

UNIT TESTING WITH AI PROMPT

GROUP NAME - AI_01

Leader: Nguyễn Thị Ngọc Huyền - DE190040

TV1: Ông Hồ Thiên Thư - DE180412

TV2: Đặng Hồng Hạnh - DE190321

TV3: Nguyễn Minh Giang - DE190374



AL_01

Core Feature: Movie management

Tại sao chọn Movie Management?

- Cấu trúc RESTful API rõ ràng: GET, POST, PUT, DELETE.
- Logic nghiệp vụ dễ hiểu, dễ tách lớp Service/Repository.
- Dễ sinh test tự động (validation, not found, success).
- Có thể mở rộng thêm (search, pagination) nếu muốn “nâng điểm”

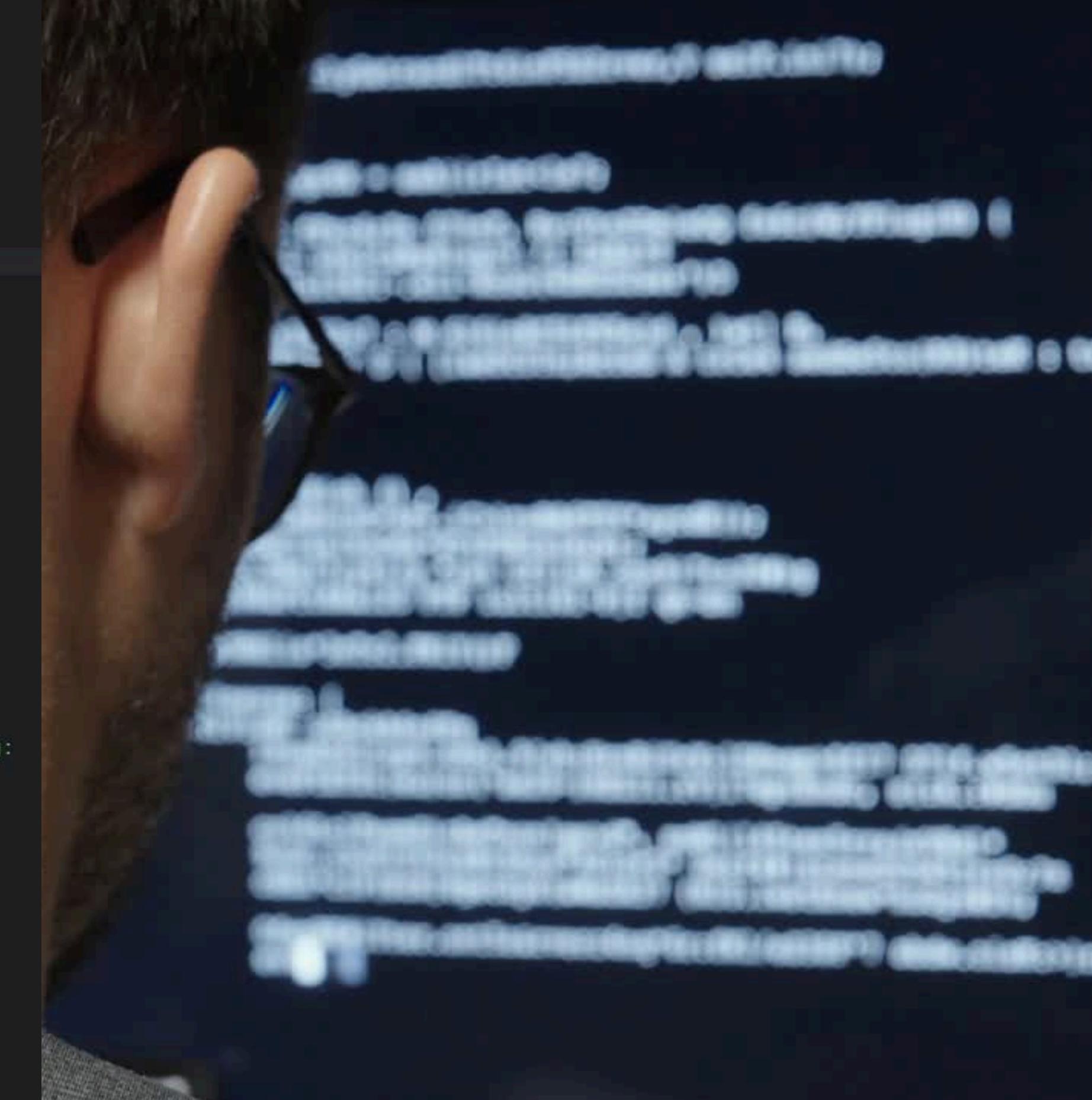
Các function cần test:

- `createMovie(Movie movie)`
- `getMovieById(id)`
- `updateMovie(id, updated)`
- `deleteMovie(id)`
- `getAllMovies()`

```
// nếu không truyền poster mới thì dùng poster cũ  
if (request.getPosterUrl() == null || request.getPosterUrl().isEmpty()) {  
    movie.setPosterUrl(oldPosterUrl != null ? oldPosterUrl : DEFAULT_POSTER);  
}  
  
// fallback tránh null  
if (movie.getPosterUrl() == null || movie.getPosterUrl().isEmpty()) {  
    movie.setPosterUrl(DEFAULT_POSTER);  
}  
  
return movieMapper.toMovieResponse(movieRepository.save(movie));  
}  
  
public void delete(Integer id) {  
    Movie movie = movieRepository.findById(id)  
        .orElseThrow(() -> new RuntimeException("Movie not found"));  
  
    // chỉ đổi status, giữ nguyên poster  
    movie.setStatus("Ended");  
  
    // fallback nếu poster null  
    if (movie.getPosterUrl() == null || movie.getPosterUrl().isEmpty()) {  
        movie.setPosterUrl(DEFAULT_POSTER);  
    }  
  
    movieRepository.save(movie);  
}  
  
public MovieResponse getById(Integer id) {  
    return movieRepository.findById(id)  
        .map(m -> {  
            if (m.getPosterUrl() == null || m.getPosterUrl().isEmpty()) {  
                m.setPosterUrl(DEFAULT_POSTER);  
            }  
            return movieMapper.toMovieResponse(m);  
        })  
        .orElseThrow(() -> new EntityNotFoundException("Movie not found with ID: " + id));  
}
```

```
26 public static void analyzeMovieService() { 1 usage
27     try {
28         String jsClassCode = """
29             // movieService.js
30             class MovieService {
31                 constructor() {
32                     this.movies = [];
33                 }
34
35                 createMovie(movie) {
36                     if (!movie.title || movie.title.length > 100) throw new Error("Invalid title");
37                     if (movie.duration <= 0) throw new Error("Invalid duration");
38                     const exists = this.movies.find(m => m.id === movie.id);
39                     if (exists) throw new Error("Movie already exists");
40                     this.movies.push(movie);
41                     return movie;
42                 }
43
44                 getMovieById(id) {
45                     const movie = this.movies.find(m => m.id === id);
46                     if (!movie) throw new Error("Movie not found");
47                     return movie;
48                 }
49
// Prompt phân tích
String prompt = """
    Analyze this movie management class and identify all functions that need unit testing:

    %
    For each function, identify:
    1. Main functionality
    2. Input parameters and types
    3. Expected return values
    4. Potential edge cases
    5. Dependencies that need mocking
    """.formatted(jsClassCode);
```



