czas +
                ";" + gatunek + ";" + cena;
163
164         // Gdy: Dodajemy film
165         String wynik = adminController.dodajFilm(daneFilmu);
166
167         // Wtedy: Film powinien być dodany
168         assertNotNull(wynik, "Wynik nie powinien być null");
169         assertTrue(wynik.contains("pomyslnie"), "Film powinien być
                dodany pomyślnie");
170         assertTrue(wynik.contains("ID:"), "Wynik powinien zawierać ID");
171     }
172
173     @ParameterizedTest
174     @Order(7)
175     @DisplayName("Test dodawania filmów o różnych cenach")
176     @ValueSource(doubles = { 10.0, 15.5, 20.0, 25.99, 30.0, 35.5, 40.0
            })
177     void testDodajFilmRozneCeny(double cena) {
178         // Jeśli: Dane filmu z różnymi cenami
179         String daneFilmu = "F" + ((int) (cena * 10)) +
                ";Film;Opis;120;Gatunek;" + cena;
180
181         // Gdy: Dodajemy film
182         String wynik = adminController.dodajFilm(daneFilmu);
183
184         // Wtedy: Film powinien być dodany niezależnie od ceny
185         assertNotNull(wynik);
186         assertTrue(wynik.contains("pomyslnie"));
187
188         // Weryfikacja że cena jest zapisana
189         String zapisanyFilm = dao.znajdzFilm("F" + ((int) (cena * 10)));
190         assertNotNull(zapisanyFilm);
191         assertTrue(zapisanyFilm.contains(String.valueOf(cena)),
192                 "Zapisany film powinien zawierać cenę");
193     }
194
195     @Test
196     @Order(8)
197     @DisplayName("Test pełnego przepływu dodawania filmu")
198     void testPelnyPrzeplyw() {
199         // Jeśli: Kompletne dane filmu
200         String daneFilmu = "F001;Avatar;Fantasy epicki;162;SciFi;35.0";
201
202         // Gdy: Dodajemy film przez AdminController
203         // (co wywołuje EdytowanieOfertyKina -> DodanieNowegoFilmu ->
                Model -> Fabryka
204         // -> DAO)
205         String wynik = adminController.dodajFilm(daneFilmu);
206
207         // Wtedy: Cały przepływ powinien zakończyć się sukcesem
```

```
208         assertNotNull(wynik);
209         assertTrue(wynik.contains("pomyslnie"));
210 
211         // Weryfikacja na poziomie DAO
212         String zapisanyFilm = dao.znajdzFilm("F001");
213         assertNotNull(zapisanyFilm, "Film powinien być w bazie danych");
214         assertTrue(zapisanyFilm.contains("Avatar"), "Film powinien mieć
               poprawny tytuł");
215         assertTrue(zapisanyFilm.contains("35.0"), "Film powinien mieć
               poprawną cenę");
216     }
217 
218     @Test
219     @Order(9)
220     @DisplayName("Test że AdminController nie modyfikuje danych filmu")
221     void testNiezmiennoscDanych() {
222         // Jeśli: Oryginalne dane filmu
223         String oryginalneDane =
               "F001;Original;Description;100;Genre;20.0";
224 
225         // Gdy: Dodajemy film
226         String wynik = adminController.dodajFilm(oryginalneDane);
227 
228         // Wtedy: Dane w DAO powinny odpowiadać oryginalnym
229         String zapisanyFilm = dao.znajdzFilm("F001");
230         assertNotNull(zapisanyFilm);
231         assertTrue(zapisanyFilm.contains("Original"), "Tytuł nie
               powinien być zmieniony");
232         assertTrue(zapisanyFilm.contains("Description"), "Opis nie
               powinien być zmieniony");
233         assertTrue(zapisanyFilm.contains("100"), "Czas nie powinien być
               zmieniony");
234         assertTrue(zapisanyFilm.contains("20.0"), "Cena nie powinna być
               zmieniona");
235     }
236 }
```

2.2 Przypadek użycia: Przeglądanie repertuaru

W tym zadaniu testujemy klasy bez symulacji zależności (bez mockowania).

2.2.1 model.TestSeans

Plik: `src/test/java/model/TestSeans.java`

```java
package model;

import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Testy jednostkowe dla klasy Seans.
 * Testuje encję danych seansu - podstawową strukturę przechowującą informacje o
 * seansie filmowym.
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Warstwa: Encja (model)
 * Zadanie: 1 (testy bez mockowania)
 */
@DisplayName("Testy klasy Seans - encja danych")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@Tag("encja")
@Tag("repertuar")
class TestSeans {

    // Dane testowe przygotowywane przed każdym testem
    private Seans seans;
    private Film filmTestowy;
    private static final String TEST_ID = "S001";
    private static final String TEST_DATA = "2024-12-20 18:00";
    private static final String TEST_SALA = "Sala1";
    private static final int TEST_MIEJSCA = 100;

    @BeforeAll
    static void setUpBeforeClass() {
        // Przygotowanie przed wszystkimi testami
        System.out.println("Rozpoczęcie testów klasy Seans");
    }

    @BeforeEach
    void setUp() {
        // Jeśli: Przygotowanie danych testowych przed każdym testem
        // Tworzymy film testowy potrzebny do seansu
        filmTestowy = new Film("F001", "Matrix", "Cyberpunk thriller",
            136, "SciFi", 28.5);
        // Tworzymy nowy obiekt Seans z określonymi danymi
        seans = new Seans(TEST_ID, filmTestowy, TEST_DATA, TEST_SALA,
            TEST_MIEJSCA);
    }
```

```
48
49      @AfterEach
50      void tearDown() {
51          // Sprzątanie po każdym teście
52          seans = null;
53          filmTestowy = null;
54      }
55
56      @AfterAll
57      static void tearDownAfterClass() {
58          // Sprzątanie po wszystkich testach
59          System.out.println("Zakończenie testów klasy Seans");
60      }
61
62      // ========== TESTY KONSTRUKTORA ==========
63
64      @Test
65      @Order(1)
66      @DisplayName("Test tworzenia seansu przez konstruktor")
67      void testTworzenieSeansu() {
68          // Jeśli: Dane do utworzenia seansu zostały przygotowane w
                setUp()
69
70          // Gdy: Seans został utworzony w setUp()
71
72          // Wtedy: Obiekt seansu nie powinien być null i powinien mieć
                poprawne dane
73          assertNotNull(seans, "Seans nie powinien być null po
                utworzeniu");
74          assertInstanceOf(Seans.class, seans, "Obiekt powinien być
                instancją klasy Seans");
75          assertTrue(seans instanceof ISeans, "Seans powinien
                implementować interfejs ISeans");
76      }
77
78      // ========== TESTY GETTERÓW ==========
79
80      @Test
81      @Order(2)
82      @DisplayName("Test metody dajId() - zwracanie identyfikatora
            seansu")
83      void testDajId() {
84          // Jeśli: Seans został utworzony z ID = "S001"
85
86          // Gdy: Pobieramy ID seansu
87          String id = seans.dajId();
88
89          // Wtedy: ID powinno być równe wartości podanej w konstruktorze
90          assertNotNull(id, "ID nie powinno być null");
91          assertEquals(TEST_ID, id, "ID powinno być równe 'S001'");
92          assertTrue(id.startsWith("S"), "ID powinno zaczynać się od
                'S'");
93      }
94
95      @Test
96      @Order(3)
97      @DisplayName("Test metody dajFilm() - zwracanie filmu seansu")
98      void testDajFilm() {
```

```
99          // Jeśli: Seans został utworzony z filmem "Matrix"
100
101         // Gdy: Pobieramy film seansu
102         IFilm film = seans.dajFilm();
103
104         // Wtedy: Film powinien być równy filmowi podanemu w
                konstruktorze
105         assertNotNull(film, "Film nie powinien być null");
106         assertEquals(filmTestowy, film, "Film powinien być tym samym
                obiektem");
107         assertEquals("Matrix", film.dajTytul(), "Tytuł filmu powinien
                być 'Matrix'");
108     }
109
110     @Test
111     @Order(4)
112     @DisplayName("Test metody dajDate() - zwracanie daty seansu")
113     void testDajDate() {
114         // Jeśli: Seans został utworzony z datą "2024-12-20 18:00"
115
116         // Gdy: Pobieramy datę seansu
117         String data = seans.dajDate();
118
119         // Wtedy: Data powinna być równa wartości podanej w
                konstruktorze
120         assertNotNull(data, "Data nie powinna być null");
121         assertEquals(TEST_DATA, data, "Data powinna być równa wartości
                testowej");
122         assertFalse(data.isEmpty(), "Data nie powinna być pusta");
123     }
124
125     @Test
126     @Order(5)
127     @DisplayName("Test metody dajWolneMiejsca() - początkowa liczba
            wolnych miejsc")
128     void testDajWolneMiejscaPoczatkowo() {
129         // Jeśli: Seans został utworzony z 100 miejscami, żadne nie
                jest zajęte
130
131         // Gdy: Pobieramy wolne miejsca
132         int[] wolneMiejsca = seans.dajWolneMiejsca();
133
134         // Wtedy: Wszystkie miejsca powinny być wolne
135         assertNotNull(wolneMiejsca, "Tablica wolnych miejsc nie powinna
                być null");
136         assertEquals(TEST_MIEJSCA, wolneMiejsca.length, "Wszystkie 100
                miejsc powinno być wolnych");
137         assertEquals(1, wolneMiejsca[0], "Pierwsze wolne miejsce
                powinno mieć numer 1");
138         assertEquals(100, wolneMiejsca[99], "Ostatnie wolne miejsce
                powinno mieć numer 100");
139     }
140
141     // ========== TESTY REZERWACJI MIEJSC ==========
142
143     @Test
144     @Order(6)
145     @DisplayName("Test zarezerwujMiejsce() - udana rezerwacja")
```

```
146     void testZarezerwujMiejsceUdana() {
147         // Jeśli: Miejsce nr 50 jest wolne
148 
149         // Gdy: Rezerwujemy miejsce nr 50
150         boolean wynik = seans.zarezerwujMiejsce(50);
151 
152         // Wtedy: Rezerwacja powinna się udać
153         assertTrue(wynik, "Rezerwacja powinna się udać");
154 
155         // I liczba wolnych miejsc powinna się zmniejszyć o 1
156         int[] wolneMiejsca = seans.dajWolneMiejsca();
157         assertEquals(99, wolneMiejsca.length, "Powinno być 99 wolnych
                miejsc");
158     }
159 
160     @Test
161     @Order(7)
162     @DisplayName("Test zarezerwujMiejsce() - próba rezerwacji zajętego
            miejsca")
163     void testZarezerwujMiejsceZajete() {
164         // Jeśli: Miejsce nr 25 zostało już zarezerwowane
165         seans.zarezerwujMiejsce(25);
166 
167         // Gdy: Próbujemy zarezerwować to samo miejsce ponownie
168         boolean wynik = seans.zarezerwujMiejsce(25);
169 
170         // Wtedy: Rezerwacja powinna się nie udać
171         assertFalse(wynik, "Rezerwacja zajętego miejsca powinna się nie
                udać");
172     }
173 
174     @Test
175     @Order(8)
176     @DisplayName("Test zarezerwujMiejsce() - nieprawidłowy numer
            miejsca")
177     void testZarezerwujMiejsceNieprawidlowe() {
178         // Jeśli: Seans ma 100 miejsc (numery 1-100)
179 
180         // Gdy: Próbujemy zarezerwować miejsce o numerze 0 lub 101
181         boolean wynik0 = seans.zarezerwujMiejsce(0);
182         boolean wynik101 = seans.zarezerwujMiejsce(101);
183         boolean wynikUjemny = seans.zarezerwujMiejsce(-5);
184 
185         // Wtedy: Wszystkie rezerwacje powinny się nie udać
186         assertFalse(wynik0, "Rezerwacja miejsca 0 powinna się nie
                udać");
187         assertFalse(wynik101, "Rezerwacja miejsca 101 powinna się nie
                udać");
188         assertFalse(wynikUjemny, "Rezerwacja miejsca -5 powinna się nie
                udać");
189     }
190 
191     @Test
192     @Order(9)
193     @DisplayName("Test zwolnijMiejsce() - zwalnianie zarezerwowanego
            miejsca")
194     void testZwolnijMiejsce() {
195         // Jeśli: Miejsce nr 30 zostało zarezerwowane
```

```
196         seans.zarezerwujMiejsce(30);
197         int wolnychPoRezerwacji = seans.dajWolneMiejsca().length;
198 
199         // Gdy: Zwalniamy miejsce nr 30
200         seans.zwolnijMiejsce(30);
201 
202         // Wtedy: Miejsce powinno być ponownie dostępne
203         int wolnychPoZwolnieniu = seans.dajWolneMiejsca().length;
204         assertEquals(wolnychPoRezerwacji + 1, wolnychPoZwolnieniu,
205                 "Liczba wolnych miejsc powinna wzrosnąć o 1");
206     }
207 
208     // ========== TESTY PARAMETRYZOWANE - @CsvSource ==========
209 
210     @ParameterizedTest
211     @Order(10)
212     @DisplayName("Test tworzenia seansów z różnymi danymi - @CsvSource")
213     @CsvSource({
214             "S001, 2024-12-20 18:00, Sala1, 100",
215             "S002, 2024-12-21 20:30, Sala2, 150",
216             "S003, 2024-12-22 15:00, SalaVIP, 50",
217             "S004, 2024-12-23 21:00, Sala3, 200"
218     })
219     void testTworzenieSeansowZRoznymiDanymi(String id, String data,
        String sala, int miejsca) {
220         // Jeśli: Dane seansu z parametrów CSV
221 
222         // Gdy: Tworzymy seans z tymi danymi
223         Seans testSeans = new Seans(id, filmTestowy, data, sala,
            miejsca);
224 
225         // Wtedy: Wszystkie pola powinny być poprawnie ustawione
226         assertEquals(id, testSeans.dajId(), "ID powinno być poprawne");
227         assertEquals(data, testSeans.dajDate(), "Data powinna być
            poprawna");
228         assertEquals(miejsca, testSeans.dajWolneMiejsca().length,
            "Liczba miejsc powinna być poprawna");
229         assertNotNull(testSeans.dajFilm(), "Film nie powinien być
            null");
230     }
231 
232     // ========== TESTY PARAMETRYZOWANE - @ValueSource ==========
233 
234     @ParameterizedTest
235     @Order(11)
236     @DisplayName("Test seansów z różną liczbą miejsc - @ValueSource")
237     @ValueSource(ints = { 20, 50, 100, 150, 200, 300 })
238     void testSeanseZRoznaLiczbaMiejsc(int liczbaMiejsc) {
239         // Jeśli: Różne liczby miejsc w sali
240 
241         // Gdy: Tworzymy seans z określoną liczbą miejsc
242         Seans testSeans = new Seans("ST", filmTestowy, "2024-12-25
            12:00", "TestSala", liczbaMiejsc);
243 
244         // Wtedy: Liczba wolnych miejsc powinna być poprawna
245         assertEquals(liczbaMiejsc, testSeans.dajWolneMiejsca().length,
246                 "Liczba wolnych miejsc powinna być równa " +
                    liczbaMiejsc);
```

```
247     }
248
249     @ParameterizedTest
250     @Order(12)
251     @DisplayName("Test rezerwacji różnych numerów miejsc -
            @ValueSource")
252     @ValueSource(ints = { 1, 25, 50, 75, 100 })
253     void testRezerwacjaRoznychMiejsc(int nrMiejsca) {
254         // Jeśli: Różne numery miejsc do rezerwacji
255
256         // Gdy: Rezerwujemy określone miejsce
257         boolean wynik = seans.zarezerwujMiejsce(nrMiejsca);
258
259         // Wtedy: Rezerwacja powinna się udać
260         assertTrue(wynik, "Rezerwacja miejsca " + nrMiejsca + " powinna
                się udać");
261         assertEquals(99, seans.dajWolneMiejsca().length, "Powinno być
                99 wolnych miejsc");
262     }
263
264     // ========== TESTY WARTOŚCI BRZEGOWYCH ==========
265
266     @Test
267     @Order(13)
268     @DisplayName("Test rezerwacji pierwszego i ostatniego miejsca")
269     void testRezerwacjaMiejscBrzegowych() {
270         // Jeśli: Seans ma 100 miejsc
271
272         // Gdy: Rezerwujemy pierwsze i ostatnie miejsce
273         boolean pierwszeUdane = seans.zarezerwujMiejsce(1);
274         boolean ostatnieUdane = seans.zarezerwujMiejsce(100);
275
276         // Wtedy: Obie rezerwacje powinny się udać
277         assertTrue(pierwszeUdane, "Rezerwacja miejsca 1 powinna się
                udać");
278         assertTrue(ostatnieUdane, "Rezerwacja miejsca 100 powinna się
                udać");
279         assertEquals(98, seans.dajWolneMiejsca().length, "Powinno być
                98 wolnych miejsc");
280     }
281
282     @Test
283     @Order(14)
284     @DisplayName("Test seansu z minimalną liczbą miejsc")
285     void testSeanseMinimalnaLiczbaMiejsc() {
286         // Jeśli: Seans ma tylko 1 miejsce
287
288         // Gdy: Tworzymy seans z 1 miejscem
289         Seans maly = new Seans("SM", filmTestowy, "2024-12-25 12:00",
                "MicroSala", 1);
290
291         // Wtedy: Powinno być 1 wolne miejsce
292         assertEquals(1, maly.dajWolneMiejsca().length, "Powinno być 1
                wolne miejsce");
293
294         // I rezerwacja powinna się udać
295         assertTrue(maly.zarezerwujMiejsce(1), "Rezerwacja jedynego
                miejsca powinna się udać");
```

```
296             assertEquals(0, maly.dajWolneMiejsca().length, "Powinno być 0
                    wolnych miejsc po rezerwacji");
297         }
298 
299         // ========== TESTY NIEZMIENNICZOŚCI ==========
300 
301         @Test
302         @Order(15)
303         @DisplayName("Test niezmienniczości danych seansu")
304         void testNiezmiennoscDanych() {
305             // Jeśli: Seans został utworzony z określonymi danymi
306             String originalId = seans.dajId();
307             String originalData = seans.dajDate();
308             IFilm originalFilm = seans.dajFilm();
309 
310             // Gdy: Pobieramy dane wielokrotnie
311             String id2 = seans.dajId();
312             String data2 = seans.dajDate();
313             IFilm film2 = seans.dajFilm();
314 
315             // Wtedy: Dane powinny pozostać niezmienione
316             assertSame(originalId, id2, "ID nie powinno się zmienić");
317             assertSame(originalData, data2, "Data nie powinna się zmienić");
318             assertSame(originalFilm, film2, "Film nie powinien się
                    zmienić");
319         }
320 }
```

2.2.2 model.TestDAOSeansy

Plik: src/test/java/model/TestDAOSeansy.java

```java
package model;

import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Testy jednostkowe dla klasy DAO - operacje związane z seansami.
 * Testuje dodawanie, wyszukiwanie i usuwanie seansów.
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Warstwa: Encja (model)
 * Zadanie: 1 (testy bez mockowania)
 */
@DisplayName("Testy klasy DAO - operacje seansów")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@Tag("encja")
@Tag("repertuar")
class TestDAOSeansy {

    private DAO dao;

    @BeforeAll
    static void setUpBeforeClass() {
        // Przygotowanie przed wszystkimi testami
        System.out.println("Rozpoczęcie testów DAO - operacje seansów");
    }

    @BeforeEach
    void setUp() {
        // Jeśli: Utworzenie nowego DAO przed każdym testem
        dao = new DAO();
    }

    @AfterEach
    void tearDown() {
        // Sprzątanie po każdym teście
        dao = null;
    }

    @AfterAll
    static void tearDownAfterClass() {
        // Sprzątanie po wszystkich testach
        System.out.println("Zakończenie testów DAO - operacje seansów");
    }

    // ========== TESTY DODAWANIA SEANSÓW ==========

    @Test
    @Order(1)
    @DisplayName("Test dodawania seansu i generowania ID")
```

