**Software Quality Engineering**

**Test Plan**

**Course Instructor**

Madam Uzma Mahar

**Group Members**

Abdullah Daoud……………………………………………………….(22I-2626)

Usman Ali……………………………………………………………(22I-2725)

Faizan Rasheed……………………………………………………..(22I-2734)

**Section**

SE-E

**Date**

**Wednesday, December 11, 2024**

**Fall 2024**



**Department of Software Engineering**

FAST – National University of Computer & Emerging Sciences

Islamabad Campus

Table of Contents

[1. Test Plan Identifier 1](#_Toc184825043)

[2. Introduction 1](#_Toc184825044)

[(i). References 1](#_Toc184825045)

[(ii). Scope 1](#_Toc184825046)

[3. Test Items 2](#_Toc184825047)

[(i). Modules: 2](#_Toc184825048)

[(a). Movie Management: 2](#_Toc184825049)

[(b). User Authentication and Profiles: 2](#_Toc184825050)

[(c). Review Management: 2](#_Toc184825051)

[(d). Recommendation Engine: 2](#_Toc184825052)

[(e). Subscription Plans and Payments: 2](#_Toc184825053)

[(f). Notifications: 2](#_Toc184825054)

[(g). Watch History: 2](#_Toc184825055)

[(ii). Configuration Requirements: 2](#_Toc184825056)

[(a). Version Control: 2](#_Toc184825057)

[(b). Critical Elements Schedule: 2](#_Toc184825058)

[4. Features to be Tested 3](#_Toc184825059)

[(i). Features: 3](#_Toc184825060)

[(a). Movies 3](#_Toc184825061)

[(b). Users 3](#_Toc184825062)

[(c). Reviews 3](#_Toc184825063)

[(d). Recommendations 3](#_Toc184825064)

[(e). Subscriptions 3](#_Toc184825065)

[(f). Notifications 3](#_Toc184825066)

[(g). Watch History 3](#_Toc184825067)

[(ii). Justification of Risk Levels: 3](#_Toc184825068)

[5. Features NOT to be Tested 4](#_Toc184825069)

[(i). Database Scalability: 4](#_Toc184825070)

[(ii). Third-party Library Bugs: 4](#_Toc184825071)

[(iii). Justifications: 4](#_Toc184825072)

[(iv). Impact on Risk: 4](#_Toc184825073)

[6. Test Approach 5](#_Toc184825074)

[(i). Types of Testing: 5](#_Toc184825075)

[(a). Functional Testing: 5](#_Toc184825076)

[(b). Integration Testing: 5](#_Toc184825077)

[(c). Performance Testing: 5](#_Toc184825078)

[(d). Security Testing: 6](#_Toc184825079)

[(e). User Interface Testing: 6](#_Toc184825080)

[(f). Regression Testing: 6](#_Toc184825081)

[(ii). Tools and Methods: 7](#_Toc184825082)

[(a). Jest and Supertest: 7](#_Toc184825083)

[(b). Postman: 7](#_Toc184825084)

[(c). Load Testing Tools (e.g., JMeter): 7](#_Toc184825085)

[(d). Static Code Analysis Tools: 7](#_Toc184825086)

[(iii). Configuration Management: 7](#_Toc184825087)

[(iv). Metrics to be Collected: 7](#_Toc184825088)

[(v). Hardware and Software Configurations: 7](#_Toc184825089)

[(a). Hardware: 7](#_Toc184825090)

[(b). Software 8](#_Toc184825091)

[(vi). Testing Environments: 8](#_Toc184825092)

[(vii). Special Requirements: 8](#_Toc184825093)

[(viii). Meetings and Processes: 8](#_Toc184825094)

[7. Item Pass/Fail Criteria 9](#_Toc184825095)

[(i). General Completion Criteria: 9](#_Toc184825096)

[(a). Unit Test Level: 9](#_Toc184825097)

[(b). Integration Test Level: 9](#_Toc184825098)

[(c). System Test Level: 9](#_Toc184825099)

[(d). Master Test Plan Level: 10](#_Toc184825100)

[(ii). Defect Severity Guidelines: 10](#_Toc184825101)

[(iii). Metrics for Pass/Fail Evaluation: 10](#_Toc184825102)

[(a). Test Case Execution: 10](#_Toc184825103)

[(b). Defect Thresholds: 10](#_Toc184825104)

[(c). Performance and Load Testing: 11](#_Toc184825105)

[(d). Code Quality: 11](#_Toc184825106)

[(iv). Additional Criteria: 11](#_Toc184825107)

[(a). Regression Testing: 11](#_Toc184825108)