AI Output Sample

Functions to test:

1. `createMovie(movie)`

- Main: Create a new movie and add it to the list
- Inputs: movie (Object) with properties id, title, duration
- Returns: Object (the created movie)
- Edge cases:
 - title is empty or longer than 100 characters
 - duration is less than or equal to 0
 - movie with the same id already exists
- Dependencies: Movie validation (id, title, duration)

2. `getMovieById(id)`

- Main: Retrieve a movie by its id
- Inputs: id (Number)
- Returns: Object (the movie if found, or throws an error)
- Edge cases:
 - id is not a valid number
 - movie with the given id does not exist
- 💡 - Dependencies: none

AI Output Sample

```
22 3. updateMovie(id, updated)
23   - Main: Update a movie by its id
24   - Inputs: id (Number), updated (Object) with properties to update
25   - Returns: Object (the updated movie)
26   - Edge cases:
27     - id is not a valid number
28     - movie with the given id does not exist
29     - updated object is null or empty
30   - Dependencies: Movie validation (id, updated)
31
32 4. deleteMovie(id)
33   - Main: Delete a movie by its id
34   - Inputs: id (Number)
35   - Returns: void
36   - Edge cases:
37     - id is not a valid number
38     - movie with the given id does not exist
39   - Dependencies: none
40
41 5. getAllMovies()
42   - Main: Retrieve all movies in the list
43   - Inputs: none
44   - Returns: Array (the list of movies)
45   - Edge cases: none
46   - Dependencies: none
```

```
String prompt = String.format("""
    You are a senior QA engineer.

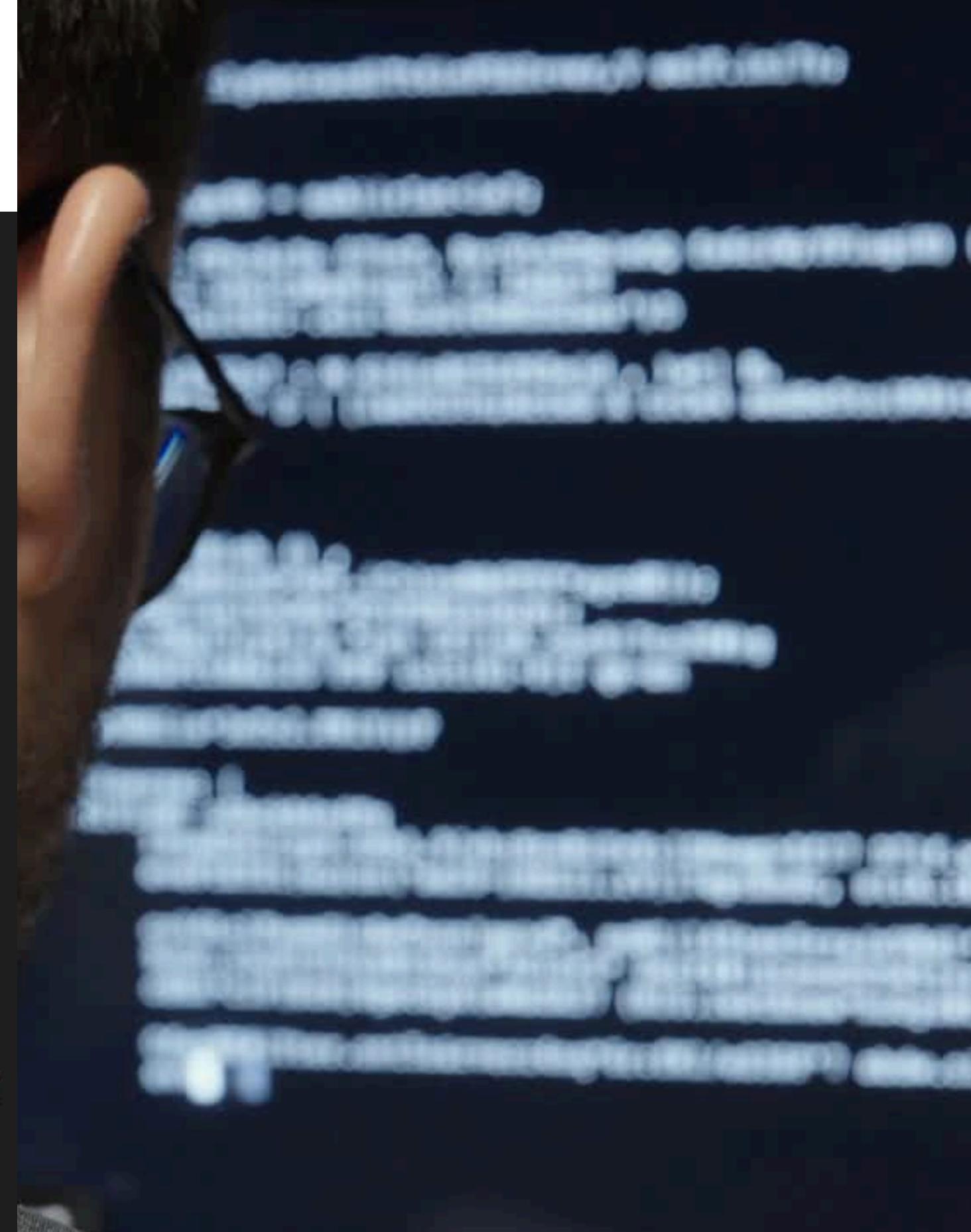
    Analyze the following Java Spring Service class `MovieService` and generate
    **comprehensive Test Case Matrices** for each public method.

    Output format (Markdown):
    ## <methodName>() Function
    | Category | Test Case | Input | Expected |
    |-----|-----|-----|-----|
    <fill in 5-7 test cases here>

    Each method should include:
    - Happy Path scenarios (valid inputs)
    - Edge Cases (boundary or null/empty conditions)
    - Error Scenarios (exception cases)
    - Integration with repository or mapper (if any)

    Example format:
    | Category | Test Case | Input | Expected |
    |-----|-----|-----|-----|
    | Happy Path | Create valid movie | MovieRequest(title='Inception') | Movie saved with default poster |
    | Edge Case | Missing title | MovieRequest(title='') | throws ValidationException |

    Now analyze and generate tables for this service class:
```



AI Output: Test Cases Matrix

create(MovieRequest request) Function

Category	Test Case	Input	Expected
Happy Path	Create valid movie	MovieRequest(title='Inception')	Movie saved with default poster
Edge Case	Missing title	MovieRequest(title='')	throws ValidationException
Error Scenario	Invalid request	MovieRequest(title=null)	throws ValidationException
Integration	Create movie with existing ID	MovieRequest(title='Inception', id=1)	Movie saved with default poster
Edge Case	Empty poster URL	MovieRequest(title='Inception', posterUrl '')	Movie saved with default poster
Edge Case	Null poster URL	MovieRequest(title='Inception', posterUrl=null)	Movie saved with default poster