```
56      void testDodajSeansGenerujeId() {
57          // Jeśli: Dane seansu do zapisania (format:
                idFilmu;data;sala;miejsca)
58          String daneSeansu = "F1;2024-12-20 18:00;Sala1;100";
59
60          // Gdy: Dodajemy seans do DAO
61          String id = dao.dodajSeans(daneSeansu);
62
63          // Wtedy: ID powinno być wygenerowane
64          assertNotNull(id, "ID nie powinno być null");
65          assertTrue(id.startsWith("S"), "ID powinno zaczynać się od S");
66      }
67
68      @Test
69      @Order(2)
70      @DisplayName("Test że kolejne seanse mają unikalne ID")
71      void testDodajSeanseUnikalneldId() {
72          // Jeśli: Dane kilku seansów
73          String seans1 = "F1;2024-12-20 18:00;Sala1;100";
74          String seans2 = "F1;2024-12-20 21:00;Sala2;80";
75          String seans3 = "F2;2024-12-21 19:00;Sala1;100";
76
77          // Gdy: Dodajemy seanse kolejno
78          String id1 = dao.dodajSeans(seans1);
79          String id2 = dao.dodajSeans(seans2);
80          String id3 = dao.dodajSeans(seans3);
81
82          // Wtedy: Wszystkie ID powinny być unikalne
83          assertNotEquals(id1, id2, "ID pierwszego i drugiego seansu
                powinny być różne");
84          assertNotEquals(id2, id3, "ID drugiego i trzeciego seansu
                powinny być różne");
85          assertNotEquals(id1, id3, "ID pierwszego i trzeciego seansu
                powinny być różne");
86      }
87
88      // ========== TESTY WYSZUKIWANIA SEANSÓW ==========
89
90      @Test
91      @Order(3)
92      @DisplayName("Test znajdowania seansu po ID")
93      void testZnajdzSeans() {
94          // Jeśli: Seans został dodany do DAO
95          String daneSeansu = "F1;2024-12-20 18:00;Sala1;100";
96          String id = dao.dodajSeans(daneSeansu);
97
98          // Gdy: Szukamy seansu po ID
99          String znalezionySeans = dao.znajdzSeans(id);
100
101         // Wtedy: Powinniśmy znaleźć ten sam seans
102         assertNotNull(znalezionySeans, "Seans powinien zostać
                znaleziony");
103         assertEquals(daneSeansu, znalezionySeans, "Dane seansu powinny
                być identyczne");
104     }
105
106     @Test
107     @Order(4)
```

```
108     @DisplayName("Test szukania nieistniejącego seansu")
109     void testZnajdzSeansNieistniejacy() {
110         // Jeśli: DAO jest puste lub seans nie istnieje
111 
112         // Gdy: Szukamy nieistniejącego seansu
113         String znalezionySeans = dao.znajdzSeans("S999");
114 
115         // Wtedy: Powinniśmy otrzymać null
116         assertNull(znalezionySeans, "Nieistniejący seans powinien
                zwrócić null");
117     }
118 
119     @Test
120     @Order(5)
121     @DisplayName("Test znajdowania seansów dla filmu")
122     void testZnajdzSeansyFilmu() {
123         // Jeśli: Dodano kilka seansów dla tego samego filmu
124         dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
125         dao.dodajSeans("F1;2024-12-20 21:00;Sala2;80");
126         dao.dodajSeans("F2;2024-12-21 19:00;Sala1;100");
127 
128         // Gdy: Szukamy seansów dla filmu F1
129         String[] seansyF1 = dao.znajdzSeansyFilmu("F1");
130 
131         // Wtedy: Powinny być 2 seanse dla filmu F1
132         assertNotNull(seansyF1, "Tablica seansów nie powinna być null");
133         assertEquals(2, seansyF1.length, "Powinny być 2 seanse dla
                filmu F1");
134     }
135 
136     @Test
137     @Order(6)
138     @DisplayName("Test znajdowania seansów dla filmu który nie ma
            seansów")
139     void testZnajdzSeansyFilmuBezSeansow() {
140         // Jeśli: Dodano seanse dla filmu F1, ale nie dla F99
141         dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
142 
143         // Gdy: Szukamy seansów dla filmu F99
144         String[] seansyF99 = dao.znajdzSeansyFilmu("F99");
145 
146         // Wtedy: Tablica powinna być pusta
147         assertNotNull(seansyF99, "Tablica nie powinna być null");
148         assertEquals(0, seansyF99.length, "Tablica powinna być pusta
                dla filmu bez seansów");
149     }
150 
151     @Test
152     @Order(7)
153     @DisplayName("Test że znajdzSeansyFilmu zwraca poprawne ID seansów")
154     void testZnajdzSeansyFilmuPoprawneId() {
155         // Jeśli: Dodajemy seanse i zapisujemy ich ID
156         String id1 = dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
157         String id2 = dao.dodajSeans("F1;2024-12-20 21:00;Sala2;80");
158         dao.dodajSeans("F2;2024-12-21 19:00;Sala1;100"); // inny film
159 
160         // Gdy: Szukamy seansów dla filmu F1
161         String[] seansyF1 = dao.znajdzSeansyFilmu("F1");
```

```
162
163         // Wtedy: Zwrócone ID powinny odpowiadać dodanym seansom
164         assertEquals(2, seansyF1.length, "Powinny być 2 seanse");
165
166         // Sprawdzamy czy oba ID są w tablicy
167         boolean zawieraId1 = false;
168         boolean zawieraId2 = false;
169         for (String id : seansyF1) {
170             if (id.equals(id1))
171                 zawieraId1 = true;
172             if (id.equals(id2))
173                 zawieraId2 = true;
174         }
175         assertTrue(zawieraId1, "Tablica powinna zawierać ID pierwszego
                seansu");
176         assertTrue(zawieraId2, "Tablica powinna zawierać ID drugiego
                seansu");
177     }
178
179     // ========== TESTY USUWANIA SEANSÓW ==========
180
181     @Test
182     @Order(8)
183     @DisplayName("Test usuwania seansu")
184     void testUsunSeans() {
185         // Jeśli: Seans został dodany
186         String id = dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
187         assertNotNull(dao.znajdzSeans(id), "Seans powinien istnieć
                przed usunięciem");
188
189         // Gdy: Usuwamy seans
190         dao.usunSeans(id);
191
192         // Wtedy: Seans nie powinien być już dostępny
193         assertNull(dao.znajdzSeans(id), "Seans nie powinien istnieć po
                usunięciu");
194     }
195
196     @Test
197     @Order(9)
198     @DisplayName("Test że usunięcie seansu nie wpływa na inne seanse")
199     void testUsunSeansNieWplywaNaInne() {
200         // Jeśli: Dodano kilka seansów
201         String id1 = dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
202         String id2 = dao.dodajSeans("F1;2024-12-20 21:00;Sala2;80");
203
204         // Gdy: Usuwamy pierwszy seans
205         dao.usunSeans(id1);
206
207         // Wtedy: Drugi seans powinien nadal istnieć
208         assertNull(dao.znajdzSeans(id1), "Usunięty seans nie powinien
                istnieć");
209         assertNotNull(dao.znajdzSeans(id2), "Inny seans powinien nadal
                istnieć");
210     }
211
212     // ========== TESTY PARAMETRYZOWANE - @CsvSource ==========
213
```

```
214     @ParameterizedTest
215     @Order(10)
216     @DisplayName("Test dodawania seansów z różnymi danymi - @CsvSource")
217     @CsvSource({
218             "F1, 2024-12-20 18:00, Sala1, 100",
219             "F2, 2024-12-21 20:30, Sala2, 150",
220             "F3, 2024-12-22 15:00, SalaVIP, 50",
221             "F4, 2024-12-23 21:00, Sala3, 200"
222     })
223     void testDodajSeansParametryzowany(String idFilmu, String data,
            String sala, int miejsca) {
224         // Jeśli: Dane seansu z parametrów
225         String daneSeansu = idFilmu + ";" + data + ";" + sala + ";" +
                miejsca;
226 
227         // Gdy: Dodajemy seans
228         String id = dao.dodajSeans(daneSeansu);
229 
230         // Wtedy: Seans powinien być dodany i możliwy do znalezienia
231         assertNotNull(id, "ID nie powinno być null");
232         String znaleziony = dao.znajdzSeans(id);
233         assertNotNull(znaleziony, "Seans powinien być znaleziony");
234         assertEquals(daneSeansu, znaleziony, "Dane seansu powinny być
                identyczne");
235     }
236 
237     @ParameterizedTest
238     @Order(11)
239     @DisplayName("Test znajdowania seansów dla różnych filmów -
            @CsvSource")
240     @CsvSource({
241             "F1, 3",
242             "F2, 2",
243             "F3, 1"
244     })
245     void testZnajdzSeansyDlaRoznychFilmow(String idFilmu, int
            oczekiwanaLiczba) {
246         // Jeśli: Dodajemy seanse dla różnych filmów
247         dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
248         dao.dodajSeans("F1;2024-12-20 21:00;Sala2;80");
249         dao.dodajSeans("F1;2024-12-21 18:00;Sala1;100");
250         dao.dodajSeans("F2;2024-12-21 19:00;Sala1;100");
251         dao.dodajSeans("F2;2024-12-21 21:00;Sala2;80");
252         dao.dodajSeans("F3;2024-12-22 20:00;Sala1;100");
253 
254         // Gdy: Szukamy seansów dla określonego filmu
255         String[] seanse = dao.znajdzSeansyFilmu(idFilmu);
256 
257         // Wtedy: Liczba seansów powinna być zgodna z oczekiwaną
258         assertEquals(oczekiwanaLiczba, seanse.length,
259                 "Film " + idFilmu + " powinien mieć " +
                        oczekiwanaLiczba + " seansów");
260     }
261 
262     // ========== TESTY PARAMETRYZOWANE - @ValueSource ==========
263 
264     @ParameterizedTest
265     @Order(12)
```

```
266     @DisplayName("Test dodawania seansów z różnymi ID filmów -
            @ValueSource")
267     @ValueSource(strings = { "F1", "F10", "F100", "F999", "FILM001" })
268     void testDodajSeansRozneIdFilmow(String idFilmu) {
269         // Jeśli: Różne ID filmów
270         String daneSeansu = idFilmu + ";2024-12-25 18:00;Sala1;100";
271 
272         // Gdy: Dodajemy seans
273         String id = dao.dodajSeans(daneSeansu);
274 
275         // Wtedy: Seans powinien być dodany
276         assertNotNull(id, "ID seansu nie powinno być null");
277 
278         // I powinien być znajdowany przez znajdzSeansyFilmu
279         String[] seanse = dao.znajdzSeansyFilmu(idFilmu);
280         assertEquals(1, seanse.length, "Powinien być 1 seans dla filmu
                " + idFilmu);
281     }
282 
283     @ParameterizedTest
284     @Order(13)
285     @DisplayName("Test dodawania seansów z różną liczbą miejsc -
            @ValueSource")
286     @ValueSource(ints = { 20, 50, 100, 150, 200, 500 })
287     void testDodajSeansRoznaLiczbaMiejsc(int liczbaMiejsc) {
288         // Jeśli: Różne liczby miejsc
289         String daneSeansu = "F1;2024-12-25 18:00;Sala1;" + liczbaMiejsc;
290 
291         // Gdy: Dodajemy seans
292         String id = dao.dodajSeans(daneSeansu);
293 
294         // Wtedy: Seans powinien być zapisany z poprawną liczbą miejsc
295         String znaleziony = dao.znajdzSeans(id);
296         assertNotNull(znaleziony, "Seans powinien zostać znaleziony");
297         assertTrue(znaleziony.contains(String.valueOf(liczbaMiejsc)),
298                 "Dane seansu powinny zawierać liczbę miejsc");
299     }
300 
301     // ========== TESTY INTEGRALNOŚCI DANYCH ==========
302 
303     @Test
304     @Order(14)
305     @DisplayName("Test że dane seansu są przechowywane bez modyfikacji")
306     void testDaneSeansaBezModyfikacji() {
307         // Jeśli: Oryginalne dane seansu
308         String oryginalneDane = "F1;2024-12-20 18:00;Sala Główna;120";
309 
310         // Gdy: Dodajemy i pobieramy seans
311         String id = dao.dodajSeans(oryginalneDane);
312         String pobraneDane = dao.znajdzSeans(id);
313 
314         // Wtedy: Dane powinny być identyczne
315         assertEquals(oryginalneDane, pobraneDane, "Dane nie powinny być
                modyfikowane");
316     }
317 
318     @Test
319     @Order(15)
```

```
320     @DisplayName("Test wielokrotnego wyszukiwania tego samego seansu")
321     void testWielokrotneWyszukiwanie() {
322         // Jeśli: Seans został dodany
323         String daneSeansu = "F1;2024-12-20 18:00;Sala1;100";
324         String id = dao.dodajSeans(daneSeansu);
325 
326         // Gdy: Szukamy seansu wielokrotnie
327         String wynik1 = dao.znajdzSeans(id);
328         String wynik2 = dao.znajdzSeans(id);
329         String wynik3 = dao.znajdzSeans(id);
330 
331         // Wtedy: Każde wyszukiwanie powinno zwrócić te same dane
332         assertEquals(wynik1, wynik2, "Wyniki powinny być identyczne");
333         assertEquals(wynik2, wynik3, "Wyniki powinny być identyczne");
334         assertEquals(daneSeansu, wynik1, "Dane powinny odpowiadać
                oryginalnym");
335     }
336 }
```

2.2.3 model.TestModelPobierzRepertuar

Plik: src/test/java/model/TestModelPobierzRepertuar.java

```java
package model;

import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Testy jednostkowe dla klasy Model - operacja pobierania repertuaru.
 * Testuje metodę Model.pobierzRepertuar() bez mockowania (z prawdziwym DAO).
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Warstwa: Encja (model)
 * Zadanie: 1 (testy bez mockowania)
 */
@DisplayName("Testy klasy Model - pobieranie repertuaru")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@Tag("encja")
@Tag("repertuar")
class TestModelPobierzRepertuar {

    private Model model;
    private DAO dao;
    private Oferta oferta;

    @BeforeAll
    static void setUpBeforeClass() {
        // Przygotowanie przed wszystkimi testami
        System.out.println("Rozpoczęcie testów Model - pobieranie repertuaru");
    }

    @BeforeEach
    void setUp() {
        // Jeśli: Utworzenie systemu przed każdym testem
        dao = new DAO();
        oferta = new Oferta(dao);
        model = new Model(oferta, dao);
    }

    @AfterEach
    void tearDown() {
        // Sprzątanie po każdym teście
        model = null;
        oferta = null;
        dao = null;
    }

    @AfterAll
    static void tearDownAfterClass() {
        // Sprzątanie po wszystkich testach
```

```
54          System.out.println("Zakończenie testów Model - pobieranie
                repertuaru");
55      }
56
57      // ========== TESTY POBIERANIA REPERTUARU Z SEANSAMI ==========
58
59      @Test
60      @Order(1)
61      @DisplayName("Test pobierania repertuaru gdy są seanse dla filmu")
62      void testPobierzRepertuarZSeansami() {
63          // Jeśli: Dodano seanse dla filmu F1
64          dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
65          dao.dodajSeans("F1;2024-12-20 21:00;Sala2;80");
66
67          // Gdy: Pobieramy repertuar dla filmu F1
68          String repertuar = model.pobierzRepertuar("F1");
69
70          // Wtedy: Repertuar powinien zawierać informacje o seansach
71          assertNotNull(repertuar, "Repertuar nie powinien być null");
72          assertFalse(repertuar.isEmpty(), "Repertuar nie powinien być
                pusty");
73          assertTrue(repertuar.contains("Repertuar"), "Repertuar powinien
                zawierać nagłówek");
74          assertTrue(repertuar.contains("Seans"), "Repertuar powinien
                zawierać informacje o seansach");
75      }
76
77      @Test
78      @Order(2)
79      @DisplayName("Test pobierania repertuaru gdy brak seansów dla
            filmu")
80      void testPobierzRepertuarBezSeansow() {
81          // Jeśli: Nie dodano żadnych seansów dla filmu F99
82
83          // Gdy: Pobieramy repertuar dla filmu F99
84          String repertuar = model.pobierzRepertuar("F99");
85
86          // Wtedy: Powinniśmy otrzymać komunikat o braku seansów
87          assertNotNull(repertuar, "Repertuar nie powinien być null");
88          assertTrue(repertuar.contains("Brak seansow"),
89                  "Repertuar powinien zawierać komunikat o braku
                        seansów");
90          assertTrue(repertuar.contains("F99"),
91                  "Komunikat powinien zawierać ID filmu");
92      }
93
94      @Test
95      @Order(3)
96      @DisplayName("Test że repertuar zawiera dane wszystkich seansów
            filmu")
97      void testRepertuarZawieraDaneSeansow() {
98          // Jeśli: Dodano seanse z konkretnymi danymi
99          dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
100         dao.dodajSeans("F1;2024-12-20 21:00;Sala2;80");
101
102         // Gdy: Pobieramy repertuar
103         String repertuar = model.pobierzRepertuar("F1");
104
```

```
105         // Wtedy: Repertuar powinien zawierać dane wszystkich seansów
106         assertTrue(repertuar.contains("2024-12-20 18:00"),
107                 "Repertuar powinien zawierać datę pierwszego seansu");
108         assertTrue(repertuar.contains("2024-12-20 21:00"),
109                 "Repertuar powinien zawierać datę drugiego seansu");
110         assertTrue(repertuar.contains("Sala1"),
111                 "Repertuar powinien zawierać salę pierwszego seansu");
112         assertTrue(repertuar.contains("Sala2"),
113                 "Repertuar powinien zawierać salę drugiego seansu");
114     }
115 
116     @Test
117     @Order(4)
118     @DisplayName("Test formatu repertuaru")
119     void testFormatRepertuaru() {
120         // Jeśli: Dodano seans
121         dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
122 
123         // Gdy: Pobieramy repertuar
124         String repertuar = model.pobierzRepertuar("F1");
125 
126         // Wtedy: Repertuar powinien mieć odpowiedni format
127         assertTrue(repertuar.startsWith("Repertuar dla filmu"),
128                 "Repertuar powinien zaczynać się od nagłówka");
129         assertTrue(repertuar.contains("F1"),
130                 "Repertuar powinien zawierać ID filmu");
131         assertTrue(repertuar.contains("- Seans:"),
132                 "Repertuar powinien formatować seanse z myślnikiem");
133     }
134 
135     @Test
136     @Order(5)
137     @DisplayName("Test że repertuar nie zawiera seansów innych filmów")
138     void testRepertuarNieZawieraInnychFilmow() {
139         // Jeśli: Dodano seanse dla różnych filmów
140         dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
141         dao.dodajSeans("F2;2024-12-21 19:00;Sala2;80");
142         dao.dodajSeans("F3;2024-12-22 20:00;Sala3;120");
143 
144         // Gdy: Pobieramy repertuar dla filmu F1
145         String repertuar = model.pobierzRepertuar("F1");
146 
147         // Wtedy: Repertuar powinien zawierać tylko seanse F1
148         assertTrue(repertuar.contains("2024-12-20 18:00"),
149                 "Repertuar powinien zawierać seans F1");
150         assertFalse(repertuar.contains("2024-12-21 19:00"),
151                 "Repertuar nie powinien zawierać seansu F2");
152         assertFalse(repertuar.contains("2024-12-22 20:00"),
153                 "Repertuar nie powinien zawierać seansu F3");
154     }
155 
156     // ========== TESTY WIELU SEANSÓW ==========
157 
158     @Test
159     @Order(6)
160     @DisplayName("Test pobierania repertuaru z wieloma seansami")
161     void testPobierzRepertuarWieleSeansow() {
162         // Jeśli: Dodano 5 seansów dla tego samego filmu
```