[(b). Documentation: 11](#_Toc184825109)

[(c). Stakeholder Sign-Off: 11](#_Toc184825110)

[8. Testing Deliverables 11](#_Toc184825111)

[9. Testing Environment 11](#_Toc184825112)

[10. Roles and Responsibilities 12](#_Toc184825113)

[(i). Test Manager 12](#_Toc184825114)

[(ii). Test Engineers 12](#_Toc184825115)

[(iii). Developers 13](#_Toc184825116)

[(iv). Decision Making and Training: 13](#_Toc184825117)

[11. Risks and Contingencies 14](#_Toc184825118)

[(i). Late Delivery of Features 14](#_Toc184825119)

[(ii). Unavailability of Test Data 14](#_Toc184825120)

[(iii). Dependency on External APIs (e.g., Stripe) 14](#_Toc184825121)

[(iv). Lack of Personnel Resources 14](#_Toc184825122)

[(v). Delays in Training 15](#_Toc184825123)

[(vi). Changes to the Original Requirements or Designs 15](#_Toc184825124)

[(vii). Delays in Availability of Hardware or Software 15](#_Toc184825125)

[(viii). Performance and Load Testing Failures 15](#_Toc184825126)

[(ix). Incomplete Regression Testing 15](#_Toc184825127)

[(x). Unforeseen Technical Issues or Bugs 16](#_Toc184825128)

# 1. Test Plan Identifier

This is the initial version of the test plan for the Movie Management System (MMS-TP-001) project, aligned with the IEEE 829 format. Revisions will be identified incrementally (e.g., V1.1, V2.0).

# 2. Introduction

The purpose of this test plan is to define the testing approach for the Movie Management System (MMS). MMS is a MERN stack project designed for managing movies, users, subscriptions, and reviews. This is a master-level test plan, providing a comprehensive strategy for all testing phases, including functional, integration, performance, and security testing.

## (i). References

This test plan references the following document:

* **IEEE 829 Test Plan Format:** Used as the standard for this document.

## (ii). Scope

The scope of this test plan includes:

* Validation of all functional modules such as movie management, user authentication, reviews, recommendations, subscriptions, notifications, and watch history.
* Ensuring system performance under load conditions.
* Verifying data security and user privacy.
* Testing UI responsiveness across browsers and devices.

This plan is aligned with the overall software project plan and includes resource and budget constraints. Testing will support related evaluation activities, such as code reviews and design analysis, ensuring seamless coordination with development and change control processes.

The plan will serve as a foundation for communication and coordination among the development, testing, and management teams, ensuring timely identification and resolution of risks and issues.

# 3. Test Items

The following modules, features, and configurations will be tested to ensure functionality, integration, and performance:

## (i). Modules:

### (a). Movie Management:

* Adding, editing, deleting, filtering, and retrieving movies.
* Streaming functionality (if applicable).

### (b). User Authentication and Profiles:

* User registration, login, and logout processes.
* Profile updates and management.

### (c). Review Management:

* Creating, updating, and deleting reviews.
* Visualizing reviews through pie and bar charts.

### (d). Recommendation Engine:

* Generating recommendations based on user activities, genres, and ratings.

### (e). Subscription Plans and Payments:

* Adding and managing subscription plans.
* Processing payments and managing user subscriptions.

### (f). Notifications:

* Creating, retrieving, and deleting notifications for users.

### (g). Watch History:

* Adding, viewing, and deleting movies from a user's watch history.

## (ii). Configuration Requirements:

### (a). Version Control:

* Backend: Node.js (v16.x or later) with MongoDB Atlas.
* Frontend: React.js (v18.x or later).

### (b). Critical Elements Schedule:

* Ensure timely delivery of API endpoints for testing.
* Completion of UI components for movie and review management.

These items represent the features and functionalities to be delivered to the client, aligned with the software's configuration management plan and project timeline. Testing will focus on modules, ensuring compatibility and adherence to client requirements.

# 4. Features to be Tested

The features to be tested are described from the User’s perspective and are categorized based on their functional importance. Each feature is assigned a risk level (H: High, M: Medium, L: Low) to prioritize testing efforts.

## (i). Features:

### (a). Movies

* Add, edit, delete, filter, and retrieve movies (H).
* Streaming functionality (H).

### (b). Users

* Registration, login, and logout processes (H).
* Profile updates and wishlist management (M).
* Activity-based recommendations (H).

### (c). Reviews

* Creating, updating, and deleting reviews (M).
* Visualization through pie/bar graphs (L).

### (d). Recommendations

* Generate recommendations based on user ratings, activity, and movie similarity (H).

### (e). Subscriptions

* Payment processing and plan creation (H).
* User subscription management (M).