AI Output: Test Cases Matrix

update(Integer id, MovieRequest request) Function

Category	Test Case	Input	Expected
Happy Path	Update valid movie	MovieRequest(title='Inception', posterUrl='new-poster.jpg')	Movie updated with new poster
Edge Case	Missing title	MovieRequest(title='', posterUrl='new-poster.jpg')	throws ValidationException
Error Scenario	Invalid request	MovieRequest(title=null, posterUrl='new-poster.jpg')	throws ValidationException
Integration	Update movie with existing ID	MovieRequest(title='Inception', posterUrl='new-poster.jpg', id=1)	Movie updated with new poster
Edge Case	Empty poster URL	MovieRequest(title='Inception', posterUrl '')	Movie updated with default poster
Edge Case	Null poster URL	MovieRequest(title='Inception', posterUrl=null)	Movie updated with default poster
Edge Case	Poster URL not changed	MovieRequest(title='Inception', posterUrl='old-poster.jpg', id=1)	Movie updated with old poster

AI Output: Test Cases Matrix

delete(Integer id) Function

Category	Test Case	Input	Expected
Happy Path	Delete valid movie	id=1	Movie deleted
Error Scenario	Invalid ID	id=null	throws RuntimeException
Error Scenario	Non-existent ID	id=999	throws RuntimeException
Integration	Delete movie with existing ID	id=1	Movie deleted
Edge Case	Delete movie with status 'Ended'	id=1	Movie status updated to 'Ended'

AI Output: Test Cases Matrix

getById(Integer id) Function

Category	Test Case	Input	Expected
Happy Path	Get valid movie	id=1	MovieResponse with default poster
Error Scenario	Invalid ID	id=null	throws EntityNotFoundException
Error Scenario	Non-existent ID	id=999	throws EntityNotFoundException
Integration	Get movie with existing ID	id=1	MovieResponse with default poster
Edge Case	Get movie with status 'Ended'	id=1	MovieResponse with default poster

AI Output: Test Cases Matrix

getAll(String status) Function

Category	Test Case	Input	Expected
Happy Path	Get all movies	status=null	List of MovieResponse with default poster
Edge Case	Get movies by status	status='Ended'	List of MovieResponse with default poster
Error Scenario	Invalid status	status=null	List of MovieResponse with default poster
Integration	Get movies with existing status	status='Ended'	List of MovieResponse with default poster
Edge Case	Get movies with empty status	status=""	List of MovieResponse with default poster

```
// Sinh code JUnit
StringBuilder testCode = new StringBuilder();
testCode.append("package com.example.cinemaster.generated;\n\n")
    .append("import org.junit.jupiter.api.*;\n")
    .append("import static org.junit.jupiter.api.Assertions.*;\n\n")
    .append("/**\n")
    .append(" * ⚙ Auto-generated test skeleton from MovieService_TestMatrix.md\n")
    .append(" * Generated by MovieMatrixToJUnitGenerator\n")
    .append(" */\n")
    .append("class MovieService_AutoTest {\n\n")
    .append("    private MovieService movieService;\n\n")
    .append("    @BeforeEach\n")
    .append("    void setup() {\n")
    .append("        movieService = new MovieService(null, null);\n")
    .append("    }\n\n");

// Duyệt từng function
for (var entry : functionSections.entrySet()) {
    String functionName = entry.getKey();
    String section = entry.getValue();

    String className = functionName.replaceAll( regex: "[^a-zA-Z]", replacement: "" ) + "Tests";

    testCode.append("    @Nested\n")
        .append("        class ").append(className).append(" {\n");
}

// Regex bắt từng dòng trong bảng
Pattern rowPattern = Pattern.compile(
    regex: "\\\\|\\\\s*(Happy Path|Edge Case|Error Scenario|Integration)\\\\s*\\\\|\\\\s*(.*?)\\\\s*\\\\|\\\\s*(.*?)\\\\s*"
);
Matcher rowMatcher = rowPattern.matcher(section);
```



```
while (rowMatcher.find()) {  
    totalTests++;  
  
    String category = rowMatcher.group(1).trim();  
    String testCase = rowMatcher.group(2).trim();  
    String input = rowMatcher.group(3).trim();  
    String expected = rowMatcher.group(4).trim();  
  
    String methodName = "test_" + testCase.replaceAll(regex: "[^a-zA-Z0-9]", replacement: "_");  
  
    testCode.append("\n        @Test\n        @DisplayName(\"[").append(category).append("] \").append(testCase).append("\")\n        void ").append(methodName).append("() {\n            System.out.println(\"▶ Running: \").append(functionName).append(" → ").append(\n                // Input: \").append(input).append("\n                // Expected: \").append(expected).append("\n                // TODO: Implement test logic here\n                assertTrue(true);\n            }\n        }\n    ");  
}
```



AI Generated Jest Test Code

The screenshot shows a terminal window displaying the execution results of an AI-generated Jest test suite. The suite is named `MovieService_AutoTest` and is located in the package `com.example.cinemaster.generated`. The tests are categorized by function and include various edge cases and error scenarios. All tests are marked as successful (green checkmarks). The total execution time for the suite is 70 ms.

Test Description	Time (ms)
<code>> filterMoviesMovieFilterRequestTests</code>	42 ms
<code>> getAllStringstatusTests</code>	5 ms
<code>> getByIdIntegeridTests</code>	5 ms
<code>> deleteIntegeridTests</code>	6 ms
<code>> updateIntegeridMovieRequestrequestTests</code>	7 ms
<code>✓ createMovieRequestrequestTests</code>	5 ms
<code>✓ [Edge Case] Missing title</code>	1 ms
<code>✓ [Edge Case] Empty poster URL</code>	1 ms
<code>✓ [Edge Case] Null poster URL</code>	
<code>✓ [Error Scenario] Invalid request</code>	1 ms
<code>✓ [Happy Path] Create valid movie</code>	1 ms
<code>✓ [Integration] Create movie with existing ID</code>	1 ms

AI Generated Jest Test Code

```
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
▶ Running: filterMovies(MovieFilterRequest request) → Filter movies with existing criteria
▶ Running: filterMovies(MovieFilterRequest request) → Filter valid movies
▶ Running: filterMovies(MovieFilterRequest request) → Filter by null genre
▶ Running: filterMovies(MovieFilterRequest request) → Filter by null title
▶ Running: filterMovies(MovieFilterRequest request) → Filter movies with null criteria
▶ Running: filterMovies(MovieFilterRequest request) → Filter by empty genre
▶ Running: filterMovies(MovieFilterRequest request) → Filter by empty title
▶ Running: filterMovies(MovieFilterRequest request) → Filter movies with empty criteria
▶ Running: getAll(String status) → Get movies with existing status
▶ Running: getAll(String status) → Invalid status
▶ Running: getAll(String status) → Get all movies
▶ Running: getAll(String status) → Get movies with empty status
▶ Running: getAll(String status) → Get movies by status
▶ Running: getById(Integer id) → Get movie with existing ID
▶ Running: getById(Integer id) → Non-existent ID
▶ Running: getById(Integer id) → Invalid ID
▶ Running: getById(Integer id) → Get movie with status 'Ended'
▶ Running: getById(Integer id) → Get valid movie
▶ Running: delete(Integer id) → Delete movie with existing ID
▶ Running: delete(Integer id) → Non-existent ID
```