```
163         dao.dodajSeans("F1;2024-12-20 10:00;Sala1;100");
164         dao.dodajSeans("F1;2024-12-20 13:00;Sala1;100");
165         dao.dodajSeans("F1;2024-12-20 16:00;Sala2;80");
166         dao.dodajSeans("F1;2024-12-20 19:00;Sala1;100");
167         dao.dodajSeans("F1;2024-12-20 22:00;Sala2;80");
168
169         // Gdy: Pobieramy repertuar
170         String repertuar = model.pobierzRepertuar("F1");
171
172         // Wtedy: Wszystkie seanse powinny być wymienione
173         assertNotNull(repertuar);
174         // Liczymy wystąpienia "Seans" w repertuarze
175         int liczbaSeansow = repertuar.split("Seans").length - 1;
176         assertEquals(5, liczbaSeansow, "Powinno być 5 seansów w
                repertuarze");
177     }
178
179     @Test
180     @Order(7)
181     @DisplayName("Test pobierania repertuaru z pojedynczym seansem")
182     void testPobierzRepertuarJedenSeans() {
183         // Jeśli: Dodano tylko jeden seans
184         dao.dodajSeans("F1;2024-12-25 20:00;SalaGłówna;150");
185
186         // Gdy: Pobieramy repertuar
187         String repertuar = model.pobierzRepertuar("F1");
188
189         // Wtedy: Repertuar powinien zawierać dokładnie jeden seans
190         assertNotNull(repertuar);
191         assertTrue(repertuar.contains("Seans"), "Repertuar powinien
                zawierać seans");
192         assertTrue(repertuar.contains("2024-12-25 20:00"),
193                 "Repertuar powinien zawierać datę seansu");
194     }
195
196     // ========== TESTY PARAMETRYZOWANE - @CsvSource ==========
197
198     @ParameterizedTest
199     @Order(8)
200     @DisplayName("Test pobierania repertuaru dla różnych filmów -
            @CsvSource")
201     @CsvSource({
202             "F1, 2024-12-20 18:00, Sala1",
203             "F2, 2024-12-21 20:30, Sala2",
204             "F3, 2024-12-22 15:00, SalaVIP"
205     })
206     void testPobierzRepertuarParametryzowany(String idFilmu, String
            data, String sala) {
207         // Jeśli: Dodano seans z parametrów
208         dao.dodajSeans(idFilmu + ";" + data + ";" + sala + ";100");
209
210         // Gdy: Pobieramy repertuar dla danego filmu
211         String repertuar = model.pobierzRepertuar(idFilmu);
212
213         // Wtedy: Repertuar powinien zawierać dane seansu
214         assertNotNull(repertuar, "Repertuar nie powinien być null");
215         assertTrue(repertuar.contains(data), "Repertuar powinien
                zawierać datę");
```

```
216         assertTrue(repertuar.contains(sala), "Repertuar powinien
                zawierać salę");
217         assertFalse(repertuar.contains("Brak seansow"),
218                 "Nie powinno być komunikatu o braku seansów");
219     }
220 
221     @ParameterizedTest
222     @Order(9)
223     @DisplayName("Test liczby seansów w repertuarze - @CsvSource")
224     @CsvSource({
225             "F1, 1",
226             "F2, 2",
227             "F3, 3"
228     })
229     void testLiczbaSeansowWRepertuarze(String idFilmu, int
            oczekiwanaLiczba) {
230         // Jeśli: Dodajemy określoną liczbę seansów dla każdego filmu
231         for (int i = 0; i < oczekiwanaLiczba; i++) {
232             dao.dodajSeans(idFilmu + ";2024-12-20 " + (10 + i) +
                    ":00;Sala1;100");
233         }
234         // Dodajemy też seanse innych filmów jako szum
235         dao.dodajSeans("FX;2024-12-25 12:00;Sala1;100");
236 
237         // Gdy: Pobieramy repertuar
238         String repertuar = model.pobierzRepertuar(idFilmu);
239 
240         // Wtedy: Liczba seansów powinna być zgodna z oczekiwaną
241         int liczbaSeansow = repertuar.split("Seans").length - 1;
242         assertEquals(oczekiwanaLiczba, liczbaSeansow,
243                 "Repertuar dla " + idFilmu + " powinien mieć " +
                        oczekiwanaLiczba + " seansów");
244     }
245 
246     // ========== TESTY PARAMETRYZOWANE - @ValueSource ==========
247 
248     @ParameterizedTest
249     @Order(10)
250     @DisplayName("Test pobierania repertuaru dla filmów bez seansów -
            @ValueSource")
251     @ValueSource(strings = { "F99", "F100", "NIEISTNIEJACY", "XYZ123" })
252     void testPobierzRepertuarBrakSeansow(String idFilmu) {
253         // Jeśli: Film nie ma żadnych seansów
254 
255         // Gdy: Pobieramy repertuar
256         String repertuar = model.pobierzRepertuar(idFilmu);
257 
258         // Wtedy: Powinniśmy otrzymać komunikat o braku seansów
259         assertNotNull(repertuar);
260         assertTrue(repertuar.contains("Brak seansow"),
261                 "Powinien być komunikat o braku seansów dla " +
                        idFilmu);
262         assertTrue(repertuar.contains(idFilmu),
263                 "Komunikat powinien zawierać ID filmu");
264     }
265 
266     @ParameterizedTest
267     @Order(11)
```

```
268     @DisplayName("Test repertuaru z różnymi salami - @ValueSource")
269     @ValueSource(strings = { "Sala1", "Sala2", "SalaVIP", "SalaIMAX",
            "Sala3D" })
270     void testRepertuarRozneSale(String sala) {
271         // Jeśli: Dodano seans w określonej sali
272         dao.dodajSeans("F1;2024-12-25 18:00;" + sala + ";100");
273 
274         // Gdy: Pobieramy repertuar
275         String repertuar = model.pobierzRepertuar("F1");
276 
277         // Wtedy: Repertuar powinien zawierać nazwę sali
278         assertTrue(repertuar.contains(sala),
279                 "Repertuar powinien zawierać salę: " + sala);
280     }
281 
282     // ========== TESTY EDGE CASES ==========
283 
284     @Test
285     @Order(12)
286     @DisplayName("Test pobierania repertuaru z pustym ID filmu")
287     void testPobierzRepertuarPusteId() {
288         // Jeśli: ID filmu jest puste
289 
290         // Gdy: Pobieramy repertuar z pustym ID
291         String repertuar = model.pobierzRepertuar("");
292 
293         // Wtedy: Powinniśmy otrzymać komunikat o braku seansów
294         assertNotNull(repertuar);
295         assertTrue(repertuar.contains("Brak seansow"),
296                 "Powinien być komunikat o braku seansów dla pustego
                        ID");
297     }
298 
299     @Test
300     @Order(13)
301     @DisplayName("Test że pobieranie repertuaru nie modyfikuje danych")
302     void testPobierzRepertuarNieModyfikujeDanych() {
303         // Jeśli: Dodano seans
304         String daneSeansu = "F1;2024-12-20 18:00;Sala1;100";
305         String seansId = dao.dodajSeans(daneSeansu);
306 
307         // Gdy: Pobieramy repertuar wielokrotnie
308         model.pobierzRepertuar("F1");
309         model.pobierzRepertuar("F1");
310         model.pobierzRepertuar("F1");
311 
312         // Wtedy: Dane seansu powinny pozostać niezmienione
313         String danePo = dao.znajdzSeans(seansId);
314         assertEquals(daneSeansu, danePo, "Dane seansu nie powinny być
                modyfikowane");
315     }
316 
317     @Test
318     @Order(14)
319     @DisplayName("Test spójności wielokrotnego pobierania repertuaru")
320     void testSpojnoscWielokrotnePobieranie() {
321         // Jeśli: Dodano seanse
322         dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
```

```
323         dao.dodajSeans("F1;2024-12-20 21:00;Sala2;80");
324 
325         // Gdy: Pobieramy repertuar wielokrotnie
326         String repertuar1 = model.pobierzRepertuar("F1");
327         String repertuar2 = model.pobierzRepertuar("F1");
328         String repertuar3 = model.pobierzRepertuar("F1");
329 
330         // Wtedy: Każde pobranie powinno zwrócić identyczny wynik
331         assertEquals(repertuar1, repertuar2, "Repertuar powinien być
               spójny");
332         assertEquals(repertuar2, repertuar3, "Repertuar powinien być
               spójny");
333     }
334 }
```

2.2.4 controller.TestClientControllerPrzegladanieRepertuaru

Plik: src/test/java/controller/TestClientControllerPrzegladanieRepertuaru.java

```java
package controller;

import model.*;
import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Testy jednostkowe dla klasy ClientController - przeglądanie repertuaru.
 * Testuje metodę ClientController.przegladajRepertuar() bez mockowania.
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Warstwa: Kontroli (controller)
 * Zadanie: 1 (testy bez mockowania)
 */
@DisplayName("Testy klasy ClientController - przeglądanie repertuaru")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@Tag("kontroler")
@Tag("repertuar")
class TestClientControllerPrzegladanieRepertuaru {

    private ClientController clientController;
    private Model model;
    private DAO dao;
    private Oferta oferta;

    @BeforeAll
    static void setUpBeforeClass() {
        // Przygotowanie przed wszystkimi testami
        System.out.println("Rozpoczęcie testów ClientController - przeglądanie repertuaru");
    }

    @BeforeEach
    void setUp() {
        // Jeśli: Utworzenie pełnego systemu przed każdym testem
        dao = new DAO();
        oferta = new Oferta(dao);
        model = new Model(oferta, dao);
        clientController = new ClientController(model);
    }

    @AfterEach
    void tearDown() {
        // Sprzątanie po każdym teście
        clientController = null;
        model = null;
        oferta = null;
        dao = null;
    }
```

```
54
55      @AfterAll
56      static void tearDownAfterClass() {
57          // Sprzątanie po wszystkich testach
58          System.out.println("Zakończenie testów ClientController -
                przeglądanie repertuaru");
59      }
60
61      // ========== TESTY PRZEGLĄDANIA REPERTUARU ==========
62
63      @Test
64      @Order(1)
65      @DisplayName("Test przeglądania repertuaru przez ClientController")
66      void testPrzegladajRepertuar() {
67          // Jeśli: Dodano seanse dla filmu
68          dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
69          dao.dodajSeans("F1;2024-12-20 21:00;Sala2;80");
70
71          // Gdy: Klient przegląda repertuar
72          String repertuar = clientController.przegladajRepertuar("F1");
73
74          // Wtedy: Powinien otrzymać informacje o seansach
75          assertNotNull(repertuar, "Repertuar nie powinien być null");
76          assertFalse(repertuar.isEmpty(), "Repertuar nie powinien być
                pusty");
77          assertTrue(repertuar.contains("Repertuar"), "Repertuar powinien
                zawierać nagłówek");
78      }
79
80      @Test
81      @Order(2)
82      @DisplayName("Test przeglądania repertuaru bez seansów")
83      void testPrzegladajRepertuarBezSeansow() {
84          // Jeśli: Film nie ma seansów
85
86          // Gdy: Klient przegląda repertuar
87          String repertuar = clientController.przegladajRepertuar("F99");
88
89          // Wtedy: Powinien otrzymać komunikat o braku seansów
90          assertNotNull(repertuar);
91          assertTrue(repertuar.contains("Brak seansow"),
92                  "Powinien być komunikat o braku seansów");
93      }
94
95      @Test
96      @Order(3)
97      @DisplayName("Test że ClientController deleguje do Model")
98      void testDelegacjaDoModel() {
99          // Jeśli: Dodano seans z unikalnymi danymi
100         dao.dodajSeans("F1;2024-12-25 20:00;SalaSpecjalna;200");
101
102         // Gdy: Klient przegląda repertuar
103         String repertuar = clientController.przegladajRepertuar("F1");
104
105         // Wtedy: Wynik powinien zawierać dane z DAO (przeszło przez
                Model)
106         assertTrue(repertuar.contains("SalaSpecjalna"),
107                 "Repertuar powinien zawierać dane z DAO");
```

```
108             assertTrue(repertuar.contains("2024-12-25 20:00"),
109                     "Repertuar powinien zawierać datę z DAO");
110         }
111
112         @Test
113         @Order(4)
114         @DisplayName("Test przeglądania repertuaru z wieloma seansami")
115         void testPrzegladajRepertuarWieleSeansow() {
116             // Jeśli: Dodano wiele seansów
117             dao.dodajSeans("F1;2024-12-20 10:00;Sala1;100");
118             dao.dodajSeans("F1;2024-12-20 13:00;Sala1;100");
119             dao.dodajSeans("F1;2024-12-20 16:00;Sala2;80");
120             dao.dodajSeans("F1;2024-12-20 19:00;Sala1;100");
121
122             // Gdy: Klient przegląda repertuar
123             String repertuar = clientController.przegladajRepertuar("F1");
124
125             // Wtedy: Wszystkie seanse powinny być dostępne
126             assertTrue(repertuar.contains("10:00"), "Repertuar powinien
                zawierać seans o 10:00");
127             assertTrue(repertuar.contains("13:00"), "Repertuar powinien
                zawierać seans o 13:00");
128             assertTrue(repertuar.contains("16:00"), "Repertuar powinien
                zawierać seans o 16:00");
129             assertTrue(repertuar.contains("19:00"), "Repertuar powinien
                zawierać seans o 19:00");
130         }
131
132         @Test
133         @Order(5)
134         @DisplayName("Test że repertuar zawiera tylko seanse wybranego
            filmu")
135         void testRepertuarTylkoWybranyFilm() {
136             // Jeśli: Dodano seanse dla różnych filmów
137             dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
138             dao.dodajSeans("F2;2024-12-21 19:00;Sala2;80");
139             dao.dodajSeans("F3;2024-12-22 20:00;Sala3;120");
140
141             // Gdy: Klient przegląda repertuar dla F1
142             String repertuar = clientController.przegladajRepertuar("F1");
143
144             // Wtedy: Repertuar powinien zawierać tylko seanse F1
145             assertTrue(repertuar.contains("2024-12-20 18:00"),
146                     "Repertuar powinien zawierać seans F1");
147             assertFalse(repertuar.contains("2024-12-21 19:00"),
148                     "Repertuar nie powinien zawierać seansu F2");
149             assertFalse(repertuar.contains("2024-12-22 20:00"),
150                     "Repertuar nie powinien zawierać seansu F3");
151         }
152
153         // ========== TESTY PEŁNEGO PRZEPŁYWU ==========
154
155         @Test
156         @Order(6)
157         @DisplayName("Test pełnego przepływu - od kontrolera do DAO")
158         void testPelnyPrzeplyw() {
159             // Jeśli: Kompletne dane seansu
160             String daneSeansu = "F1;2024-12-25 18:00;SalaGłówna;150";
```

```
161         dao.dodajSeans(daneSeansu);
162
163         // Gdy: Klient przegląda repertuar przez ClientController
164         // (co wywołuje Model -> DAO)
165         String repertuar = clientController.przegladajRepertuar("F1");
166
167         // Wtedy: Cały przepływ powinien zakończyć się sukcesem
168         assertNotNull(repertuar);
169         assertTrue(repertuar.contains("Repertuar dla filmu F1"),
170                 "Repertuar powinien mieć poprawny nagłówek");
171         assertTrue(repertuar.contains("SalaGłówna"),
172                 "Repertuar powinien zawierać dane seansu");
173     }
174
175     @Test
176     @Order(7)
177     @DisplayName("Test wielokrotnego przeglądania tego samego
        repertuaru")
178     void testWielokrotnePrzegladanie() {
179         // Jeśli: Dodano seanse
180         dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
181
182         // Gdy: Klient przegląda repertuar wielokrotnie
183         String repertuar1 = clientController.przegladajRepertuar("F1");
184         String repertuar2 = clientController.przegladajRepertuar("F1");
185         String repertuar3 = clientController.przegladajRepertuar("F1");
186
187         // Wtedy: Każde przeglądanie powinno zwrócić identyczny wynik
188         assertEquals(repertuar1, repertuar2, "Repertuar powinien być
            spójny");
189         assertEquals(repertuar2, repertuar3, "Repertuar powinien być
            spójny");
190     }
191
192     @Test
193     @Order(8)
194     @DisplayName("Test przeglądania repertuaru różnych filmów
        sekwencyjnie")
195     void testPrzegladanieRoznychFilmow() {
196         // Jeśli: Dodano seanse dla różnych filmów
197         dao.dodajSeans("F1;2024-12-20 18:00;Sala1;100");
198         dao.dodajSeans("F2;2024-12-21 19:00;Sala2;80");
199         dao.dodajSeans("F3;2024-12-22 20:00;Sala3;120");
200
201         // Gdy: Klient przegląda repertuary kolejno
202         String repertuarF1 = clientController.przegladajRepertuar("F1");
203         String repertuarF2 = clientController.przegladajRepertuar("F2");
204         String repertuarF3 = clientController.przegladajRepertuar("F3");
205
206         // Wtedy: Każdy repertuar powinien być inny i zawierać właściwe
            dane
207         assertNotEquals(repertuarF1, repertuarF2, "Repertuary powinny
            być różne");
208         assertNotEquals(repertuarF2, repertuarF3, "Repertuary powinny
            być różne");
209         assertTrue(repertuarF1.contains("F1"), "Repertuar F1 powinien
            zawierać ID F1");
210         assertTrue(repertuarF2.contains("F2"), "Repertuar F2 powinien
```

```
            zawierać ID F2");
211         assertTrue(repertuarF3.contains("F3"), "Repertuar F3 powinien
                zawierać ID F3");
212     }
213
214     // ========== TESTY PARAMETRYZOWANE - @CsvSource ==========
215
216     @ParameterizedTest
217     @Order(9)
218     @DisplayName("Test przeglądania repertuaru z różnymi danymi -
            @CsvSource")
219     @CsvSource({
220             "F1, 2024-12-20 18:00, Sala1, 100",
221             "F2, 2024-12-21 20:30, Sala2, 150",
222             "F3, 2024-12-22 15:00, SalaVIP, 50",
223             "F4, 2024-12-23 21:00, IMAX, 200"
224     })
225     void testPrzegladajRepertuarParametryzowany(String idFilmu, String
            data,
226             String sala, int miejsca) {
227         // Jeśli: Dodano seans z parametrów
228         dao.dodajSeans(idFilmu + ";" + data + ";" + sala + ";" +
                miejsca);
229
230         // Gdy: Klient przegląda repertuar
231         String repertuar =
                clientController.przegladajRepertuar(idFilmu);
232
233         // Wtedy: Repertuar powinien zawierać dane seansu
234         assertNotNull(repertuar);
235         assertTrue(repertuar.contains(data),
236                 "Repertuar powinien zawierać datę: " + data);
237         assertTrue(repertuar.contains(sala),
238                 "Repertuar powinien zawierać salę: " + sala);
239     }
240
241     @ParameterizedTest
242     @Order(10)
243     @DisplayName("Test przeglądania repertuaru dla różnych filmów -
            @CsvSource")
244     @CsvSource({
245             "FILM001, 3",
246             "FILM002, 2",
247             "FILM003, 1"
248     })
249     void testLiczbaSeansowWRepertuarze(String idFilmu, int
            liczbaSeansow) {
250         // Jeśli: Dodajemy określoną liczbę seansów
251         for (int i = 0; i < liczbaSeansow; i++) {
252             dao.dodajSeans(idFilmu + ";2024-12-20 " + (10 + i * 3) +
                    ":00;Sala" + (i + 1) + ";100");
253         }
254
255         // Gdy: Klient przegląda repertuar
256         String repertuar =
                clientController.przegladajRepertuar(idFilmu);
257
258         // Wtedy: Liczba seansów powinna być prawidłowa
```