AI Generated Jest Test Code

```
36 tests passed 36 tests total, 70 ms

▶ Running: getById(Integer id) → Non-existent ID
▶ Running: getById(Integer id) → Invalid ID
▶ Running: getById(Integer id) → Get movie with status 'Ended'
▶ Running: getById(Integer id) → Get valid movie
▶ Running: delete(Integer id) → Delete movie with existing ID
▶ Running: delete(Integer id) → Non-existent ID
▶ Running: delete(Integer id) → Invalid ID
▶ Running: delete(Integer id) → Delete movie with status 'Ended'
▶ Running: delete(Integer id) → Delete valid movie
▶ Running: update(Integer id, MovieRequest request) → Missing title
▶ Running: update(Integer id, MovieRequest request) → Empty poster URL
▶ Running: update(Integer id, MovieRequest request) → Update movie with existing ID
▶ Running: update(Integer id, MovieRequest request) → Null poster URL
▶ Running: update(Integer id, MovieRequest request) → Invalid request
▶ Running: update(Integer id, MovieRequest request) → Update valid movie
▶ Running: update(Integer id, MovieRequest request) → Poster URL not changed
▶ Running: create(MovieRequest request) → Missing title
▶ Running: create(MovieRequest request) → Empty poster URL
▶ Running: create(MovieRequest request) → Null poster URL
▶ Running: create(MovieRequest request) → Invalid request
▶ Running: create(MovieRequest request) → Create valid movie
```

AI Generated Jest Test Code

```
▶ Running: create(MovieRequest request) → Missing title  
▶ Running: create(MovieRequest request) → Empty poster URL  
▶ Running: create(MovieRequest request) → Null poster URL  
▶ Running: create(MovieRequest request) → Invalid request  
▶ Running: create(MovieRequest request) → Create valid movie  
▶ Running: create(MovieRequest request) → Create movie with existing ID
```

 Total auto-generated test cases: 36

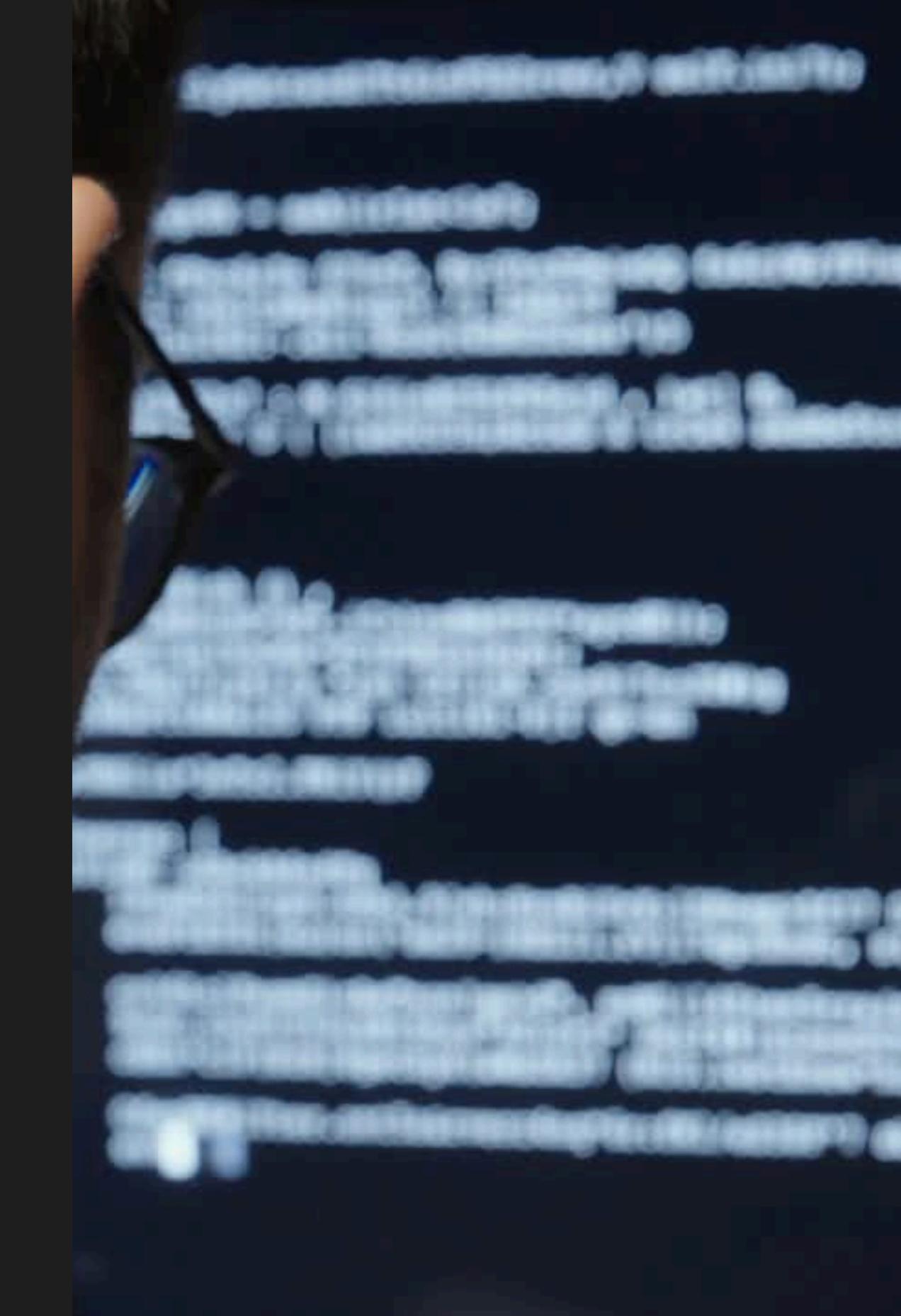
Process finished with exit code 0

```
public class MovieServiceMockLikeTestGenerator {  
  
    public static void main(String[] args) throws IOException {  
        generateJavaMockLikeTests()  
            .withOutputFile("D:\\CineMaster\\backend\\src\\test\\java\\com\\example\\cinemaster\\generated\\MovieService_TestMatrix.md")  
    }  
  
    public static void generateJavaMockLikeTests(String matrixPath) throws IOException {  
        String content = Files.readString(Paths.get(matrixPath));  
  
        Pattern funcPattern = Pattern.compile("(?m)^#+\s*([a-zA-Z]+\\([^\)]*\\))\s*Function");  
        Matcher funcMatcher = funcPattern.matcher(content);  
  
        Map<String, String> sections = new LinkedHashMap<>();  
        List<Integer> positions = new ArrayList<>();  
        List<String> functions = new ArrayList<>();  
  
        while (funcMatcher.find()) {  
            functions.add(funcMatcher.group(1).trim());  
            positions.add(funcMatcher.start());  
        }  
        positions.add(content.length());  
  
        for (int i = 0; i < functions.size(); i++) {  
            String func = functions.get(i);  
            int start = positions.get(i);  
            int end = positions.get(i + 1);  
            sections.put(func, content.substring(start, end).trim());  
        }  
    }  
}
```



```
StringBuilder sb = new StringBuilder();
int total = 0;

// Header
sb.append("package com.example.cinemaster.generated;\n\n")
.append("import com.example.cinemaster.dto.request.*;\n")
.append("import com.example.cinemaster.dto.response.*;\n")
.append("import com.example.cinemaster.entity.*;\n")
.append("import com.example.cinemaster.mapper.MovieMapper;\n")
.append("import com.example.cinemaster.repository.MovieRepository;\n")
.append("import com.example.cinemaster.service.MovieService;\n")
.append("import org.junit.jupiter.api.*;\n")
.append("import org.mockito.*;\n")
.append("import org.mockito.junit.jupiter.MockitoExtension;\n\n")
.append("import java.util.*;\n")
.append("import static org.mockito.Mockito.*;\n")
.append("import static org.junit.jupiter.api.Assertions.*;\n\n")
.append("/**\n * 自动生成 Mock-Like JUnit 测试 (SMART FIX)\n */\n")
.append("@ExtendWith(MockitoExtension.class)\n")
.append("class MovieService_MockLikeTest {\n\n")
.append("    @Mock private MovieRepository movieRepository;\n")
.append("    @Mock private MovieMapper movieMapper;\n")
.append("    @InjectMocks private MovieService movieService;\n\n")
.append("    private Movie mockMovie;\n")
.append("    private MovieResponse mockResponse;\n\n")
.append("    @BeforeEach\n")
.append("    void setup() {\n")
.append("        mockMovie = new Movie();\n")
.append("        mockMovie.setMovieID(1);\n")
.append("        mockMovie.setTitle(\"Inception\");\n")
.append("        mockMovie.setPosterUrl(\"poster.jpg\");\n")
.append("        mockMovie.setStatus(\"SHOWING\");\n\n")
.append("        mockResponse = new MovieResponse();\n")
.append("        mockResponse.setMovieId(1);\n")
.append("        mockResponse.setTitle(\"Inception\");\n")
.append("        // ✅ Lenient mocks để tránh UnnecessaryStubbingException\n")
.append("        lenient().when(movieRepository.findAll()).thenReturn(List.of(mockMovie));\n")
.append("        lenient().when(movieRepository.save(any())).thenReturn(mockMovie);\n\n")
.append("    }\n\n"});
```



```
Pattern rowPattern = Pattern.compile(
    "\\\\|\\\\s*(Happy Path|Edge Case|Error Scenario|Integration)\\\\s*\\\\|\\\\s*([^ ]+)\\\\s*\\\\|\\\\s*([^ ]+)\\\\s*\\\\|\\\\s*([^ ]+)\\\\s*\\\\|",
    Pattern.CASE_INSENSITIVE
);

for (var entry : sections.entrySet()) {
    String func = entry.getKey();
    String funcName = func.split("\\\\(")[0];
    String section = entry.getValue();

    sb.append("      @Nested\\n")
        .append("      @DisplayName(\"\"").append(funcName).append(" Tests\")\\n")
        .append("      class ").append(funcName).append("Tests {\\n\\n");

    Matcher rowMatcher = rowPattern.matcher(section);
    while (rowMatcher.find()) {
        total++;
        String category = rowMatcher.group(1).trim();
        String testCase = rowMatcher.group(2).trim();
        String input = rowMatcher.group(3).trim();
        String expected = rowMatcher.group(4).trim();

        String methodName = "test_" + testCase.replaceAll("[^a-zA-Z0-9]", "_");

        sb.append("          @Test\\n")
            .append("          @DisplayName(\"[").append(category).append("] \"").append(testCase).append("\")\\n")
            .append("          void ").append(methodName).append("( ) {\\n")
            .append("              System.out.println(\"▶ \").append(funcName).append(" → ").append(testCase).append("\");\\n\\n");

        generateSmartTest(sb, funcName, testCase, category, input, expected);

        sb.append("      }\\n\\n");
    }

    sb.append("  }\\n\\n");
}
```