```
259         int licznik = repertuar.split("Seans").length - 1;
260         assertEquals(liczbaSeansow, licznik,
261                 "Powinno być " + liczbaSeansow + " seansów dla " +
                    idFilmu);
262     }
263
264     // ========== TESTY PARAMETRYZOWANE - @ValueSource ==========
265
266     @ParameterizedTest
267     @Order(11)
268     @DisplayName("Test przeglądania repertuaru dla filmów bez seansów -
            @ValueSource")
269     @ValueSource(strings = { "NIEISTNIEJACY", "F999", "XYZ", "TEST123"
            })
270     void testPrzegladajRepertuarBrakSeansow(String idFilmu) {
271         // Jeśli: Film nie ma żadnych seansów
272
273         // Gdy: Klient przegląda repertuar
274         String repertuar =
                clientController.przegladajRepertuar(idFilmu);
275
276         // Wtedy: Powinien otrzymać komunikat o braku seansów
277         assertNotNull(repertuar);
278         assertTrue(repertuar.contains("Brak seansow"),
279                 "Powinien być komunikat o braku seansów dla " +
                    idFilmu);
280     }
281
282     @ParameterizedTest
283     @Order(12)
284     @DisplayName("Test przeglądania repertuaru z różnymi ID filmów -
            @ValueSource")
285     @ValueSource(strings = { "F1", "F10", "FILM", "MOVIE_2024" })
286     void testPrzegladajRepertuarRozneIdFilmow(String idFilmu) {
287         // Jeśli: Dodano seans dla filmu
288         dao.dodajSeans(idFilmu + ";2024-12-25 18:00;Sala1;100");
289
290         // Gdy: Klient przegląda repertuar
291         String repertuar =
                clientController.przegladajRepertuar(idFilmu);
292
293         // Wtedy: Repertuar powinien zawierać dane seansu
294         assertNotNull(repertuar);
295         assertTrue(repertuar.contains("Repertuar dla filmu " + idFilmu),
296                 "Repertuar powinien zawierać nagłówek z ID filmu");
297         assertFalse(repertuar.contains("Brak seansow"),
298                 "Nie powinno być komunikatu o braku seansów");
299     }
300
301     // ========== TESTY EDGE CASES ==========
302
303     @Test
304     @Order(13)
305     @DisplayName("Test przeglądania repertuaru z pustym ID")
306     void testPrzegladajRepertuarPusteId() {
307         // Jeśli: ID filmu jest puste
308
309         // Gdy: Klient przegląda repertuar z pustym ID
```

```
310         String repertuar = clientController.przegladajRepertuar("");
311
312         // Wtedy: Powinien otrzymać komunikat o braku seansów
313         assertNotNull(repertuar);
314         assertTrue(repertuar.contains("Brak seansow"),
315                 "Powinien być komunikat o braku seansów");
316     }
317
318     @Test
319     @Order(14)
320     @DisplayName("Test że ClientController nie modyfikuje danych")
321     void testNieModyfikujeDanych() {
322         // Jeśli: Dodano seans
323         String daneSeansu = "F1;2024-12-20 18:00;Sala1;100";
324         String seansId = dao.dodajSeans(daneSeansu);
325
326         // Gdy: Klient przegląda repertuar wielokrotnie
327         clientController.przegladajRepertuar("F1");
328         clientController.przegladajRepertuar("F1");
329
330         // Wtedy: Dane seansu powinny pozostać niezmienione
331         String danePo = dao.znajdzSeans(seansId);
332         assertEquals(daneSeansu, danePo,
333                 "Dane seansu nie powinny być modyfikowane przez
                    przeglądanie");
334     }
335 }
```

3 Zadanie 2: Testy jednostkowe z mockowaniem (Mockito)

3.1 Przypadek użycia: Dodanie filmu do oferty

W tym zadaniu testujemy klasy z symulacją zależności (z mockowaniem Mockito).

3.1.1 model.TestModelDodawanieFilmuMock

Plik: `src/test/java/model/TestModelDodawanieFilmuMock.java`

```
1  package model;
2
3  import org.junit.jupiter.api.*;
4  import org.junit.jupiter.api.Tag;
5  import org.junit.jupiter.api.extension.ExtendWith;
6  import org.junit.jupiter.params.ParameterizedTest;
7  import org.junit.jupiter.params.provider.CsvSource;
8  import org.junit.jupiter.params.provider.ValueSource;
9  import org.mockito.*;
10 import org.mockito.junit.jupiter.MockitoExtension;
11
12 import static org.junit.jupiter.api.Assertions.*;
13 import static org.mockito.Mockito.*;
14
15 /**
16  * Testy jednostkowe dla klasy Model - operacja dodawania filmu z
        użyciem
17  * mockowania.
18  * Testuje metodę Model.dodajFilm() z symulacją zależności (IDAO).
19  *
20  * Przypadek użycia: Dodanie filmu do oferty
21  * Warstwa: Encja (model)
22  * Zadanie: 2 (testy z mockowaniem)
23  *
24  * Ref. z instrukcji: "Symulować należy te fragmenty kodu (obiekty,
        operacje),
25  * od których zależy testowana operacja"
26  */
27 @DisplayName("Testy klasy Model - dodawanie filmu z mockowaniem")
28 @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
29 @ExtendWith(MockitoExtension.class)
30 @Tag("encja")
31 @Tag("dodawanie")
32 @Tag("mock")
33 class TestModelDodawanieFilmuMock {
34
35     /**
36      * Mock obiektu DAO - symulacja warstwy dostępu do danych.
37      * Ref. z instrukcji: "operacja mock() lub adnotacja @Mock"
38      */
39     @Mock
40     private IDAO mockDao;
41
42     /**
43      * Mock obiektu Oferta (nie używany bezpośrednio w dodajFilm,
44      * ale wymagany przez konstruktor Model)
45      */
46     @Mock
```

```
47      private Oferta mockOferta;
48
49      /**
50       * Testowany obiekt Model z wstrzykniętymi symulacjami.
51       * Ref. z instrukcji: "adnotacja @InjectMocks w Mockito"
52       */
53      @InjectMocks
54      private Model model;
55
56      @BeforeAll
57      static void setUpBeforeClass() {
58          // Przygotowanie przed wszystkimi testami
59          System.out.println("Rozpoczęcie testów Model z mockowaniem");
60      }
61
62      @BeforeEach
63      void setUp() {
64          // Jeśli: Inicjalizacja mocków - wykonywana automatycznie przez
65          // @ExtendWith(MockitoExtension.class)
66          // MockitoAnnotations.openMocks(this) - alternatywa dla
                @ExtendWith
67      }
68
69      @AfterEach
70      void tearDown() {
71          // Sprzątanie po każdym teście - resetowanie mocków
72          reset(mockDao, mockOferta);
73      }
74
75      @AfterAll
76      static void tearDownAfterClass() {
77          // Sprzątanie po wszystkich testach
78          System.out.println("Zakończenie testów Model z mockowaniem");
79      }
80
81      // ========== TESTY Z WHEN().THENRETURN() ==========
82
83      @Test
84      @Order(1)
85      @DisplayName("Test dodawania filmu - sukces z mockowanym DAO")
86      void testDodajFilmSukces() {
87          // Jeśli: Mock DAO zwraca ID filmu po dodaniu
88          // Ref. z instrukcji: "when().thenReturn() w Mockito"
89          String daneFilmu = "F001;Matrix;Cyberpunk
                thriller;136;SciFi;28.5";
90          when(mockDao.dodajFilm(anyString())).thenReturn("F001");
91
92          // Gdy: Wywołujemy dodajFilm na modelu
93          String wynik = model.dodajFilm(daneFilmu);
94
95          // Wtedy: Weryfikujemy wynik i użycie mocka
96          assertNotNull(wynik, "Wynik nie powinien być null");
97          assertTrue(wynik.contains("pomyslnie"), "Wynik powinien
                zawierać 'pomyslnie'");
98          assertTrue(wynik.contains("F001"), "Wynik powinien zawierać ID
                filmu");
99
100         // Ref. z instrukcji: "verify(), times()"
```

```
101         verify(mockDao, times(1)).dodajFilm(anyString());
102     }
103 
104     @Test
105     @Order(2)
106     @DisplayName("Test że DAO.dodajFilm jest wywoływane dokładnie raz")
107     void testDodajFilmWywolanieDaoJednorazowe() {
108         // Jeśli: Przygotowanie mocka
109         String daneFilmu = "F002;Avatar;Fantasy;162;SciFi;35.0";
110         when(mockDao.dodajFilm(anyString())).thenReturn("F002");
111 
112         // Gdy: Wywołujemy dodajFilm
113         model.dodajFilm(daneFilmu);
114 
115         // Wtedy: DAO.dodajFilm powinno być wywołane dokładnie raz
116         // Ref. z instrukcji: "verify(), times()"
117         verify(mockDao, times(1)).dodajFilm(anyString());
118         verify(mockDao, never()).edytujFilm(anyString());
119         verify(mockDao, never()).usunFilm(anyString());
120     }
121 
122     @Test
123     @Order(3)
124     @DisplayName("Test że DAO.dodajWpisDoLogu jest wywoływane po
        dodaniu filmu")
125     void testDodajFilmLogowanie() {
126         // Jeśli: Mock DAO zwraca ID
127         String daneFilmu = "F003;Inception;Thriller;148;Thriller;32.0";
128         when(mockDao.dodajFilm(anyString())).thenReturn("F003");
129         // doNothing() dla metod void - ref. z instrukcji:
               "doNothing().when()"
130         doNothing().when(mockDao).dodajWpisDoLogu(anyString());
131 
132         // Gdy: Wywołujemy dodajFilm
133         model.dodajFilm(daneFilmu);
134 
135         // Wtedy: Logowanie powinno nastąpić
136         verify(mockDao).dodajWpisDoLogu(contains("F003"));
137         verify(mockDao, atLeastOnce()).dodajWpisDoLogu(anyString());
138     }
139 
140     // ========== TESTY KOLEJNOŚCI WYWOŁAŃ - InOrder ==========
141 
142     @Test
143     @Order(4)
144     @DisplayName("Test kolejności wywołań: najpierw dodajFilm, potem
        log")
145     void testKolejnoscWywolanDAO() {
146         // Jeśli: Określamy kolejność wywołań
147         // Ref. z instrukcji: "klasa InOrder w Mockito"
148         String daneFilmu = "F004;Titanic;Romans;195;Dramat;25.0";
149         when(mockDao.dodajFilm(anyString())).thenReturn("F004");
150         InOrder inOrder = inOrder(mockDao);
151 
152         // Gdy: Wywołujemy dodajFilm
153         model.dodajFilm(daneFilmu);
154 
155         // Wtedy: Najpierw dodajFilm, potem log
```

```
156         inOrder.verify(mockDao).dodajFilm(anyString());
157         inOrder.verify(mockDao).dodajWpisDoLogu(anyString());
158     }
159 
160     // ========== TESTY Z WHEN().THENTHROW() ==========
161 
162     @Test
163     @Order(5)
164     @DisplayName("Test obsługi wyjątku z DAO.dodajFilm")
165     void testDodajFilmWyjatekDAO() {
166         // Jeśli: Mock DAO rzuca wyjątek
167         // Ref. z instrukcji: "when().thenThrow() w Mockito"
168         String daneFilmu = "F005;Error;Opis;120;Gatunek;20.0";
169         when(mockDao.dodajFilm(anyString()))
170                 .thenThrow(new RuntimeException("Błąd zapisu do bazy
                        danych"));
171 
172         // Gdy/Wtedy: Operacja powinna propagować wyjątek
173         RuntimeException exception =
                assertThrows(RuntimeException.class,
174                 () -> model.dodajFilm(daneFilmu),
175                 "Powinien wystąpić RuntimeException");
176 
177         assertTrue(exception.getMessage().contains("Błąd zapisu"),
178                 "Komunikat wyjątku powinien zawierać informację o
                        błędzie");
179     }
180 
181     // ========== TESTY WERYFIKACJI PARAMETRÓW ==========
182 
183     @Test
184     @Order(6)
185     @DisplayName("Test że dane filmu są przekazywane do DAO w poprawnym
            formacie")
186     void testFormatDanychPrzekzywanychDoDAO() {
187         // Jeśli: Dane filmu
188         String daneFilmu = "F006;TestFilm;TestOpis;100;Gatunek;20.0";
189         when(mockDao.dodajFilm(anyString())).thenReturn("F006");
190 
191         // Gdy: Wywołujemy dodajFilm
192         model.dodajFilm(daneFilmu);
193 
194         // Wtedy: Weryfikujemy że dane zawierają kluczowe elementy
195         ArgumentCaptor<String> captor =
                ArgumentCaptor.forClass(String.class);
196         verify(mockDao).dodajFilm(captor.capture());
197 
198         String przekazaneDane = captor.getValue();
199         assertTrue(przekazaneDane.contains("F006"), "Dane powinny
                zawierać ID");
200         assertTrue(przekazaneDane.contains("TestFilm"), "Dane powinny
                zawierać tytuł");
201     }
202 
203     // ========== TESTY PARAMETRYZOWANE Z MOCKOWANIEM ==========
204 
205     @ParameterizedTest
206     @Order(7)
```

```
207     @DisplayName("Test dodawania różnych filmów - parametryzowany")
208     @CsvSource({
209             "F010, Parasite, Dramat, 132, Dramat, 28.0",
210             "F011, Joker, Psychologiczny, 122, Thriller, 30.0",
211             "F012, 1917, Wojenny, 119, Wojenny, 27.5"
212     })
213     void testDodajFilmParametryzowany(String id, String tytul, String
            opis,
214             int czas, String gatunek, double cena) {
215         // Jeśli: Mock DAO zwraca odpowiednie ID
216         String daneFilmu = id + ";" + tytul + ";" + opis + ";" + czas +
                ";" + gatunek + ";" + cena;
217         when(mockDao.dodajFilm(anyString())).thenReturn(id);
218 
219         // Gdy: Wywołujemy dodajFilm
220         String wynik = model.dodajFilm(daneFilmu);
221 
222         // Wtedy: Film powinien być "dodany" z poprawnym ID
223         assertNotNull(wynik);
224         assertTrue(wynik.contains(id), "Wynik powinien zawierać ID: " +
                id);
225         verify(mockDao).dodajFilm(contains(tytul));
226     }
227 
228     @ParameterizedTest
229     @Order(8)
230     @DisplayName("Test dodawania filmów z różnymi ID - @ValueSource")
231     @ValueSource(strings = { "F100", "F200", "F300", "F400" })
232     void testDodajFilmRozneId(String id) {
233         // Jeśli: Mock DAO zwraca określone ID
234         String daneFilmu = id + ";Film;Opis;120;Gatunek;25.0";
235         when(mockDao.dodajFilm(anyString())).thenReturn(id);
236 
237         // Gdy: Wywołujemy dodajFilm
238         String wynik = model.dodajFilm(daneFilmu);
239 
240         // Wtedy: Wynik powinien zawierać to ID
241         assertTrue(wynik.contains(id));
242         verify(mockDao, atMostOnce()).dodajFilm(anyString());
243     }
244 
245     // ========== TESTY WERYFIKACJI LICZBY WYWOŁAŃ ==========
246 
247     @Test
248     @Order(9)
249     @DisplayName("Test że dodanie wielu filmów wywołuje DAO odpowiednią
            liczbę razy")
250     void testDodanieWieluFilmow() {
251         // Jeśli: Mock DAO zwraca różne ID
252         when(mockDao.dodajFilm(anyString()))
253                 .thenReturn("F001")
254                 .thenReturn("F002")
255                 .thenReturn("F003");
256 
257         // Gdy: Dodajemy 3 filmy
258         model.dodajFilm("F001;Film1;Opis1;90;Gatunek;20.0");
259         model.dodajFilm("F002;Film2;Opis2;100;Gatunek;25.0");
260         model.dodajFilm("F003;Film3;Opis3;110;Gatunek;30.0");
```

```
261
262         // Wtedy: DAO.dodajFilm powinno być wywołane 3 razy
263         // Ref. z instrukcji: "atLeast(), times()"
264         verify(mockDao, times(3)).dodajFilm(anyString());
265         verify(mockDao, atLeast(3)).dodajWpisDoLogu(anyString());
266     }
267
268     @Test
269     @Order(10)
270     @DisplayName("Test że atMost weryfikuje maksymalną liczbę wywołań")
271     void testAtMostWywolania() {
272         // Jeśli: Mock DAO
273         when(mockDao.dodajFilm(anyString())).thenReturn("FX");
274
275         // Gdy: Dodajemy 2 filmy
276         model.dodajFilm("FX;Film1;Opis;90;Gatunek;20.0");
277         model.dodajFilm("FX;Film2;Opis;100;Gatunek;25.0");
278
279         // Wtedy: Weryfikacja atMost
280         // Ref. z instrukcji: "atMost(), atMostOnce()"
281         verify(mockDao, atMost(5)).dodajFilm(anyString());
282         verify(mockDao, atMost(5)).dodajWpisDoLogu(anyString());
283     }
284
285     // ========== TEST DORETURN().WHEN() DLA VOID ==========
286
287     @Test
288     @Order(11)
289     @DisplayName("Test użycia doNothing().when() dla metody void")
290     void testDoNothingDlaMetodyVoid() {
291         // Jeśli: Konfigurujemy mock dla metody void
292         // Ref. z instrukcji: "doNothing().when() w Mockito"
293         String daneFilmu = "FV;VoidTest;Opis;90;Gatunek;15.0";
294         when(mockDao.dodajFilm(anyString())).thenReturn("FV");
295         doNothing().when(mockDao).dodajWpisDoLogu(anyString());
296
297         // Gdy: Wywołujemy dodajFilm
298         String wynik = model.dodajFilm(daneFilmu);
299
300         // Wtedy: Operacja powinna się zakończyć sukcesem
301         assertNotNull(wynik);
302         verify(mockDao).dodajWpisDoLogu(anyString());
303     }
304
305     // ========== TEST DOTHROW().WHEN() DLA VOID ==========
306
307     @Test
308     @Order(12)
309     @DisplayName("Test użycia doThrow().when() dla metody void")
310     void testDoThrowDlaMetodyVoid() {
311         // Jeśli: Mock dla metody void rzuca wyjątek
312         // Ref. z instrukcji: "doThrow().when() w Mockito"
313         String daneFilmu = "FE;ErrorLog;Opis;90;Gatunek;15.0";
314         when(mockDao.dodajFilm(anyString())).thenReturn("FE");
315         doThrow(new RuntimeException("Błąd logowania"))
316                 .when(mockDao).dodajWpisDoLogu(anyString());
317
318         // Gdy/Wtedy: Wyjątek powinien być propagowany
```

```
319             assertThrows(RuntimeException.class,
320                     () -> model.dodajFilm(daneFilmu),
321                     "Wyjątek z logowania powinien być propagowany");
322         }
323 
324         // ========== TEST NEVER() ==========
325 
326         @Test
327         @Order(13)
328         @DisplayName("Test że metody niepowiązane nie są wywoływane")
329         void testNeverWywolania() {
330             // Jeśli: Mock DAO
331             String daneFilmu = "FN;NeverTest;Opis;90;Gatunek;15.0";
332             when(mockDao.dodajFilm(anyString())).thenReturn("FN");
333 
334             // Gdy: Wywołujemy dodajFilm
335             model.dodajFilm(daneFilmu);
336 
337             // Wtedy: Metody edycji i usuwania nie powinny być wywołane
338             // Ref. z instrukcji: "never() w Mockito"
339             verify(mockDao, never()).edytujFilm(anyString());
340             verify(mockDao, never()).usunFilm(anyString());
341             verify(mockDao, never()).znajdzFilm(anyString());
342         }
343 }
```

3.1.2 controller.TestAdminControllerDodawanieFilmuMock

Plik: src/test/java/controller/TestAdminControllerDodawanieFilmuMock.java