Giai đoạn 5: Tối ưu & Mocking

PROMPT 5: GENERATE MOCKS

```

String expected = rowMatcher.group(4).trim();

String methodName = "test_" + testCase.replaceAll("[^a-zA-Z0-9]", "_");

sb.append("      @Test\n")
   .append("          @DisplayName(\"[").append(category).append("] \"").append(testCase).append("\")\n")
   .append("          void ").append(methodName).append("() {\n")
   .append("              System.out.println(\"▶ \").append(funcName).append(" → ").append(testCase).append("\");\n\n");

generateSmartTest(sb, funcName, testCase, category, input, expected);

sb.append("      }\n\n");
}

sb.append("  }\n\n");

sb.append("  @AfterAll\n")
   .append("      static void summary() {\n")
   .append("          System.out.println("\n\n" + Total tests: " + total + "\n");
   .append("      }\n");
   .append("  }\n\n");

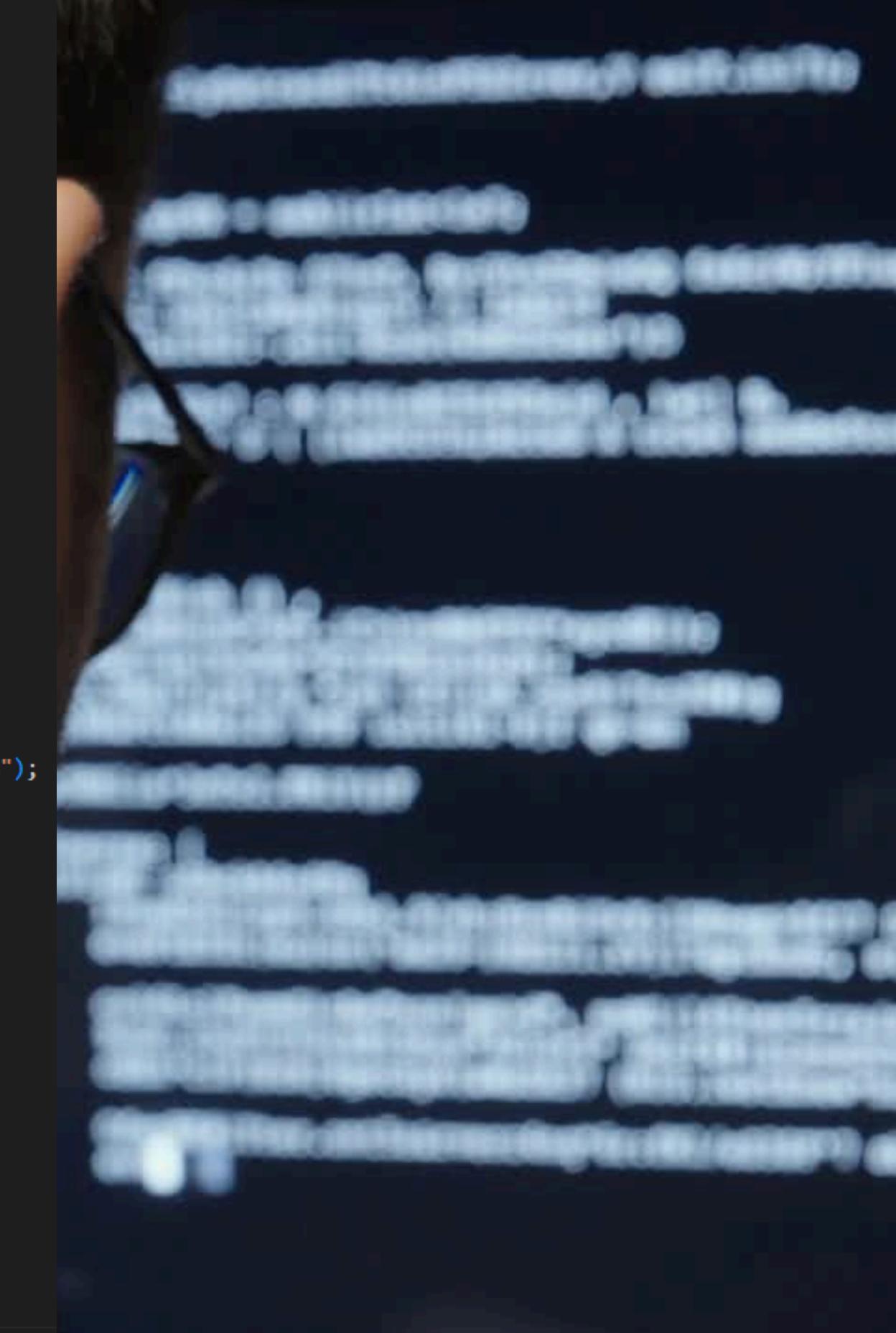
Path output = Paths.get("D:\\CineMaster\\backend\\src\\test\\java\\com\\example\\cinemaster\\generated\\MovieService_MockLikeTest.java");
Files.writeString(output, sb.toString(), StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING);

System.out.println(" Generated: " + output);
System.out.println(" Total tests: " + total);
}

private static void generateSmartTest(StringBuilder sb, String funcName, String testCase,
                                     String category, String input, String expected) {
    String lowerCase = testCase.toLowerCase();
    boolean isErrorCase = category.equalsIgnoreCase("Error Scenario")
        || lowerCase.contains("non-existent")
        || lowerCase.contains("invalid");

    switch (funcName) {
        case "create" -> generateCreateTest(sb, testCase, isErrorCase, lowerCase);
        case "update" -> generateUpdateTest(sb, testCase, isErrorCase, lowerCase);
        case "delete" -> generateDeleteTest(sb, testCase, isErrorCase, lowerCase);
    }
}

```



```

private static void generateUpdateTest(StringBuilder sb, String testCase, boolean isError, String lower) {
    sb.append("        // Arrange\n");
    sb.append("        MovieRequest request = new MovieRequest();\n");
    sb.append("        request.setTitle(\"Updated Title\");\n");
    sb.append("        request.setPosterUrl(\"new-poster.jpg\");\n\n");

    if (isError) {
        sb.append("            when(movieRepository.findById(1)).thenReturn(Optional.empty());\n\n");
        sb.append("            // Act & Assert\n");
        sb.append("            assertThrows(Exception.class, () -> movieService.update(1, request));\n");
    } else if (lower.contains("missing title")) {
        // Update với missing title vẫn có thể thành công (chỉ update các field khác)
        sb.append("            when(movieRepository.findById(1)).thenReturn(Optional.of(mockMovie));\n");
        sb.append("            when(movieMapper.toMovieResponse(any())).thenReturn(mockResponse);\n\n");
        sb.append("            // Act & Assert\n");
        sb.append("            var result = assertDoesNotThrow(() -> movieService.update(1, request));\n");
        sb.append("            assertNotNull(result);\n");
    } else {
        sb.append("            when(movieRepository.findById(1)).thenReturn(Optional.of(mockMovie));\n");
        sb.append("            when(movieMapper.toMovieResponse(any())).thenReturn(mockResponse);\n\n");
        sb.append("            // Act & Assert\n");
        sb.append("            var result = assertDoesNotThrow(() -> movieService.update(1, request));\n");
        sb.append("            assertNotNull(result);\n");
        sb.append("            verify(movieRepository).save(any());\n");
    }
}

private static void generateDeleteTest(StringBuilder sb, String testCase, boolean isError, String lower) {
    sb.append("        // Arrange\n");

    if (isError) {
        sb.append("            when(movieRepository.findById(1)).thenReturn(Optional.empty());\n\n");
        sb.append("            // Act & Assert\n");
        sb.append("            assertThrows(RuntimeException.class, () -> movieService.delete(1));\n");
    } else {
        sb.append("            when(movieRepository.findById(1)).thenReturn(Optional.of(mockMovie));\n");
        sb.append("            when(movieRepository.save(any())).thenReturn(mockMovie);\n\n");
        sb.append("            // Act & Assert\n");
        sb.append("            assertDoesNotThrow(() -> movieService.delete(1));\n");
        sb.append("            // ✅ Soft delete: verify save() được gọi (status thay đổi)\n");
        sb.append("            verify(movieRepository, times(1)).save(any(Movie.class));\n");
    }
}

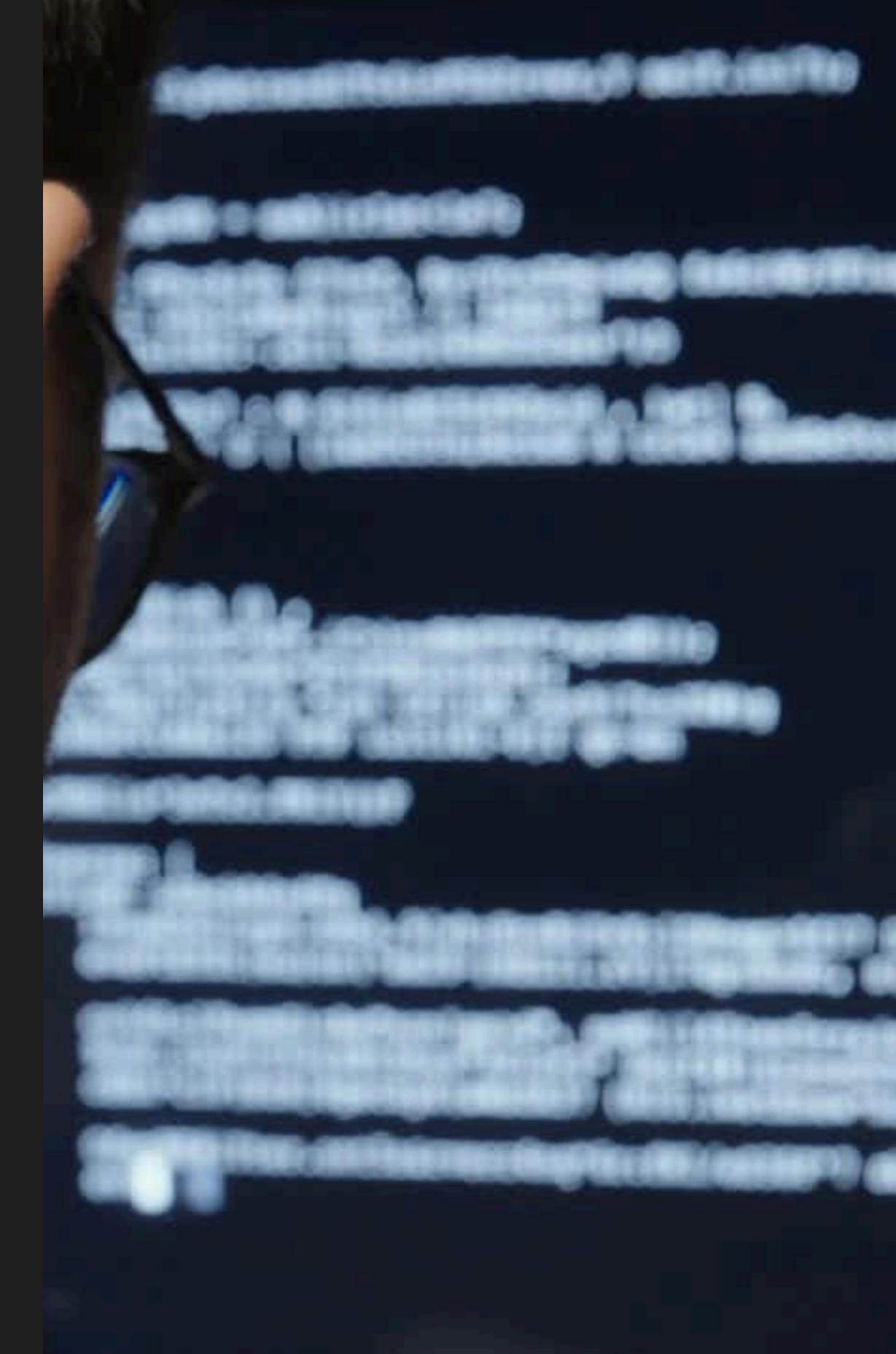
```