```
1  package controller;
2
3  import model.*;
4  import org.junit.jupiter.api.*;
5  import org.junit.jupiter.api.Tag;
6  import org.junit.jupiter.api.extension.ExtendWith;
7  import org.junit.jupiter.params.ParameterizedTest;
8  import org.junit.jupiter.params.provider.CsvSource;
9  import org.junit.jupiter.params.provider.ValueSource;
10 import org.mockito.*;
11 import org.mockito.junit.jupiter.MockitoExtension;
12
13 import static org.junit.jupiter.api.Assertions.*;
14 import static org.mockito.Mockito.*;
15
16 /**
17  * Testy jednostkowe dla klasy AdminController - operacja dodawania
        filmu z
18  * użyciem mockowania.
19  * Testuje metodę AdminController.dodajFilm() z symulacją zależności
        (IModel).
20  *
21  * Przypadek użycia: Dodanie filmu do oferty
22  * Warstwa: Kontroli (controller)
23  * Zadanie: 2 (testy z mockowaniem)
24  *
25  * Ref. z instrukcji: "Testy klas modelujących elementarne usługi
        biznesowe
26  * (np. krok realizacji przypadku użycia) w warstwie kontroli"
27  */
28 @DisplayName("Testy klasy AdminController - dodawanie filmu z
       mockowaniem")
29 @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
30 @ExtendWith(MockitoExtension.class)
31 @Tag("kontroler")
32 @Tag("dodawanie")
33 @Tag("mock")
34 class TestAdminControllerDodawanieFilmuMock {
35
36     /**
37      * Mock obiektu Model - symulacja warstwy modelu.
38      * Ref. z instrukcji: "operacja mock() lub adnotacja @Mock"
39      */
40     @Mock
41     private IModel mockModel;
42
43     /**
44      * Testowany kontroler - NIE używamy @InjectMocks bo AdminController
45      * tworzy wewnętrznie EdytowanieOfertyKina, więc tworzymy go
           ręcznie.
46      */
47     private AdminController adminController;
48
49     @BeforeAll
50     static void setUpBeforeClass() {
```

```
51          // Przygotowanie przed wszystkimi testami
52          System.out.println("Rozpoczęcie testów AdminController z
                mockowaniem");
53      }
54
55      @BeforeEach
56      void setUp() {
57          // Jeśli: Utworzenie kontrolera z mockowanym modelem
58          adminController = new AdminController(mockModel);
59      }
60
61      @AfterEach
62      void tearDown() {
63          // Sprzątanie po każdym teście
64          reset(mockModel);
65          adminController = null;
66      }
67
68      @AfterAll
69      static void tearDownAfterClass() {
70          // Sprzątanie po wszystkich testach
71          System.out.println("Zakończenie testów AdminController z
                mockowaniem");
72      }
73
74      // ========== TESTY DELEGACJI DO MODELU ==========
75
76      @Test
77      @Order(1)
78      @DisplayName("Test że AdminController deleguje do
            Model.dodajFilm()")
79      void testDelegacjaDoModelu() {
80          // Jeśli: Mock Model zwraca sukces
81          // Ref. z instrukcji: "when().thenReturn()"
82          String daneFilmu = "F001;Matrix;Cyberpunk;136;SciFi;28.5";
83          when(mockModel.dodajFilm(anyString())).thenReturn("Film dodany
                pomyslnie. ID: F001");
84
85          // Gdy: Wywołujemy dodajFilm przez kontroler
86          String wynik = adminController.dodajFilm(daneFilmu);
87
88          // Wtedy: Model powinien być wywołany
89          assertNotNull(wynik);
90          assertTrue(wynik.contains("pomyslnie"));
91
92          // Ref. z instrukcji: "verify(), times()"
93          verify(mockModel, times(1)).dodajFilm(anyString());
94      }
95
96      @Test
97      @Order(2)
98      @DisplayName("Test że wynik z modelu jest zwracany bez modyfikacji")
99      void testZwracanieWynikuBezModyfikacji() {
100         // Jeśli: Model zwraca określony komunikat
101         String oczekiwanyWynik = "Film dodany pomyslnie. ID: F001";
102         when(mockModel.dodajFilm(anyString())).thenReturn(oczekiwanyWynik);
103
104         // Gdy: Wywołujemy przez kontroler
```

```
105         String wynik =
                adminController.dodajFilm("F001;Film;Opis;120;Gatunek;20.0");
106
107         // Wtedy: Wynik powinien być identyczny
108         assertEquals(oczekiwanyWynik, wynik, "Wynik powinien być
                identyczny z tym z modelu");
109     }
110
111     @Test
112     @Order(3)
113     @DisplayName("Test że dane są przekazywane do modelu")
114     void testPrzekazywanieDanych() {
115         // Jeśli: Dane filmu
116         String daneFilmu =
                "F001;TestFilm;TestOpis;100;TestGatunek;25.0";
117         when(mockModel.dodajFilm(anyString())).thenReturn("OK");
118
119         // Gdy: Wywołujemy kontroler
120         adminController.dodajFilm(daneFilmu);
121
122         // Wtedy: Dane powinny być przekazane do modelu (choć przez
                strategię)
123         // Model.dodajFilm powinien być wywołany
124         verify(mockModel).dodajFilm(anyString());
125     }
126
127     // ========== TESTY OBSŁUGI BŁĘDÓW ==========
128
129     @Test
130     @Order(4)
131     @DisplayName("Test obsługi wyjątku z modelu")
132     void testWyjatekZModelu() {
133         // Jeśli: Model rzuca wyjątek
134         // Ref. z instrukcji: "when().thenThrow()"
135         when(mockModel.dodajFilm(anyString()))
136                 .thenThrow(new RuntimeException("Błąd w modelu"));
137
138         // Gdy/Wtedy: Kontroler powinien propagować wyjątek
139         RuntimeException ex = assertThrows(RuntimeException.class,
140                 () ->
                        adminController.dodajFilm("F001;Film;Opis;100;Gatunek;20.0"));
141
142         assertTrue(ex.getMessage().contains("Błąd w modelu"));
143     }
144
145     @Test
146     @Order(5)
147     @DisplayName("Test że wyjątek nie powoduje wielokrotnych wywołań")
148     void testWyjatekNiePowodujePonownegoWywolania() {
149         // Jeśli: Model rzuca wyjątek
150         when(mockModel.dodajFilm(anyString()))
151                 .thenThrow(new RuntimeException("Error"));
152
153         // Gdy: Próba dodania filmu
154         try {
155             adminController.dodajFilm("FX;Film;Opis;100;Gatunek;20.0");
156         } catch (RuntimeException e) {
157             // Oczekiwany wyjątek
```

```
158         }
159
160         // Wtedy: Model powinien być wywołany tylko raz
161         // Ref. z instrukcji: "atMostOnce()"
162         verify(mockModel, atMostOnce()).dodajFilm(anyString());
163     }
164
165     // ========== TESTY KOLEJNOŚCI - InOrder ==========
166
167     @Test
168     @Order(6)
169     @DisplayName("Test kolejności wywołań z InOrder")
170     void testKolejnoscWywolan() {
171         // Jeśli: Mock modelu
172         // Ref. z instrukcji: "klasa InOrder w Mockito"
173         when(mockModel.dodajFilm(anyString())).thenReturn("OK");
174         InOrder inOrder = inOrder(mockModel);
175
176         // Gdy: Wywołujemy dodajFilm
177         adminController.dodajFilm("F001;Film;Opis;100;Gatunek;20.0");
178
179         // Wtedy: Model.dodajFilm powinien być wywołany
180         inOrder.verify(mockModel).dodajFilm(anyString());
181     }
182
183     // ========== TESTY WERYFIKACJI NEVER ==========
184
185     @Test
186     @Order(7)
187     @DisplayName("Test że inne metody modelu nie są wywoływane")
188     void testNieWywolywanieInnychMetod() {
189         // Jeśli: Mock modelu
190         when(mockModel.dodajFilm(anyString())).thenReturn("OK");
191
192         // Gdy: Wywołujemy dodajFilm
193         adminController.dodajFilm("F001;Film;Opis;100;Gatunek;20.0");
194
195         // Wtedy: Inne metody nie powinny być wywołane
196         // Ref. z instrukcji: "never()"
197         verify(mockModel, never()).edytujFilm(anyString(), anyString());
198         verify(mockModel, never()).usunFilm(anyString());
199         verify(mockModel, never()).pobierzRepertuar(anyString());
200     }
201
202     // ========== TESTY PARAMETRYZOWANE ==========
203
204     @ParameterizedTest
205     @Order(8)
206     @DisplayName("Test dodawania różnych filmów - @CsvSource")
207     @CsvSource({
208             "F001, Matrix, SciFi, 136, SciFi, 28.0",
209             "F002, Avatar, Fantasy, 162, SciFi, 35.0",
210             "F003, Titanic, Romans, 195, Dramat, 25.0"
211     })
212     void testDodajFilmParametryzowany(String id, String tytul, String
        opis,
213             int czas, String gatunek, double cena) {
214         // Jeśli: Mock zwraca sukces
```

```
215         String expectedResult = "Film dodany pomyslnie. ID: " + id;
216         when(mockModel.dodajFilm(anyString())).thenReturn(expectedResult);
217 
218         // Gdy: Wywołujemy z parametrami
219         String daneFilmu = id + ";" + tytul + ";" + opis + ";" + czas +
                ";" + gatunek + ";" + cena;
220         String wynik = adminController.dodajFilm(daneFilmu);
221 
222         // Wtedy: Wynik powinien zawierać ID
223         assertTrue(wynik.contains(id), "Wynik powinien zawierać ID: " +
                id);
224         verify(mockModel).dodajFilm(anyString());
225     }
226 
227     @ParameterizedTest
228     @Order(9)
229     @DisplayName("Test dodawania filmów z różnymi komunikatami -
            @ValueSource")
230     @ValueSource(strings = {
231             "Film dodany pomyslnie. ID: F001",
232             "Sukces - film F002 został dodany",
233             "OK"
234     })
235     void testRozneKomunikatyZwrotne(String komunikatZModelu) {
236         // Jeśli: Model zwraca różne komunikaty
237         when(mockModel.dodajFilm(anyString())).thenReturn(komunikatZModelu);
238 
239         // Gdy: Wywołujemy kontroler
240         String wynik =
                adminController.dodajFilm("FX;Film;Opis;100;Gatunek;20.0");
241 
242         // Wtedy: Komunikat powinien być przekazany bez zmian
243         assertEquals(komunikatZModelu, wynik);
244     }
245 
246     // ========== TESTY WIELOKROTNYCH WYWOŁAŃ ==========
247 
248     @Test
249     @Order(10)
250     @DisplayName("Test wielokrotnego dodawania filmów")
251     void testWielokrotneDodawanieFilmow() {
252         // Jeśli: Model zwraca różne odpowiedzi
253         when(mockModel.dodajFilm(anyString()))
254                 .thenReturn("Film dodany. ID: F001")
255                 .thenReturn("Film dodany. ID: F002")
256                 .thenReturn("Film dodany. ID: F003");
257 
258         // Gdy: Dodajemy 3 filmy
259         String wynik1 =
                adminController.dodajFilm("F001;Film1;Opis;100;Gatunek;20.0");
260         String wynik2 =
                adminController.dodajFilm("F002;Film2;Opis;110;Gatunek;25.0");
261         String wynik3 =
                adminController.dodajFilm("F003;Film3;Opis;120;Gatunek;30.0");
262 
263         // Wtedy: Każdy wynik powinien być inny
264         assertTrue(wynik1.contains("F001"));
265         assertTrue(wynik2.contains("F002"));
```

```
266             assertTrue(wynik3.contains("F003"));
267
268             // Ref. z instrukcji: "times(), atLeast()"
269             verify(mockModel, times(3)).dodajFilm(anyString());
270             verify(mockModel, atLeast(3)).dodajFilm(anyString());
271         }
272
273         @Test
274         @Order(11)
275         @DisplayName("Test weryfikacji atMost")
276         void testAtMostWywolania() {
277             // Jeśli: Mock modelu
278             when(mockModel.dodajFilm(anyString())).thenReturn("OK");
279
280             // Gdy: Dodajemy 2 filmy
281             adminController.dodajFilm("F001;Film1;Opis;100;Gatunek;20.0");
282             adminController.dodajFilm("F002;Film2;Opis;110;Gatunek;25.0");
283
284             // Wtedy: Weryfikacja atMost
285             // Ref. z instrukcji: "atMost()"
286             verify(mockModel, atMost(5)).dodajFilm(anyString());
287         }
288
289         // ========== TEST ARGUMENT CAPTOR ==========
290
291         @Test
292         @Order(12)
293         @DisplayName("Test przechwytywania argumentów przekazanych do
                modelu")
294         void testPrzechwytywanieDanych() {
295             // Jeśli: Mock modelu
296             when(mockModel.dodajFilm(anyString())).thenReturn("OK");
297
298             // Gdy: Wywołujemy z konkretnymi danymi
299             String daneFilmu = "FTEST;CaptorTest;Opis
                    testowy;99;TestGatunek;19.99";
300             adminController.dodajFilm(daneFilmu);
301
302             // Wtedy: Weryfikujemy przekazane dane
303             ArgumentCaptor<String> captor =
                    ArgumentCaptor.forClass(String.class);
304             verify(mockModel).dodajFilm(captor.capture());
305
306             // Dane przekazane przez strategię powinny zawierać elementy
                    oryginalne
307             String captured = captor.getValue();
308             assertNotNull(captured);
309         }
310 }
```

3.1.3 controller.TestDodanieNowegoFilmuMock

Plik: src/test/java/controller/TestDodanieNowegoFilmuMock.java

```
1  package controller;
2
3  import model.*;
4  import org.junit.jupiter.api.*;
5  import org.junit.jupiter.api.Tag;
6  import org.junit.jupiter.api.extension.ExtendWith;
7  import org.junit.jupiter.params.ParameterizedTest;
8  import org.junit.jupiter.params.provider.CsvSource;
9  import org.junit.jupiter.params.provider.ValueSource;
10 import org.mockito.*;
11 import org.mockito.junit.jupiter.MockitoExtension;
12
13 import static org.junit.jupiter.api.Assertions.*;
14 import static org.mockito.Mockito.*;
15
16 /**
17  * Testy jednostkowe dla klasy DodanieNowegoFilmu - strategia dodawania
        filmu.
18  * Testuje metodę DodanieNowegoFilmu.edytujOferte() z symulacją
        zależności
19  * (IModel).
20  *
21  * Przypadek użycia: Dodanie filmu do oferty
22  * Warstwa: Kontroli (controller)
23  * Zadanie: 2 (testy z mockowaniem)
24  *
25  * Ref. z instrukcji: "Testy klas modelujących elementarne usługi
        biznesowe"
26  */
27 @DisplayName("Testy klasy DodanieNowegoFilmu - strategia z mockowaniem")
28 @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
29 @ExtendWith(MockitoExtension.class)
30 @Tag("kontroler")
31 @Tag("dodawanie")
32 @Tag("mock")
33 class TestDodanieNowegoFilmuMock {
34
35     /**
36      * Mock obiektu Model - symulacja warstwy modelu.
37      * Ref. z instrukcji: "adnotacja @Mock"
38      */
39     @Mock
40     private IModel mockModel;
41
42     /**
43      * Testowana strategia.
44      */
45     private DodanieNowegoFilmu strategia;
46
47     @BeforeAll
48     static void setUpBeforeClass() {
49         // Przygotowanie przed wszystkimi testami
50         System.out.println("Rozpoczęcie testów DodanieNowegoFilmu z
              mockowaniem");
51     }
```

```
52
53      @BeforeEach
54      void setUp() {
55          // Jeśli: Utworzenie strategii z mockowanym modelem
56          strategia = new DodanieNowegoFilmu(mockModel);
57      }
58
59      @AfterEach
60      void tearDown() {
61          // Sprzątanie po każdym teście
62          reset(mockModel);
63          strategia = null;
64      }
65
66      @AfterAll
67      static void tearDownAfterClass() {
68          // Sprzątanie po wszystkich testach
69          System.out.println("Zakończenie testów DodanieNowegoFilmu z
                mockowaniem");
70      }
71
72      // ========== TESTY DELEGACJI ==========
73
74      @Test
75      @Order(1)
76      @DisplayName("Test że strategia deleguje do Model.dodajFilm()")
77      void testDelegacjaDoModelu() {
78          // Jeśli: Mock Model zwraca sukces
79          // Ref. z instrukcji: "when().thenReturn()"
80          String daneFilmu = "F001;Matrix;Cyberpunk;136;SciFi;28.5";
81          when(mockModel.dodajFilm(daneFilmu)).thenReturn("Film dodany
                pomyslnie. ID: F001");
82
83          // Gdy: Wywołujemy edytujOferte
84          String wynik = strategia.edytujOferte(daneFilmu);
85
86          // Wtedy: Model.dodajFilm powinien być wywołany raz
87          assertNotNull(wynik);
88          assertTrue(wynik.contains("pomyslnie"));
89
90          // Ref. z instrukcji: "verify(), times()"
91          verify(mockModel, times(1)).dodajFilm(daneFilmu);
92      }
93
94      @Test
95      @Order(2)
96      @DisplayName("Test że dane są przekazywane bez modyfikacji")
97      void testPrzekazywanieDanychBezModyfikacji() {
98          // Jeśli: Dane filmu
99          String daneFilmu = "F001;Original;Description;100;Genre;20.0";
100         when(mockModel.dodajFilm(daneFilmu)).thenReturn("OK");
101
102         // Gdy: Wywołujemy strategię
103         strategia.edytujOferte(daneFilmu);
104
105         // Wtedy: Model powinien otrzymać dokładnie te same dane
106         verify(mockModel).dodajFilm(eq(daneFilmu));
107     }
```

```
108
109     @Test
110     @Order(3)
111     @DisplayName("Test że wynik z modelu jest zwracany")
112     void testZwracanieWyniku() {
113         // Jeśli: Model zwraca określony wynik
114         String oczekiwanyWynik = "Film dodany pomyslnie. ID: F002";
115         when(mockModel.dodajFilm(anyString())).thenReturn(oczekiwanyWynik);
116
117         // Gdy: Wywołujemy strategię
118         String wynik =
                strategia.edytujOferte("F002;Film;Opis;120;Gatunek;25.0");
119
120         // Wtedy: Wynik powinien być dokładnie taki jak z modelu
121         assertEquals(oczekiwanyWynik, wynik);
122     }
123
124     // ========== TESTY Z WYJĄTKAMI ==========
125
126     @Test
127     @Order(4)
128     @DisplayName("Test obsługi wyjątku z modelu")
129     void testWyjatekZModelu() {
130         // Jeśli: Model rzuca wyjątek
131         // Ref. z instrukcji: "when().thenThrow()"
132         when(mockModel.dodajFilm(anyString()))
133                 .thenThrow(new RuntimeException("Błąd dodawania
                        filmu"));
134
135         // Gdy/Wtedy: Strategia powinna propagować wyjątek
136         assertThrows(RuntimeException.class,
137                 () ->
                        strategia.edytujOferte("FE;Error;Opis;100;Gatunek;20.0"));
138     }
139
140     // ========== TESTY KOLEJNOŚCI - InOrder ==========
141
142     @Test
143     @Order(5)
144     @DisplayName("Test kolejności wywołań")
145     void testKolejnoscWywolan() {
146         // Jeśli: InOrder dla weryfikacji
147         // Ref. z instrukcji: "klasa InOrder w Mockito"
148         when(mockModel.dodajFilm(anyString())).thenReturn("OK");
149         InOrder inOrder = inOrder(mockModel);
150
151         // Gdy: Wywołujemy strategię
152         strategia.edytujOferte("F001;Film;Opis;100;Gatunek;20.0");
153
154         // Wtedy: Model.dodajFilm powinien być jedynym wywołaniem
155         inOrder.verify(mockModel).dodajFilm(anyString());
156     }
157
158     // ========== TESTY WERYFIKACJI ==========
159
160     @Test
161     @Order(6)
162     @DisplayName("Test że inne metody modelu nie są wywoływane")
```