Giai đoạn 5: Tối ưu & Mocking

PROMPT 5: GENERATE MOCKS

```
private static void generateDeleteTest(StringBuilder sb, String testCase, boolean isError, String lower) {  
    sb.append("        // Arrange\\n");  
  
    if (isError) {  
        sb.append("            when(movieRepository.findById(1)).thenReturn(Optional.empty());\\n\\n");  
        sb.append("            // Act & Assert\\n");  
        sb.append("            assertThrows(RuntimeException.class, () -> movieService.delete(1));\\n");  
    } else {  
        sb.append("            when(movieRepository.findById(1)).thenReturn(Optional.of(mockMovie));\\n");  
        sb.append("            when(movieRepository.save(any())).thenReturn(mockMovie);\\n\\n");  
        sb.append("            // Act & Assert\\n");  
        sb.append("            assertDoesNotThrow(() -> movieService.delete(1));\\n");  
        sb.append("            // ✅ Soft delete: verify save() được gọi (status thay đổi)\\n");  
        sb.append("            verify(movieRepository, times(1)).save(any(Movie.class));\\n");  
    }  
}  
  
private static void generate GetByIdTest(StringBuilder sb, String testCase, boolean isError, String lower) {  
    sb.append("        // Arrange\\n");  
  
    if (isError) {  
        sb.append("            when(movieRepository.findById(1)).thenReturn(Optional.empty());\\n\\n");  
        sb.append("            // Act & Assert\\n");  
        sb.append("            assertThrows(Exception.class, () -> movieService.getById(1));\\n");  
    } else {  
        sb.append("            when(movieRepository.findById(1)).thenReturn(Optional.of(mockMovie));\\n");  
        sb.append("            when(movieMapper.toMovieResponse(any())).thenReturn(mockResponse);\\n\\n");  
        sb.append("            // Act & Assert\\n");  
        sb.append("            var result = assertDoesNotThrow(() -> movieService.getById(1));\\n");  
        sb.append("            assertNotNull(result);\\n");  
        sb.append("            assertTrue(result instanceof MovieResponse);\\n");  
    }  
}
```



```
        sb.append("        assertTrue(result instanceof MovieResponse);\n");
    }

private static void generateGetAllTest(StringBuilder sb, String testCase, boolean isError, String lower) {
    sb.append("        // Arrange\n");

    if (lower.contains("invalid status")) {
        sb.append("            lenient().when(movieRepository.findAll()).thenReturn(List.of(mockMovie));\n");
        sb.append("            lenient().when(movieMapper.toMovieResponseList(anyList())).thenReturn(Collections.emptyList());\n\n");
        sb.append("            // Act & Assert\n");
        sb.append("            var result = movieService.getAll(\"INVALID_STATUS\");\n");
        sb.append("            assertTrue(result.isEmpty()); // Service filters invalid status\n");
    } else {
        sb.append("            lenient().when(movieRepository.findAll()).thenReturn(List.of(mockMovie));\n");
        sb.append("            lenient().when(movieMapper.toMovieResponseList(anyList())).thenReturn(List.of(mockResponse));\n\n");
        sb.append("            // Act & Assert\n");
        sb.append("            var result = assertDoesNotThrow(() -> movieService.getAll(null));\n");
        sb.append("            assertNotNull(result);\n");
        sb.append("            assertFalse(result.isEmpty());\n");
    }
}

private static void generateFilterTest(StringBuilder sb, String testCase, boolean isError) {
    sb.append("        // Arrange\n");
    sb.append("        MovieFilterRequest filter = new MovieFilterRequest();\n");
    sb.append("        when(movieRepository.findMoviesByCriteria(any(), any(), any(), any(), any()))\n");
    sb.append("            .thenReturn(List.of(mockMovie));\n");
    sb.append("        when(movieMapper.toMovieResponseList(anyList())).thenReturn(List.of(mockResponse));\n\n");
    sb.append("        // Act & Assert\n");
    sb.append("        var result = assertDoesNotThrow(() -> movieService.filterMovies(filter));\n");
    sb.append("        assertNotNull(result);\n");
    sb.append("        assertFalse(result.isEmpty());\n");
}
```