```
163     void testNieWywolywanieInnychMetod() {
164         // Jeśli: Mock
165         when(mockModel.dodajFilm(anyString())).thenReturn("OK");
166 
167         // Gdy: Wywołujemy strategię
168         strategia.edytujOferte("F001;Film;Opis;100;Gatunek;20.0");
169 
170         // Wtedy: Tylko dodajFilm powinien być wywołany
171         // Ref. z instrukcji: "never()"
172         verify(mockModel, never()).edytujFilm(anyString(), anyString());
173         verify(mockModel, never()).usunFilm(anyString());
174     }
175 
176     // ========== TESTY PARAMETRYZOWANE ==========
177 
178     @ParameterizedTest
179     @Order(7)
180     @DisplayName("Test strategii z różnymi filmami - @CsvSource")
181     @CsvSource({
182             "F001, Matrix, SciFi, 136, SciFi, 28.0",
183             "F002, Avatar, Fantasy, 162, SciFi, 35.0",
184             "F003, Titanic, Romans, 195, Dramat, 25.0"
185     })
186     void testStrategiaParametryzowana(String id, String tytul, String
        opis,
187             int czas, String gatunek, double cena) {
188         // Jeśli: Mock zwraca sukces
189         String expectedResult = "Film dodany pomyslnie. ID: " + id;
190         String daneFilmu = id + ";" + tytul + ";" + opis + ";" + czas +
            ";" + gatunek + ";" + cena;
191         when(mockModel.dodajFilm(daneFilmu)).thenReturn(expectedResult);
192 
193         // Gdy: Wywołujemy strategię
194         String wynik = strategia.edytujOferte(daneFilmu);
195 
196         // Wtedy: Wynik powinien zawierać ID
197         assertTrue(wynik.contains(id));
198         verify(mockModel).dodajFilm(daneFilmu);
199     }
200 
201     @ParameterizedTest
202     @Order(8)
203     @DisplayName("Test strategii z różnymi komunikatami - @ValueSource")
204     @ValueSource(strings = { "OK", "Sukces", "Film dodany" })
205     void testRozneKomunikaty(String komunikat) {
206         // Jeśli: Model zwraca różne komunikaty
207         when(mockModel.dodajFilm(anyString())).thenReturn(komunikat);
208 
209         // Gdy: Wywołujemy strategię
210         String wynik =
                strategia.edytujOferte("FX;Film;Opis;100;Gatunek;20.0");
211 
212         // Wtedy: Komunikat powinien być przekazany
213         assertEquals(komunikat, wynik);
214     }
215 
216     // ========== TESTY WIELOKROTNYCH WYWOŁAŃ ==========
217 
```

```
218     @Test
219     @Order(9)
220     @DisplayName("Test wielokrotnego użycia strategii")
221     void testWielokrotneUzycie() {
222         // Jeśli: Mock zwraca różne wyniki
223         when(mockModel.dodajFilm(anyString()))
224                 .thenReturn("ID: F001")
225                 .thenReturn("ID: F002");
226 
227         // Gdy: Używamy strategii wielokrotnie
228         String wynik1 =
               strategia.edytujOferte("F001;Film1;Opis;100;Gatunek;20.0");
229         String wynik2 =
               strategia.edytujOferte("F002;Film2;Opis;110;Gatunek;25.0");
230 
231         // Wtedy: Każdy wynik powinien być inny
232         assertNotEquals(wynik1, wynik2);
233 
234         // Ref. z instrukcji: "times(), atLeast()"
235         verify(mockModel, times(2)).dodajFilm(anyString());
236         verify(mockModel, atLeast(2)).dodajFilm(anyString());
237         verify(mockModel, atMost(5)).dodajFilm(anyString());
238     }
239 }
```

3.2 Przypadek użycia: Przeglądanie repertuaru

W tym zadaniu testujemy klasy z symulacją zależności (z mockowaniem Mockito).

3.2.1 model.TestModelPobierzRepertuarMock

Plik: `src/test/java/model/TestModelPobierzRepertuarMock.java`

```java
package model;

import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.extension.ExtendWith;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;
import org.mockito.*;
import org.mockito.junit.jupiter.MockitoExtension;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

/**
 * Testy jednostkowe dla klasy Model - pobieranie repertuaru z mockowaniem.
 * Testuje metodę Model.pobierzRepertuar() z symulacją zależności (IDAO).
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Warstwa: Encja (model)
 * Zadanie: 2 (testy z mockowaniem)
 *
 * Ref. z instrukcji: "Symulować należy te fragmenty kodu (obiekty, operacje),
 * od których zależy testowana operacja"
 */
@DisplayName("Testy klasy Model - pobieranie repertuaru z mockowaniem")
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@ExtendWith(MockitoExtension.class)
@Tag("encja")
@Tag("repertuar")
@Tag("mock")
class TestModelPobierzRepertuarMock {

    /**
     * Mock obiektu DAO - symulacja warstwy dostępu do danych.
     * Ref. z instrukcji: "operacja mock() lub adnotacja @Mock"
     */
    @Mock
    private IDAO mockDao;

    /**
     * Mock obiektu Oferta (wymagany przez konstruktor Model).
     */
    @Mock
    private Oferta mockOferta;

    /**
```

```
48          * Testowany obiekt Model z wstrzykniętymi symulacjami.
49          * Ref. z instrukcji: "adnotacja @InjectMocks w Mockito"
50          */
51         @InjectMocks
52         private Model model;
53 
54         @BeforeAll
55         static void setUpBeforeClass() {
56             // Przygotowanie przed wszystkimi testami
57             System.out.println("Rozpoczęcie testów Model z mockowaniem -
                pobieranie repertuaru");
58         }
59 
60         @BeforeEach
61         void setUp() {
62             // Jeśli: Inicjalizacja mocków - wykonywana automatycznie przez
63             // @ExtendWith(MockitoExtension.class)
64         }
65 
66         @AfterEach
67         void tearDown() {
68             // Sprzątanie po każdym teście - resetowanie mocków
69             reset(mockDao, mockOferta);
70         }
71 
72         @AfterAll
73         static void tearDownAfterClass() {
74             // Sprzątanie po wszystkich testach
75             System.out.println("Zakończenie testów Model z mockowaniem -
                pobieranie repertuaru");
76         }
77 
78         // ========== TESTY Z WHEN().THENRETURN() ==========
79 
80         @Test
81         @Order(1)
82         @DisplayName("Test pobierania repertuaru z seansami - sukces")
83         void testPobierzRepertuarZSeansami() {
84             // Jeśli: Mock DAO zwraca seanse dla filmu
85             // Ref. z instrukcji: "when().thenReturn() w Mockito"
86             when(mockDao.znajdzSeansyFilmu("F1")).thenReturn(new String[] {
                    "S1", "S2" });
87             when(mockDao.znajdzSeans("S1")).thenReturn("F1;2024-12-20
                    18:00;Sala1;100");
88             when(mockDao.znajdzSeans("S2")).thenReturn("F1;2024-12-20
                    21:00;Sala2;80");
89 
90             // Gdy: Pobieramy repertuar
91             String repertuar = model.pobierzRepertuar("F1");
92 
93             // Wtedy: Weryfikujemy wynik i użycie mocka
94             assertNotNull(repertuar, "Repertuar nie powinien być null");
95             assertTrue(repertuar.contains("Repertuar"), "Repertuar powinien
                    zawierać nagłówek");
96             assertTrue(repertuar.contains("Seans"), "Repertuar powinien
                    zawierać seanse");
97 
98             // Ref. z instrukcji: "verify(), times()"
```

```
99          verify(mockDao, times(1)).znajdzSeansyFilmu("F1");
100         verify(mockDao, times(2)).znajdzSeans(anyString());
101     }
102
103     @Test
104     @Order(2)
105     @DisplayName("Test pobierania repertuaru bez seansów")
106     void testPobierzRepertuarBezSeansow() {
107         // Jeśli: Mock DAO zwraca pustą tablicę
108         when(mockDao.znajdzSeansyFilmu("F99")).thenReturn(new String[]
                {});
109
110         // Gdy: Pobieramy repertuar dla filmu bez seansów
111         String repertuar = model.pobierzRepertuar("F99");
112
113         // Wtedy: Powinien być komunikat o braku seansów
114         assertNotNull(repertuar);
115         assertTrue(repertuar.contains("Brak seansow"),
116                 "Repertuar powinien zawierać komunikat o braku
                        seansów");
117
118         // znajdzSeans nie powinien być wywołany
119         // Ref. z instrukcji: "never()"
120         verify(mockDao, never()).znajdzSeans(anyString());
121     }
122
123     @Test
124     @Order(3)
125     @DisplayName("Test pobierania repertuaru gdy znajdzSeansyFilmu
            zwraca null")
126     void testPobierzRepertuarZwracaNull() {
127         // Jeśli: Mock DAO zwraca null
128         when(mockDao.znajdzSeansyFilmu("FNULL")).thenReturn(null);
129
130         // Gdy: Pobieramy repertuar
131         String repertuar = model.pobierzRepertuar("FNULL");
132
133         // Wtedy: Powinien być komunikat o braku seansów (nie wyjątek)
134         assertNotNull(repertuar);
135         assertTrue(repertuar.contains("Brak seansow"));
136         verify(mockDao, never()).znajdzSeans(anyString());
137     }
138
139     @Test
140     @Order(4)
141     @DisplayName("Test pobierania repertuaru gdy seans nie istnieje w
            bazie")
142     void testPobierzRepertuarSeansNieIstnieje() {
143         // Jeśli: Mock DAO zwraca ID seansu, ale seans nie istnieje
144         when(mockDao.znajdzSeansyFilmu("F1")).thenReturn(new String[] {
                "S999" });
145         when(mockDao.znajdzSeans("S999")).thenReturn(null);
146
147         // Gdy: Pobieramy repertuar
148         String repertuar = model.pobierzRepertuar("F1");
149
150         // Wtedy: Nie powinno być błędu, ale seans nie będzie w
                repertuarze
```

```
151         assertNotNull(repertuar);
152         verify(mockDao).znajdzSeans("S999");
153     }
154 
155     // ========== TESTY KOLEJNOŚCI WYWOŁAŃ - InOrder ==========
156 
157     @Test
158     @Order(5)
159     @DisplayName("Test kolejności wywołań: najpierw znajdzSeansyFilmu, potem znajdzSeans")
160     void testKolejnoscWywolanDAO() {
161         // Jeśli: Określamy kolejność wywołań
162         // Ref. z instrukcji: "klasa InOrder w Mockito"
163         when(mockDao.znajdzSeansyFilmu("F1")).thenReturn(new String[] { "S1" });
164         when(mockDao.znajdzSeans("S1")).thenReturn("F1;2024-12-20 18:00;Sala1;100");
165         InOrder inOrder = inOrder(mockDao);
166 
167         // Gdy: Pobieramy repertuar
168         model.pobierzRepertuar("F1");
169 
170         // Wtedy: Najpierw szukamy seansów filmu, potem szczegóły seansu
171         inOrder.verify(mockDao).znajdzSeansyFilmu("F1");
172         inOrder.verify(mockDao).znajdzSeans("S1");
173     }
174 
175     @Test
176     @Order(6)
177     @DisplayName("Test kolejności wywołań dla wielu seansów")
178     void testKolejnoscWywolanWieleSeansow() {
179         // Jeśli: Film ma 3 seanse
180         when(mockDao.znajdzSeansyFilmu("F1")).thenReturn(new String[] { "S1", "S2", "S3" });
181         when(mockDao.znajdzSeans(anyString())).thenReturn("F1;2024-12-20 18:00;Sala1;100");
182         InOrder inOrder = inOrder(mockDao);
183 
184         // Gdy: Pobieramy repertuar
185         model.pobierzRepertuar("F1");
186 
187         // Wtedy: Najpierw znajdzSeansyFilmu, potem znajdzSeans dla każdego
188         inOrder.verify(mockDao).znajdzSeansyFilmu("F1");
189         inOrder.verify(mockDao).znajdzSeans("S1");
190         inOrder.verify(mockDao).znajdzSeans("S2");
191         inOrder.verify(mockDao).znajdzSeans("S3");
192     }
193 
194     // ========== TESTY WERYFIKACJI LICZBY WYWOŁAŃ ==========
195 
196     @Test
197     @Order(7)
198     @DisplayName("Test że znajdzSeans jest wywoływane dla każdego ID seansu")
199     void testZnajdzSeansWywolywaneDlaKazdego() {
200         // Jeśli: Mock DAO zwraca 4 seanse
201         when(mockDao.znajdzSeansyFilmu("F1")).thenReturn(new String[] {
```

```
            "S1", "S2", "S3", "S4" });
202         when(mockDao.znajdzSeans(anyString())).thenReturn("dane
                seansu");
203
204         // Gdy: Pobieramy repertuar
205         model.pobierzRepertuar("F1");
206
207         // Wtedy: znajdzSeans powinno być wywołane 4 razy
208         // Ref. z instrukcji: "times(), atLeast()"
209         verify(mockDao, times(4)).znajdzSeans(anyString());
210         verify(mockDao, atLeast(4)).znajdzSeans(anyString());
211         verify(mockDao, atMost(4)).znajdzSeans(anyString());
212     }
213
214     @Test
215     @Order(8)
216     @DisplayName("Test że znajdzSeansyFilmu jest wywoływane dokładnie
            raz")
217     void testZnajdzSeansyFilmuJednorazowo() {
218         // Jeśli: Mock DAO
219         when(mockDao.znajdzSeansyFilmu("F1")).thenReturn(new String[] {
                "S1" });
220         when(mockDao.znajdzSeans("S1")).thenReturn("dane");
221
222         // Gdy: Pobieramy repertuar
223         model.pobierzRepertuar("F1");
224
225         // Wtedy: znajdzSeansyFilmu powinno być wywołane dokładnie raz
226         verify(mockDao, times(1)).znajdzSeansyFilmu("F1");
227         verify(mockDao, atMostOnce()).znajdzSeansyFilmu(anyString());
228     }
229
230     // ========== TESTY Z WHEN().THENTHROW() ==========
231
232     @Test
233     @Order(9)
234     @DisplayName("Test obsługi wyjątku z DAO.znajdzSeansyFilmu")
235     void testWyjatekZZnajdzSeansyFilmu() {
236         // Jeśli: Mock DAO rzuca wyjątek
237         // Ref. z instrukcji: "when().thenThrow() w Mockito"
238         when(mockDao.znajdzSeansyFilmu("FERROR"))
239                 .thenThrow(new RuntimeException("Błąd bazy danych"));
240
241         // Gdy/Wtedy: Operacja powinna propagować wyjątek
242         RuntimeException exception =
                assertThrows(RuntimeException.class,
243                 () -> model.pobierzRepertuar("FERROR"),
244                 "Powinien wystąpić RuntimeException");
245
246         assertTrue(exception.getMessage().contains("Błąd bazy danych"));
247     }
248
249     @Test
250     @Order(10)
251     @DisplayName("Test obsługi wyjątku z DAO.znajdzSeans")
252     void testWyjatekZZnajdzSeans() {
253         // Jeśli: Mock DAO rzuca wyjątek przy znajdzSeans
254         when(mockDao.znajdzSeansyFilmu("F1")).thenReturn(new String[] {
```

```
            "S1" });
        when(mockDao.znajdzSeans("S1"))
                .thenThrow(new RuntimeException("Błąd odczytu seansu"));

        // Gdy/Wtedy: Wyjątek powinien być propagowany
        assertThrows(RuntimeException.class,
                () -> model.pobierzRepertuar("F1"));
    }

    // ========== TESTY WERYFIKACJI ARGUMENTÓW ==========

    @Test
    @Order(11)
    @DisplayName("Test że poprawne ID filmu jest przekazywane do DAO")
    void testPoprawneIdFilmuPrzekazywane() {
        // Jeśli: Mock DAO
        when(mockDao.znajdzSeansyFilmu(anyString())).thenReturn(new
            String[] {});

        // Gdy: Pobieramy repertuar z określonym ID
        model.pobierzRepertuar("FILM_TEST_123");

        // Wtedy: DAO powinno otrzymać dokładnie to ID
        verify(mockDao).znajdzSeansyFilmu(eq("FILM_TEST_123"));
    }

    @Test
    @Order(12)
    @DisplayName("Test przechwytywania argumentów przekazanych do DAO")
    void testPrzechwytywanieDanych() {
        // Jeśli: Mock DAO
        when(mockDao.znajdzSeansyFilmu(anyString())).thenReturn(new
            String[] { "S1" });
        when(mockDao.znajdzSeans(anyString())).thenReturn("dane");

        // Gdy: Pobieramy repertuar
        model.pobierzRepertuar("F_CAPTOR");

        // Wtedy: Weryfikujemy przekazane ID
        ArgumentCaptor<String> captor =
            ArgumentCaptor.forClass(String.class);
        verify(mockDao).znajdzSeansyFilmu(captor.capture());

        assertEquals("F_CAPTOR", captor.getValue(),
                "Przekazane ID powinno być F_CAPTOR");
    }

    // ========== TESTY PARAMETRYZOWANE ==========

    @ParameterizedTest
    @Order(13)
    @DisplayName("Test pobierania repertuaru dla różnych filmów -
        @CsvSource")
    @CsvSource({
            "F1, 2, 2024-12-20 18:00",
            "F2, 1, 2024-12-21 19:00",
            "F3, 3, 2024-12-22 20:00"
    })
```

```
308     void testPobierzRepertuarParametryzowany(String idFilmu, int
            liczbaSeansow, String data) {
309         // Jeśli: Mock DAO zwraca określoną liczbę seansów
310         String[] seansyIds = new String[liczbaSeansow];
311         for (int i = 0; i < liczbaSeansow; i++) {
312             seansyIds[i] = "S" + (i + 1);
313         }
314         when(mockDao.znajdzSeansyFilmu(idFilmu)).thenReturn(seansyIds);
315         when(mockDao.znajdzSeans(anyString())).thenReturn(idFilmu + ";"
                + data + ";Sala1;100");
316 
317         // Gdy: Pobieramy repertuar
318         String repertuar = model.pobierzRepertuar(idFilmu);
319 
320         // Wtedy: Weryfikujemy wynik
321         assertNotNull(repertuar);
322         assertTrue(repertuar.contains(idFilmu));
323         verify(mockDao, times(liczbaSeansow)).znajdzSeans(anyString());
324     }
325 
326     @ParameterizedTest
327     @Order(14)
328     @DisplayName("Test pobierania repertuaru dla różnych ID filmów -
            @ValueSource")
329     @ValueSource(strings = { "F1", "F10", "FILM_2024", "XYZ123" })
330     void testPobierzRepertuarRozneIdFilmow(String idFilmu) {
331         // Jeśli: Mock DAO zwraca pusty wynik
332         when(mockDao.znajdzSeansyFilmu(idFilmu)).thenReturn(new
                String[] {});
333 
334         // Gdy: Pobieramy repertuar
335         String repertuar = model.pobierzRepertuar(idFilmu);
336 
337         // Wtedy: Powinien być komunikat o braku seansów z ID filmu
338         assertNotNull(repertuar);
339         assertTrue(repertuar.contains("Brak seansow"));
340         assertTrue(repertuar.contains(idFilmu));
341         verify(mockDao).znajdzSeansyFilmu(idFilmu);
342     }
343 
344     // ========== TESTY WIELOKROTNYCH WYWOŁAŃ ==========
345 
346     @Test
347     @Order(15)
348     @DisplayName("Test wielokrotnego pobierania repertuaru")
349     void testWielokrotnePobieranieRepertuaru() {
350         // Jeśli: Mock DAO zwraca różne wyniki
351         when(mockDao.znajdzSeansyFilmu("F1")).thenReturn(new String[] {
                "S1" });
352         when(mockDao.znajdzSeansyFilmu("F2")).thenReturn(new String[] {
                "S2", "S3" });
353         when(mockDao.znajdzSeans(anyString())).thenReturn("dane
                seansu");
354 
355         // Gdy: Pobieramy repertuar dla różnych filmów
356         String repertuarF1 = model.pobierzRepertuar("F1");
357         String repertuarF2 = model.pobierzRepertuar("F2");
358 
```