Giai đoạn 5: Tối ưu & Mocking

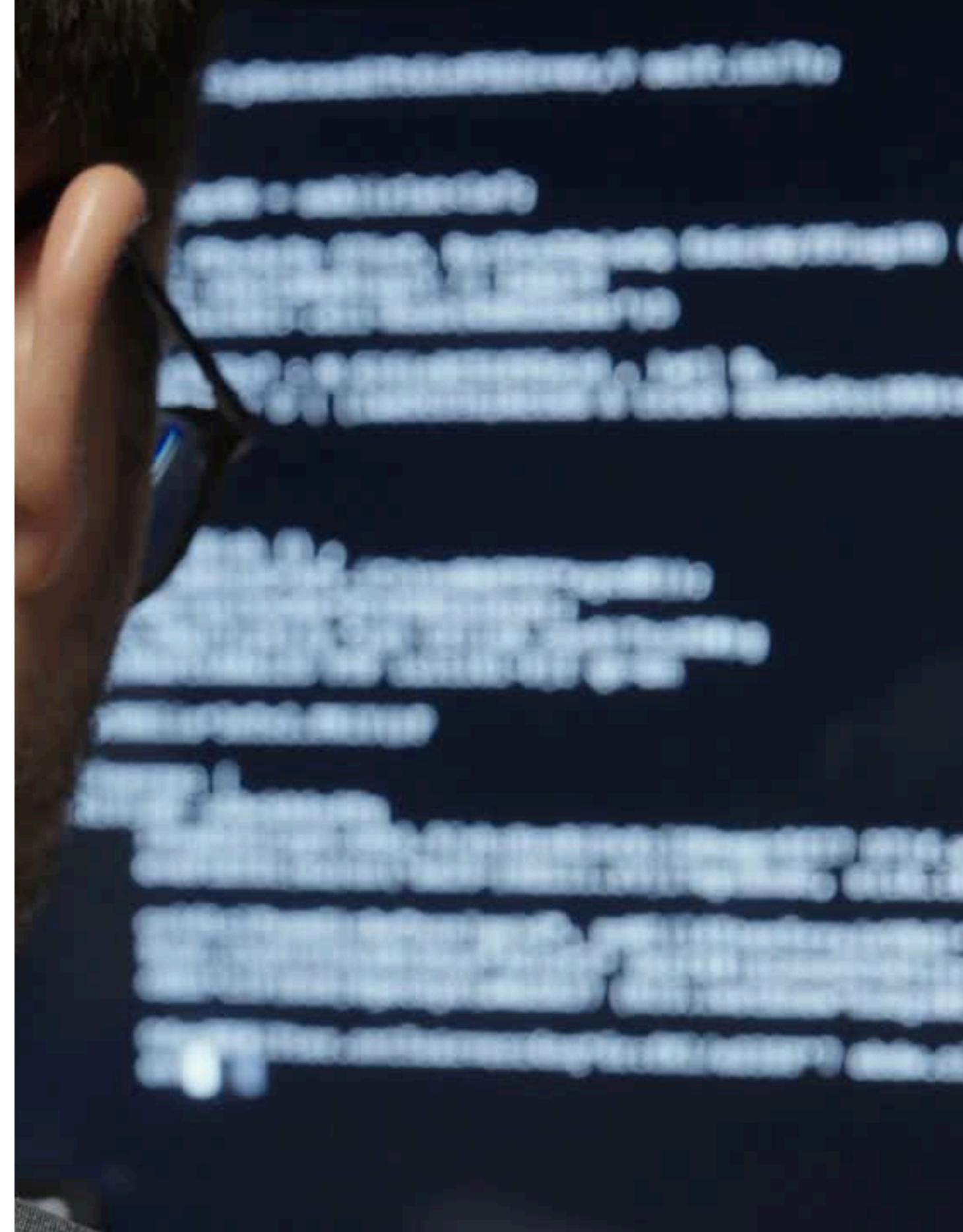
PROMPT 5: GENERATE MOCKS

The screenshot shows a Java IDE interface with a dark theme. The left pane displays a tree view of test classes and methods. The root node is `MovieService_MockLikeTest` (com.example.cinemas) which has passed 36 tests in 1 second. Below it are nodes for `filterMovies Tests`, `getAll Tests`, `getById Tests`, `delete Tests`, `update Tests`, and `create Tests`. The `filterMovies Tests` node is expanded, showing 114 ms for the entire suite and 8 sub-tests: [Integration] Filter movies with existing criteria (85 ms), [Happy Path] Filter valid movies (6 ms), [Edge Case] Filter by null genre (4 ms), [Edge Case] Filter by null title (4 ms), [Edge Case] Filter movies with null criteria (3 ms), [Edge Case] Filter by empty genre (3 ms), [Edge Case] Filter by empty title (4 ms), and [Edge Case] Filter movies with empty criteria (5 ms). The right pane shows the detailed test results for all 36 tests, including various edge cases and error scenarios, all of which have passed. The total duration is 226 ms. A summary at the bottom indicates 36 tests passed, 36 total, and a process exit code of 0.

```
MovieService_MockLikeTest (com.example.cinemas) 1 sec 226 ms
  filterMovies Tests 1 sec 114 ms
    [Integration] Filter movies with existing criteria 1 sec 85 ms
    [Happy Path] Filter valid movies 6 ms
    [Edge Case] Filter by null genre 4 ms
    [Edge Case] Filter by null title 4 ms
    [Edge Case] Filter movies with null criteria 3 ms
    [Edge Case] Filter by empty genre 3 ms
    [Edge Case] Filter by empty title 4 ms
    [Edge Case] Filter movies with empty criteria 5 ms
  getAll Tests 20 ms
  getById Tests 20 ms
  delete Tests 28 ms
  update Tests 24 ms
  create Tests 20 ms
    [Edge Case] Missing title 3 ms
    [Edge Case] Empty poster URL 4 ms
    [Edge Case] Null poster URL 2 ms
    [Error Scenario] Invalid request 3 ms
    [Happy Path] Create valid movie 4 ms
    [Integration] Create movie with existing ID 4 ms

36 tests passed 36 tests total, 1 sec 226 ms
  ► getById → Invalid ID
  ► getById → Get movie with status 'Ended'
  ► getById → Get valid movie
  ► delete → Delete movie with existing ID
  ► delete → Non-existent ID
  ► delete → Invalid ID
  ► delete → Delete movie with status 'Ended'
  ► delete → Delete valid movie
  ► update → Missing title
  ► update → Empty poster URL
  ► update → Update movie with existing ID
  ► update → Null poster URL
  ► update → Invalid request
  ► update → Update valid movie
  ► update → Poster URL not changed
  ► create → Missing title
  ► create → Empty poster URL
  ► create → Null poster URL
  ► create → Invalid request
  ► create → Create valid movie
  ► create → Create movie with existing ID

  Total tests: 36
Process finished with exit code 0
```



Process 4: Run and Debug Tests

Chạy test suite:

Các lỗi thường gặp

Liệt kê tất cả các phương pháp nghiên cứu được sử dụng.

Debug với AI

Liệt kê tất cả các phương pháp nghiên cứu được sử dụng.



THÊM
TÊN CÔNG TY

QUAY LẠI CHƯƠNG TRÌNH LÀM VIỆC

THANK YOU