```
359         // Wtedy: Każde wywołanie powinno użyć odpowiedniego ID
360         verify(mockDao).znajdzSeansyFilmu("F1");
361         verify(mockDao).znajdzSeansyFilmu("F2");
362         verify(mockDao, times(3)).znajdzSeans(anyString()); // 1 + 2 = 3
363     }
364
365     @Test
366     @Order(16)
367     @DisplayName("Test że inne metody DAO nie są wywoływane")
368     void testNieWywolywanieInnychMetod() {
369         // Jeśli: Mock DAO
370         when(mockDao.znajdzSeansyFilmu("F1")).thenReturn(new String[]
                {});
371
372         // Gdy: Pobieramy repertuar
373         model.pobierzRepertuar("F1");
374
375         // Wtedy: Inne metody nie powinny być wywołane
376         // Ref. z instrukcji: "never()"
377         verify(mockDao, never()).dodajSeans(anyString());
378         verify(mockDao, never()).usunSeans(anyString());
379         verify(mockDao, never()).dodajFilm(anyString());
380     }
381 }
```

3.2.2 controller.TestClientControllerPrzegladanieRepertuaruMock

Plik: src/test/java/controller/TestClientControllerPrzegladanieRepertuaruMock.java

```
1  package controller;
2
3  import model.*;
4  import org.junit.jupiter.api.*;
5  import org.junit.jupiter.api.Tag;
6  import org.junit.jupiter.api.extension.ExtendWith;
7  import org.junit.jupiter.params.ParameterizedTest;
8  import org.junit.jupiter.params.provider.CsvSource;
9  import org.junit.jupiter.params.provider.ValueSource;
10 import org.mockito.*;
11 import org.mockito.junit.jupiter.MockitoExtension;
12
13 import static org.junit.jupiter.api.Assertions.*;
14 import static org.mockito.Mockito.*;
15
16 /**
17  * Testy jednostkowe dla klasy ClientController - przeglądanie repertuaru z
18  * mockowaniem.
19  * Testuje metodę ClientController.przegladajRepertuar() z symulacją zależności
20  * (IModel).
21  *
22  * Przypadek użycia: Przeglądanie repertuaru
23  * Warstwa: Kontroli (controller)
24  * Zadanie: 2 (testy z mockowaniem)
25  *
26  * Ref. z instrukcji: "Testy klas modelujących elementarne usługi biznesowe
27  * w warstwie kontroli"
28  */
29 @DisplayName("Testy klasy ClientController - przeglądanie repertuaru z mockowaniem")
30 @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
31 @ExtendWith(MockitoExtension.class)
32 @Tag("kontroler")
33 @Tag("repertuar")
34 @Tag("mock")
35 class TestClientControllerPrzegladanieRepertuaruMock {
36
37     /**
38      * Mock obiektu Model - symulacja warstwy modelu.
39      * Ref. z instrukcji: "adnotacja @Mock"
40      */
41     @Mock
42     private IModel mockModel;
43
44     /**
45      * Testowany kontroler z wstrzykniętą symulacją modelu.
46      * Ref. z instrukcji: "adnotacja @InjectMocks"
47      */
48     @InjectMocks
49     private ClientController clientController;
50
51     @BeforeAll
```

```
52      static void setUpBeforeClass() {
53          // Przygotowanie przed wszystkimi testami
54          System.out.println("Rozpoczęcie testów ClientController z
                mockowaniem - przeglądanie repertuaru");
55      }
56
57      @BeforeEach
58      void setUp() {
59          // Jeśli: Inicjalizacja mocków - wykonywana automatycznie przez
60          // @ExtendWith(MockitoExtension.class)
61      }
62
63      @AfterEach
64      void tearDown() {
65          // Sprzątanie po każdym teście - resetowanie mocków
66          reset(mockModel);
67      }
68
69      @AfterAll
70      static void tearDownAfterClass() {
71          // Sprzątanie po wszystkich testach
72          System.out.println("Zakończenie testów ClientController z
                mockowaniem - przeglądanie repertuaru");
73      }
74
75      // ========== TESTY DELEGACJI DO MODELU ==========
76
77      @Test
78      @Order(1)
79      @DisplayName("Test że ClientController deleguje do
            Model.pobierzRepertuar()")
80      void testDelegacjaDoModelu() {
81          // Jeśli: Mock Model zwraca repertuar
82          // Ref. z instrukcji: "when().thenReturn()"
83          String oczekiwanyRepertuar = "Repertuar dla filmu F1:\n  -
                Seans: dane";
84          when(mockModel.pobierzRepertuar("F1")).thenReturn(oczekiwanyRepertuar);
85
86          // Gdy: Wywołujemy przegladajRepertuar przez kontroler
87          String wynik = clientController.przegladajRepertuar("F1");
88
89          // Wtedy: Model powinien być wywołany
90          assertNotNull(wynik);
91          assertEquals(oczekiwanyRepertuar, wynik);
92
93          // Ref. z instrukcji: "verify(), times()"
94          verify(mockModel, times(1)).pobierzRepertuar("F1");
95      }
96
97      @Test
98      @Order(2)
99      @DisplayName("Test że wynik z modelu jest zwracany bez modyfikacji")
100     void testZwracanieWynikuBezModyfikacji() {
101         // Jeśli: Model zwraca określony repertuar
102         String repertuarZModelu = "Repertuar testowy:\n  - Seans 1\n  -
                Seans 2";
103         when(mockModel.pobierzRepertuar("F1")).thenReturn(repertuarZModelu);
104
```

```
105         // Gdy: Wywołujemy przez kontroler
106         String wynik = clientController.przegladajRepertuar("F1");
107 
108         // Wtedy: Wynik powinien być identyczny
109         assertEquals(repertuarZModelu, wynik,
110                 "Wynik powinien być identyczny z tym z modelu");
111     }
112 
113     @Test
114     @Order(3)
115     @DisplayName("Test że kryteria są przekazywane do modelu bez
        modyfikacji")
116     void testPrzekazywanieDanych() {
117         // Jeśli: Mock modelu
118         when(mockModel.pobierzRepertuar(anyString())).thenReturn("repertuar");
119 
120         // Gdy: Wywołujemy z konkretnymi kryteriami
121         clientController.przegladajRepertuar("FILM_KRYTERIA_123");
122 
123         // Wtedy: Kryteria powinny być przekazane bez zmian
124         verify(mockModel).pobierzRepertuar(eq("FILM_KRYTERIA_123"));
125     }
126 
127     // ========== TESTY OBSŁUGI BŁĘDÓW ==========
128 
129     @Test
130     @Order(4)
131     @DisplayName("Test obsługi wyjątku z modelu")
132     void testWyjatekZModelu() {
133         // Jeśli: Model rzuca wyjątek
134         // Ref. z instrukcji: "when().thenThrow()"
135         when(mockModel.pobierzRepertuar("FERROR"))
136                 .thenThrow(new RuntimeException("Błąd w modelu"));
137 
138         // Gdy/Wtedy: Kontroler powinien propagować wyjątek
139         RuntimeException ex = assertThrows(RuntimeException.class,
140                 () -> clientController.przegladajRepertuar("FERROR"));
141 
142         assertTrue(ex.getMessage().contains("Błąd w modelu"));
143     }
144 
145     @Test
146     @Order(5)
147     @DisplayName("Test że wyjątek nie powoduje wielokrotnych wywołań")
148     void testWyjatekNiePowodujePonownegoWywolania() {
149         // Jeśli: Model rzuca wyjątek
150         when(mockModel.pobierzRepertuar(anyString()))
151                 .thenThrow(new RuntimeException("Error"));
152 
153         // Gdy: Próba pobrania repertuaru
154         try {
155             clientController.przegladajRepertuar("FX");
156         } catch (RuntimeException e) {
157             // Oczekiwany wyjątek
158         }
159 
160         // Wtedy: Model powinien być wywołany tylko raz
161         // Ref. z instrukcji: "atMostOnce()"
```

```
162             verify(mockModel, atMostOnce()).pobierzRepertuar(anyString());
163         }
164
165         // ========== TESTY KOLEJNOŚCI - InOrder ==========
166
167         @Test
168         @Order(6)
169         @DisplayName("Test kolejności wywołań z InOrder")
170         void testKolejnoscWywolan() {
171             // Jeśli: Mock modelu
172             // Ref. z instrukcji: "klasa InOrder w Mockito"
173             when(mockModel.pobierzRepertuar(anyString())).thenReturn("repertuar");
174             InOrder inOrder = inOrder(mockModel);
175
176             // Gdy: Wywołujemy przegladajRepertuar
177             clientController.przegladajRepertuar("F1");
178
179             // Wtedy: Model.pobierzRepertuar powinien być wywołany
180             inOrder.verify(mockModel).pobierzRepertuar("F1");
181         }
182
183         // ========== TESTY WERYFIKACJI NEVER ==========
184
185         @Test
186         @Order(7)
187         @DisplayName("Test że inne metody modelu nie są wywoływane")
188         void testNieWywolywanieInnychMetod() {
189             // Jeśli: Mock modelu
190             when(mockModel.pobierzRepertuar(anyString())).thenReturn("repertuar");
191
192             // Gdy: Wywołujemy przegladajRepertuar
193             clientController.przegladajRepertuar("F1");
194
195             // Wtedy: Inne metody nie powinny być wywołane
196             // Ref. z instrukcji: "never()"
197             verify(mockModel, never()).dodajFilm(anyString());
198             verify(mockModel, never()).usunFilm(anyString());
199             verify(mockModel, never()).zarezerwujMiejsce(anyString());
200             verify(mockModel, never()).dodajSeans(anyString());
201         }
202
203         // ========== TESTY PARAMETRYZOWANE ==========
204
205         @ParameterizedTest
206         @Order(8)
207         @DisplayName("Test przeglądania repertuaru różnych filmów -
                @CsvSource")
208         @CsvSource({
209                 "F1, Repertuar dla F1",
210                 "F2, Repertuar dla F2",
211                 "F3, Repertuar dla F3"
212         })
213         void testPrzegladajRepertuarParametryzowany(String idFilmu, String
                oczekiwanyRepertuar) {
214             // Jeśli: Mock zwraca odpowiedni repertuar
215             when(mockModel.pobierzRepertuar(idFilmu)).thenReturn(oczekiwanyRepertuar);
216
217             // Gdy: Wywołujemy z parametrami
```

```
218         String wynik = clientController.przegladajRepertuar(idFilmu);
219
220         // Wtedy: Wynik powinien odpowiadać oczekiwanemu
221         assertEquals(oczekiwanyRepertuar, wynik);
222         verify(mockModel).pobierzRepertuar(idFilmu);
223     }
224
225     @ParameterizedTest
226     @Order(9)
227     @DisplayName("Test przeglądania repertuaru z różnymi komunikatami -
        @ValueSource")
228     @ValueSource(strings = {
229             "Repertuar dla filmu F1:\n  - Seans: 18:00",
230             "Repertuar dla filmu F2:\n  - Seans: 20:00\n  - Seans:
                22:00",
231             "Brak seansow dla filmu F99"
232     })
233     void testRozneKomunikatyZwrotne(String komunikatZModelu) {
234         // Jeśli: Model zwraca różne komunikaty
235         when(mockModel.pobierzRepertuar(anyString())).thenReturn(komunikatZModelu);
236
237         // Gdy: Wywołujemy kontroler
238         String wynik = clientController.przegladajRepertuar("FX");
239
240         // Wtedy: Komunikat powinien być przekazany bez zmian
241         assertEquals(komunikatZModelu, wynik);
242     }
243
244     // ========== TESTY WIELOKROTNYCH WYWOŁAŃ ==========
245
246     @Test
247     @Order(10)
248     @DisplayName("Test wielokrotnego przeglądania repertuaru")
249     void testWielokrotnePrzegladanieRepertuaru() {
250         // Jeśli: Model zwraca różne odpowiedzi
251         when(mockModel.pobierzRepertuar("F1")).thenReturn("Repertuar
            F1");
252         when(mockModel.pobierzRepertuar("F2")).thenReturn("Repertuar
            F2");
253         when(mockModel.pobierzRepertuar("F3")).thenReturn("Repertuar
            F3");
254
255         // Gdy: Przeglądamy 3 repertuary
256         String wynik1 = clientController.przegladajRepertuar("F1");
257         String wynik2 = clientController.przegladajRepertuar("F2");
258         String wynik3 = clientController.przegladajRepertuar("F3");
259
260         // Wtedy: Każdy wynik powinien być inny
261         assertEquals("Repertuar F1", wynik1);
262         assertEquals("Repertuar F2", wynik2);
263         assertEquals("Repertuar F3", wynik3);
264
265         // Ref. z instrukcji: "times(), atLeast()"
266         verify(mockModel, times(3)).pobierzRepertuar(anyString());
267         verify(mockModel, atLeast(3)).pobierzRepertuar(anyString());
268     }
269
270     @Test
```

```
271     @Order(11)
272     @DisplayName("Test wielokrotnego przeglądania tego samego
            repertuaru")
273     void testWielokrotnePrzegladanieTegoSamego() {
274         // Jeśli: Model zwraca ten sam wynik
275         when(mockModel.pobierzRepertuar("F1")).thenReturn("Repertuar
                F1");
276 
277         // Gdy: Przeglądamy ten sam repertuar wielokrotnie
278         String wynik1 = clientController.przegladajRepertuar("F1");
279         String wynik2 = clientController.przegladajRepertuar("F1");
280         String wynik3 = clientController.przegladajRepertuar("F1");
281 
282         // Wtedy: Wszystkie wyniki powinny być identyczne
283         assertEquals(wynik1, wynik2);
284         assertEquals(wynik2, wynik3);
285 
286         // Model powinien być wywołany 3 razy
287         verify(mockModel, times(3)).pobierzRepertuar("F1");
288     }
289 
290     @Test
291     @Order(12)
292     @DisplayName("Test weryfikacji atMost")
293     void testAtMostWywolania() {
294         // Jeśli: Mock modelu
295         when(mockModel.pobierzRepertuar(anyString())).thenReturn("repertuar");
296 
297         // Gdy: Przeglądamy 2 repertuary
298         clientController.przegladajRepertuar("F1");
299         clientController.przegladajRepertuar("F2");
300 
301         // Wtedy: Weryfikacja atMost
302         // Ref. z instrukcji: "atMost()"
303         verify(mockModel, atMost(5)).pobierzRepertuar(anyString());
304     }
305 
306     // ========== TEST ARGUMENT CAPTOR ==========
307 
308     @Test
309     @Order(13)
310     @DisplayName("Test przechwytywania argumentów przekazanych do
            modelu")
311     void testPrzechwytywanieDanych() {
312         // Jeśli: Mock modelu
313         when(mockModel.pobierzRepertuar(anyString())).thenReturn("repertuar");
314 
315         // Gdy: Wywołujemy z konkretnymi kryteriami
316         clientController.przegladajRepertuar("CAPTOR_TEST_FILM");
317 
318         // Wtedy: Weryfikujemy przekazane dane
319         ArgumentCaptor<String> captor =
                ArgumentCaptor.forClass(String.class);
320         verify(mockModel).pobierzRepertuar(captor.capture());
321 
322         String przekazane = captor.getValue();
323         assertEquals("CAPTOR_TEST_FILM", przekazane,
324                 "Przekazane kryteria powinny być identyczne");
```

```
325     }
326
327     // ========== TESTY EDGE CASES ==========
328
329     @Test
330     @Order(14)
331     @DisplayName("Test przeglądania repertuaru z pustym ID")
332     void testPrzegladajRepertuarPusteId() {
333         // Jeśli: Model obsługuje puste ID
334         when(mockModel.pobierzRepertuar("")).thenReturn("Brak seansow
                dla filmu: ");
335
336         // Gdy: Wywołujemy z pustym ID
337         String wynik = clientController.przegladajRepertuar("");
338
339         // Wtedy: Powinniśmy otrzymać odpowiedni komunikat
340         assertNotNull(wynik);
341         verify(mockModel).pobierzRepertuar("");
342     }
343
344     @Test
345     @Order(15)
346     @DisplayName("Test gdy model zwraca null")
347     void testModelZwracaNull() {
348         // Jeśli: Model zwraca null
349         when(mockModel.pobierzRepertuar("FNULL")).thenReturn(null);
350
351         // Gdy: Wywołujemy kontroler
352         String wynik = clientController.przegladajRepertuar("FNULL");
353
354         // Wtedy: Wynik powinien być null (przekazany bez modyfikacji)
355         assertNull(wynik, "Jeśli model zwraca null, kontroler powinien
                zwrócić null");
356         verify(mockModel).pobierzRepertuar("FNULL");
357     }
358
359     @Test
360     @Order(16)
361     @DisplayName("Test że kontroler nie modyfikuje danych z modelu")
362     void testNieModyfikujeDanych() {
363         // Jeśli: Model zwraca konkretny tekst z wieloma liniami
364         String oryginalnyRepertuar = "Repertuar dla filmu F1:\n" +
365                 "  - Seans: F1;2024-12-20 18:00;Sala1;100\n" +
366                 "  - Seans: F1;2024-12-20 21:00;Sala2;80\n";
367         when(mockModel.pobierzRepertuar("F1")).thenReturn(oryginalnyRepertuar);
368
369         // Gdy: Wywołujemy kontroler
370         String wynik = clientController.przegladajRepertuar("F1");
371
372         // Wtedy: Wynik powinien być identyczny co do znaku
373         assertEquals(oryginalnyRepertuar, wynik,
374                 "Kontroler nie powinien modyfikować danych");
375         assertEquals(oryginalnyRepertuar.length(), wynik.length(),
376                 "Długość tekstu powinna być identyczna");
377     }
378 }
```

4 Zadanie 3: Zestawy testów (Test Suites)

4.1 Przypadek użycia: Dodanie filmu do oferty

Zestawy testów pozwalają na grupowanie testów według różnych kryteriów.

4.1.1 model.SuiteEncjiDodawanieFilmu

Plik: `src/test/java/model/SuiteEncjiDodawanieFilmu.java`

```java
package model;

import org.junit.platform.suite.api.*;

/**
 * Zestaw testów warstwy encji (model) dla przypadku użycia "Dodanie filmu do
 * oferty".
 *
 * Przypadek użycia: Dodanie filmu do oferty
 * Zadanie: 3 (zestawy testów)
 *
 * Ten zestaw uruchamia wszystkie testy z pakietu model oznaczone tagiem
 * "dodawanie".
 *
 * Ref. z instrukcji: "zestaw testów klas warstwy encji, czyli znajdujących się
 * w jej pakiecie"
 *
 * Praktyczne zastosowanie: Weryfikacja poprawności wszystkich klas encji
 * zaangażowanych w proces dodawania filmu (Film, FabrykaFilmu, DAO, Model).
 */
@Suite
@SuiteDisplayName("Zestaw testów warstwy encji - Dodanie filmu do oferty")
@SelectPackages("model")
@IncludeTags("dodawanie")
public class SuiteEncjiDodawanieFilmu {
    // Klasa zestawu testów - testy są wybierane automatycznie na podstawie
    // adnotacji

    /*
     * Ten zestaw zawiera następujące klasy testów:
     * - TestFilm.java - testy encji Film
     * - TestDAO.java - testy Data Access Object
     * - TestFabrykaStandardowegoFilmu.java - testy fabryki filmów
     * - TestModelDodawanieFilmu.java - testy Model.dodajFilm() bez mockowania
     * - TestModelDodawanieFilmuMock.java - testy Model.dodajFilm() z mockowaniem
     *
     * Wszystkie powyższe klasy są oznaczone tagami @Tag("encja")
     * i @Tag("dodawanie")
     */
}
```

4.1.2 controller.SuiteKontroliDodawanieFilmu

Plik: src/test/java/controller/SuiteKontroliDodawanieFilmu.java

```java
package controller;

import org.junit.platform.suite.api.*;

/**
 * Zestaw testów warstwy kontroli (controller) dla przypadku użycia "Dodanie
 * filmu do oferty".
 *
 * Przypadek użycia: Dodanie filmu do oferty
 * Zadanie: 3 (zestawy testów)
 *
 * Ten zestaw uruchamia wszystkie testy z pakietu controller oznaczone tagiem
 * "dodawanie".
 *
 * Ref. z instrukcji: "zestaw testów klas warstwy kontroli, czyli znajdujących
 * się w jej pakiecie"
 *
 * Praktyczne zastosowanie: Weryfikacja poprawności kontrolerów i strategii
 * zaangażowanych w proces dodawania filmu (AdminController,
 * EdytowanieOfertyKina, DodanieNowegoFilmu).
 */
@Suite
@SuiteDisplayName("Zestaw testów warstwy kontroli - Dodanie filmu do oferty")
@SelectPackages("controller")
@IncludeTags("dodawanie")
public class SuiteKontroliDodawanieFilmu {
    // Klasa zestawu testów - testy są wybierane automatycznie na podstawie
    // adnotacji

    /*
     * Ten zestaw zawiera następujące klasy testów:
     * - TestAdminControllerDodawanieFilmu.java - testy kontrolera bez mockowania
     * - TestAdminControllerDodawanieFilmuMock.java - testy kontrolera z mockowaniem
     * - TestDodanieNowegoFilmuMock.java - testy strategii z mockowaniem
     *
     * Wszystkie powyższe klasy są oznaczone tagami @Tag("kontroler")
     * i @Tag("dodawanie")
     */
}
```

4.1.3 suites.SuiteDodawanieFilmuBezMock

Plik: src/test/java/suites/SuiteDodawanieFilmuBezMock.java

```
1  package suites;
2
3  import org.junit.platform.suite.api.*;
4
5  /**
6   * Zestaw testów dla przypadku użycia "Dodanie filmu do oferty" BEZ
        mockowania.
7   *
8   * Przypadek użycia: Dodanie filmu do oferty
9   * Zadanie: 3 (zestawy testów bazujące na tagach)
10  *
11  * Ten zestaw uruchamia testy oznaczone tagiem "dodawanie", ale
        WYKLUCZA testy z
12  * tagiem "mock".
13  *
14  * Ref. z instrukcji: "przynajmniej 2 zestawy testów oznaczonych
        wybranymi
15  * tagami
16  * i nie oznaczonych innymi wybranymi tagami, które mają jakieś
        praktyczne
17  * zastosowanie"
18  *
19  * Praktyczne zastosowanie:
20  * - Testy integracyjne bez mockowania - sprawdzają rzeczywistą
        współpracę
21  * komponentów
22  * - Szybka weryfikacja podstawowej funkcjonalności dodawania filmu
23  * - Przydatne do testów regresji przed wdrożeniem
24  * - Uruchamiane gdy chcemy sprawdzić pełny przepływ bez izolacji
        zależności
25  */
26 @Suite
27 @SuiteDisplayName("Zestaw testów dodawania filmu - BEZ mockowania")
28 @SelectPackages({ "model", "controller" })
29 @IncludeTags("dodawanie")
30 @ExcludeTags("mock")
31 public class SuiteDodawanieFilmuBezMock {
32     // Klasa zestawu testów - testy są wybierane automatycznie na
          podstawie
33     // adnotacji
34
35     /*
36      * Ten zestaw zawiera następujące klasy testów (bez mockowania):
37      *
38      * Z pakietu model:
39      * - TestFilm.java - testy encji Film
40      * - TestDAO.java - testy Data Access Object
41      * - TestFabrykaStandardowegoFilmu.java - testy fabryki filmów
42      * - TestModelDodawanieFilmu.java - testy Model.dodajFilm() bez
           mockowania
43      *
44      * Z pakietu controller:
45      * - TestAdminControllerDodawanieFilmu.java - testy kontrolera bez
           mockowania
46      *
```

```
47        * WYKLUCZONE (tag "mock"):
48        * - TestModelDodawanieFilmuMock.java
49        * - TestAdminControllerDodawanieFilmuMock.java
50        * - TestDodanieNowegoFilmuMock.java
51        */
52  }
```

4.1.4 suites.SuiteDodawanieFilmuMock

Plik: src/test/java/suites/SuiteDodawanieFilmuMock.java

```java
package suites;

import org.junit.platform.suite.api.*;

/**
 * Zestaw testów dla przypadku użycia "Dodanie filmu do oferty" TYLKO Z
 * mockowaniem.
 *
 * Przypadek użycia: Dodanie filmu do oferty
 * Zadanie: 3 (zestawy testów bazujące na tagach)
 *
 * Ten zestaw uruchamia TYLKO testy oznaczone tagami "dodawanie" ORAZ
    "mock".
 *
 * Ref. z instrukcji: "przynajmniej 2 zestawy testów oznaczonych
    wybranymi
 * tagami
 * i nie oznaczonych innymi wybranymi tagami, które mają jakieś
    praktyczne
 * zastosowanie"
 *
 * Praktyczne zastosowanie:
 * - Szybkie testy jednostkowe z izolacją zależności
 * - Weryfikacja logiki biznesowej bez potrzeby rzeczywistych zależności
 * - Testy uruchamiane podczas CI/CD dla szybkiego feedbacku
 * - Przydatne gdy warstwa DAO lub inne zależności są niedostępne
 * - Testowanie obsługi błędów i edge cases (symulacja wyjątków)
 */
@Suite
@SuiteDisplayName("Zestaw testów dodawania filmu - TYLKO z mockowaniem")
@SelectPackages({ "model", "controller" })
@IncludeTags({ "dodawanie", "mock" })
public class SuiteDodawanieFilmuMock {
    // Klasa zestawu testów - testy są wybierane automatycznie na
       podstawie
    // adnotacji

    /*
     * Ten zestaw zawiera następujące klasy testów (tylko z
        mockowaniem):
     *
     * Z pakietu model:
     * - TestModelDodawanieFilmuMock.java - testy Model z mockowanym DAO
     * * Testuje: when/thenReturn, when/thenThrow, doNothing, doThrow
     * * Weryfikuje: verify, times, never, atLeast, atMost, InOrder
     *
     * Z pakietu controller:
     * - TestAdminControllerDodawanieFilmuMock.java - testy kontrolera
        z mockowanym
     * IModel
     * - TestDodanieNowegoFilmuMock.java - testy strategii z mockowanym
        IModel
     *
     * WYKLUCZONE (brak tagu "mock"):
     * - TestFilm.java
```

```
49          * - TestDAO.java
50          * - TestFabrykaStandardowegoFilmu.java
51          * - TestModelDodawanieFilmu.java
52          * - TestAdminControllerDodawanieFilmu.java
53          */
54  }
```

4.1.5 suites.SuiteDodawanieFilmuWszystkie

Plik: src/test/java/suites/SuiteDodawanieFilmuWszystkie.java

```java
package suites;

import org.junit.platform.suite.api.*;

/**
 * Zestaw testów dla przypadku użycia "Dodanie filmu do oferty" - WSZYSTKIE
 * testy.
 *
 * Przypadek użycia: Dodanie filmu do oferty
 * Zadanie: 3 (zestawy testów)
 *
 * Ten zestaw uruchamia WSZYSTKIE testy związane z dodawaniem filmu,
 * używając tagu "dodawanie" do ich wyboru.
 *
 * Ref. z instrukcji: "adnotacje @IncludeTags w JUnit6"
 *
 * Praktyczne zastosowanie:
 * - Pełna weryfikacja przypadku użycia "Dodanie filmu do oferty"
 * - Kompletny zestaw testów do uruchomienia przed release'em
 * - Zawiera zarówno testy jednostkowe jak i testy z mockowaniem
 */
@Suite
@SuiteDisplayName("Zestaw WSZYSTKICH testów - Dodanie filmu do oferty")
@SelectPackages({ "model", "controller" })
@IncludeTags("dodawanie")
public class SuiteDodawanieFilmuWszystkie {
    // Klasa zestawu testów - testy są wybierane przez @IncludeTags

    /*
     * Ten zestaw zawiera WSZYSTKIE testy dla przypadku użycia
     * "Dodanie filmu do oferty":
     *
     * ZADANIE 1 (bez mockowania) - oznaczone @Tag("dodawanie"):
     * - model.TestFilm - testy encji danych filmu
     * - model.TestDAO - testy warstwy dostępu do danych
     * - model.TestFabrykaStandardowegoFilmu - testy fabryki tworzenia filmów
     * - model.TestModelDodawanieFilmu - testy fasady Model
     * - controller.TestAdminControllerDodawanieFilmu - testy kontrolera admin
     *
     * ZADANIE 2 (z mockowaniem) - oznaczone @Tag("dodawanie") i @Tag("mock"):
     * - model.TestModelDodawanieFilmuMock - testy Model z @Mock IDAO
     * - controller.TestAdminControllerDodawanieFilmuMock - testy z @Mock IModel
     * - controller.TestDodanieNowegoFilmuMock - testy strategii z @Mock IModel
     *
     * Łącznie: 8 klas testowych, ~80+ testów
     */
}
```

4.2 Przypadek użycia: Przeglądanie repertuaru

Zestawy testów pozwalają na grupowanie testów według różnych kryteriów.

4.2.1 model.SuiteEncjiPrzegladanieRepertuaru

Plik: `src/test/java/model/SuiteEncjiPrzegladanieRepertuaru.java`

```java
package model;

import org.junit.platform.suite.api.*;

/**
 * Zestaw testów warstwy encji (model) dla przypadku użycia "Przeglądanie
 * repertuaru".
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Zadanie: 3 (zestawy testów)
 *
 * Ten zestaw uruchamia wszystkie testy z pakietu model oznaczone tagiem
 * "repertuar".
 *
 * Ref. z instrukcji: "zestaw testów klas warstwy encji, czyli znajdujących się
 * w jej pakiecie"
 *
 * Praktyczne zastosowanie: Weryfikacja poprawności wszystkich klas encji
 * zaangażowanych w proces przeglądania repertuaru (Seans, DAO, Model).
 */
@Suite
@SuiteDisplayName("Zestaw testów warstwy encji - Przeglądanie repertuaru")
@SelectPackages("model")
@IncludeTags("repertuar")
public class SuiteEncjiPrzegladanieRepertuaru {
    // Klasa zestawu testów - testy są wybierane automatycznie na podstawie
    // adnotacji

    /*
     * Ten zestaw zawiera następujące klasy testów:
     * - TestSeans.java - testy encji Seans
     * - TestDAOSeansy.java - testy operacji DAO związanych z seansami
     * - TestModelPobierzRepertuar.java - testy Model.pobierzRepertuar() bez
     * mockowania
     * - TestModelPobierzRepertuarMock.java - testy Model.pobierzRepertuar() z
     * mockowaniem
     *
     * Wszystkie powyższe klasy są oznaczone tagami @Tag("encja")
     * i @Tag("repertuar")
     */
}
```

4.2.2 controller.SuiteKontroliPrzegladanieRepertuaru

Plik: src/test/java/controller/SuiteKontroliPrzegladanieRepertuaru.java

```java
package controller;

import org.junit.platform.suite.api.*;

/**
 * Zestaw testów warstwy kontroli (controller) dla przypadku użycia
 * "Przeglądanie repertuaru".
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Zadanie: 3 (zestawy testów)
 *
 * Ten zestaw uruchamia wszystkie testy z pakietu controller oznaczone tagiem
 * "repertuar".
 *
 * Ref. z instrukcji: "zestaw testów klas warstwy kontroli, czyli znajdujących
 * się w jej pakiecie"
 *
 * Praktyczne zastosowanie: Weryfikacja poprawności kontrolerów klienckich
 * zaangażowanych w proces przeglądania repertuaru (ClientController).
 */
@Suite
@SuiteDisplayName("Zestaw testów warstwy kontroli - Przeglądanie repertuaru")
@SelectPackages("controller")
@IncludeTags("repertuar")
public class SuiteKontroliPrzegladanieRepertuaru {
    // Klasa zestawu testów - testy są wybierane automatycznie na podstawie
    // adnotacji

    /*
     * Ten zestaw zawiera następujące klasy testów:
     * - TestClientControllerPrzegladanieRepertuaru.java - testy kontrolera bez
     * mockowania
     * - TestClientControllerPrzegladanieRepertuaruMock.java - testy kontrolera z
     * mockowaniem
     *
     * Wszystkie powyższe klasy są oznaczone tagami @Tag("kontroler")
     * i @Tag("repertuar")
     */
}
```

4.2.3 suites.SuitePrzegladanieRepertuaruBezMock

Plik: src/test/java/suites/SuitePrzegladanieRepertuaruBezMock.java

```java
package suites;

import org.junit.platform.suite.api.*;

/**
 * Zestaw testów dla przypadku użycia "Przeglądanie repertuaru" BEZ mockowania.
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Zadanie: 3 (zestawy testów bazujące na tagach)
 *
 * Ten zestaw uruchamia testy oznaczone tagiem "repertuar", ale WYKLUCZA testy z
 * tagiem "mock".
 *
 * Ref. z instrukcji: "przynajmniej 2 zestawy testów oznaczonych wybranymi
 * tagami
 * i nie oznaczonych innymi wybranymi tagami, które mają jakieś praktyczne
 * zastosowanie"
 *
 * Praktyczne zastosowanie:
 * - Testy integracyjne bez mockowania - sprawdzają rzeczywistą współpracę
 * komponentów
 * - Szybka weryfikacja podstawowej funkcjonalności przeglądania repertuaru
 * - Przydatne do testów regresji przed wdrożeniem
 * - Uruchamiane gdy chcemy sprawdzić pełny przepływ bez izolacji zależności
 */
@Suite
@SuiteDisplayName("Zestaw testów przeglądania repertuaru - BEZ mockowania")
@SelectPackages({ "model", "controller" })
@IncludeTags("repertuar")
@ExcludeTags("mock")
public class SuitePrzegladanieRepertuaruBezMock {
    // Klasa zestawu testów - testy są wybierane automatycznie na podstawie
    // adnotacji

    /*
     * Ten zestaw zawiera następujące klasy testów (bez mockowania):
     *
     * Z pakietu model:
     * - TestSeans.java - testy encji Seans
     * - TestDAOSeansy.java - testy operacji DAO związanych z seansami
     * - TestModelPobierzRepertuar.java - testy Model.pobierzRepertuar() bez
     * mockowania
     *
     * Z pakietu controller:
     * - TestClientControllerPrzegladanieRepertuaru.java - testy
```

```
         kontrolera bez
46       * mockowania
47       *
48       * WYKLUCZONE (tag "mock"):
49       * - TestModelPobierzRepertuarMock.java
50       * - TestClientControllerPrzegladanieRepertuaruMock.java
51       */
52  }
```

4.2.4 suites.SuitePrzegladanieRepertuaruMock

Plik: src/test/java/suites/SuitePrzegladanieRepertuaruMock.java

```java
package suites;

import org.junit.platform.suite.api.*;

/**
 * Zestaw testów dla przypadku użycia "Przeglądanie repertuaru" TYLKO Z
 * mockowaniem.
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Zadanie: 3 (zestawy testów bazujące na tagach)
 *
 * Ten zestaw uruchamia TYLKO testy oznaczone tagami "repertuar" ORAZ
    "mock".
 *
 * Ref. z instrukcji: "przynajmniej 2 zestawy testów oznaczonych
    wybranymi
 * tagami
 * i nie oznaczonych innymi wybranymi tagami, które mają jakieś
    praktyczne
 * zastosowanie"
 *
 * Praktyczne zastosowanie:
 * - Szybkie testy jednostkowe z izolacją zależności
 * - Weryfikacja logiki biznesowej bez potrzeby rzeczywistych zależności
 * - Testy uruchamiane podczas CI/CD dla szybkiego feedbacku
 * - Przydatne gdy warstwa DAO lub inne zależności są niedostępne
 * - Testowanie obsługi błędów i edge cases (symulacja wyjątków)
 */
@Suite
@SuiteDisplayName("Zestaw testów przeglądania repertuaru - TYLKO z
   mockowaniem")
@SelectPackages({ "model", "controller" })
@IncludeTags({ "repertuar", "mock" })
public class SuitePrzegladanieRepertuaruMock {
    // Klasa zestawu testów - testy są wybierane automatycznie na
       podstawie
    // adnotacji

    /*
     * Ten zestaw zawiera następujące klasy testów (tylko z
        mockowaniem):
     *
     * Z pakietu model:
     * - TestModelPobierzRepertuarMock.java - testy Model z mockowanym
        IDAO
     * * Testuje: when/thenReturn, when/thenThrow, InOrder,
        ArgumentCaptor
     * * Weryfikuje: verify, times, never, atLeast, atMost
     *
     * Z pakietu controller:
     * - TestClientControllerPrzegladanieRepertuaruMock.java - testy z
        mockowanym
     * IModel
     *
     * WYKLUCZONE (brak tagu "mock"):
```

```
47        * - TestSeans.java
48        * - TestDAOSeansy.java
49        * - TestModelPobierzRepertuar.java
50        * - TestClientControllerPrzegladanieRepertuaru.java
51        */
52  }
```

4.2.5 suites.SuitePrzegladanieRepertuaruWszystkie

Plik: src/test/java/suites/SuitePrzegladanieRepertuaruWszystkie.java

```java
package suites;

import org.junit.platform.suite.api.*;

/**
 * Zestaw testów dla przypadku użycia "Przeglądanie repertuaru" - WSZYSTKIE
 * testy.
 *
 * Przypadek użycia: Przeglądanie repertuaru
 * Zadanie: 3 (zestawy testów)
 *
 * Ten zestaw uruchamia WSZYSTKIE testy związane z przeglądaniem repertuaru,
 * używając tagu "repertuar" do ich wyboru.
 *
 * Ref. z instrukcji: "adnotacje @IncludeTags w JUnit6"
 *
 * Praktyczne zastosowanie:
 * - Pełna weryfikacja przypadku użycia "Przeglądanie repertuaru"
 * - Kompletny zestaw testów do uruchomienia przed release'em
 * - Zawiera zarówno testy jednostkowe jak i testy z mockowaniem
 */
@Suite
@SuiteDisplayName("Zestaw WSZYSTKICH testów - Przeglądanie repertuaru")
@SelectPackages({ "model", "controller" })
@IncludeTags("repertuar")
public class SuitePrzegladanieRepertuaruWszystkie {
    // Klasa zestawu testów - testy są wybierane przez @IncludeTags

    /*
     * Ten zestaw zawiera WSZYSTKIE testy dla przypadku użycia
     * "Przeglądanie repertuaru":
     *
     * ZADANIE 1 (bez mockowania) - oznaczone @Tag("repertuar"):
     * - model.TestSeans - testy encji danych seansu
     * - model.TestDAOSeansy - testy warstwy dostępu do danych (operacje seansów)
     * - model.TestModelPobierzRepertuar - testy fasady Model
     * - controller.TestClientControllerPrzegladanieRepertuaru - testy kontrolera
     * klienta
     *
     * ZADANIE 2 (z mockowaniem) - oznaczone @Tag("repertuar") i @Tag("mock"):
     * - model.TestModelPobierzRepertuarMock - testy Model z @Mock IDAO
     * - controller.TestClientControllerPrzegladanieRepertuaruMock - testy z @Mock
     * IModel
     *
     * Łącznie: 6 klas testowych, ~90+ testów
     */
}
```

5 Podsumowanie

Sprawozdanie zawiera pełny kod testów jednostkowych dla dwóch przypadków użycia:

5.1 Przypadek użycia 1: Dodanie filmu do oferty

- **Zadanie 1**: Testy bez mockowania (5 klas testowych)
- **Zadanie 2**: Testy z mockowaniem Mockito (3 klasy testowe)
- **Zadanie 3**: Zestawy testów (5 klas zestawów)

5.2 Przypadek użycia 2: Przeglądanie repertuaru

- **Zadanie 1**: Testy bez mockowania (4 klasy testowe)
- **Zadanie 2**: Testy z mockowaniem Mockito (2 klasy testowe)
- **Zadanie 3**: Zestawy testów (5 klas zestawów)

Łącznie utworzono 24 klasy testowe zawierających ponad 170 testów jednostkowych.