### (f). Notifications

* Create, retrieve, and delete notifications (L).

### (g). Watch History

* Add, view, and delete movies from a user’s watch history (M).

## (ii). Justification of Risk Levels:

* **High (H):** Core functionalities essential for system operations (e.g., movies, user authentication, and recommendations).
* **Medium (M):** Features that enhance user experience but are not critical for system functioning (e.g., profiles, subscriptions).
* **Low (L):** Supporting functionalities that do not directly affect primary workflows (e.g., notifications, review visualizations).

This categorization ensures that critical features are tested thoroughly while maintaining efficient use of resources for lower-priority areas.

# 5. Features NOT to be Tested

The following features will not be tested, based on scope, risk level, or dependencies:

## (i). Database Scalability:

Testing the scalability of the database is outside the scope of this release and is assumed to perform well with the current dataset.

## (ii). Third-party Library Bugs:

Bugs in external libraries such as stripe, nodemailer, or cloudinary are excluded from testing as they are independently maintained and tested by their respective providers.

## (iii). Justifications:

* Not included in this release:
* Database scalability testing will be addressed in a future phase if required.
* Low risk and stable:
* The third-party libraries used in the project are well-established and have a low likelihood of introducing issues.
* Not functional for this version:
* These features do not directly impact the core functionalities being delivered to the user in this version.

## (iv). Impact on Risk:

The exclusion of these features does not pose a significant risk to the project, as the core functionalities remain unaffected. Proper planning and contingencies will address any potential impact of these exclusions in future iterations if required.

# 6. Test Approach

The **test approach** outlines the overall strategy to ensure the Movie Management System (MMS) is thoroughly tested. It covers all key aspects of the system, such as functionality, integration, performance, security, and user experience, ensuring the final product meets high standards of quality.

## (i). Types of Testing:

### (a). Functional Testing:

**Purpose:** Ensure the core functionalities of the system work as intended.

**Scope:** Verify critical features, including:

* Movie management (CRUD operations).
* User authentication (login, sign-up, password recovery).
* Subscription management (plans, payments).
* Movie recommendations (based on user preferences, history).

**Method:** Manual and automated tests will be executed to verify the expected behavior against defined use cases.

### (b). Integration Testing:

**Purpose:** Confirm that different system components interact correctly.

**Scope:**

* Verify interactions between modules, such as movies and reviews, subscriptions and notifications.
* Ensure data flows seamlessly between modules without data loss or miscommunication.

**Method:** Tests will be executed on the interactions between microservices or different layers (frontend, backend, database).

### (c). Performance Testing:

**Purpose:** Evaluate how the system performs under various loads and stress conditions.

**Scope:**

* Test the response times of key actions (e.g., user login, adding a movie, etc.).
* Measure server load capacity to handle expected and peak traffic.
* Ensure the system maintains performance with multiple concurrent users.

**Method:** Load and stress tests will be conducted using tools like JMeter to simulate user behavior and system load.

### (d). Security Testing:

**Purpose:** Identify and mitigate security vulnerabilities in the system.

**Scope:**

* Test user authentication and authorization mechanisms (e.g., role-based access).
* Verify data protection mechanisms (e.g., encryption for sensitive information).
* Check for potential vulnerabilities like SQL injection, cross-site scripting (XSS), and other common security issues.

**Method:** Automated tools and manual penetration tests will be used to identify weaknesses and ensure compliance with security standards.

### (e). User Interface Testing:

**Purpose:** Ensure the user interface is functional, responsive, and user-friendly.

**Scope:**

* Validate that the application is responsive across various screen sizes and devices (desktop, tablet, mobile).
* Check for cross-browser compatibility (Chrome, Firefox, Edge).
* Ensure that UI elements (buttons, forms, navigation) function as expected.

**Method:** Manual testing and automated UI testing tools like Selenium will be employed to verify the visual and interactive elements of the application.

### (f). Regression Testing:

**Purpose:** Ensure that newly implemented features or bug fixes do not introduce new issues.

**Scope:**

* Test fixes for previously identified defects.
* Revalidate existing functionality to ensure that no previously working features are broken after code changes.

**Method:** Automated regression test suites will be run to verify system stability after every major update or bug fix.

## (ii). Tools and Methods:

### (a). Jest and Supertest:

Jest will be used for unit testing individual components, ensuring that each function works as expected. Supertest will be employed for testing API endpoints, ensuring they return the correct responses for various request scenarios.

### (b). Postman:

Postman will be used for manual API testing. It will help testers verify the correct behavior of the backend API endpoints, including checking response statuses, headers, and response data.

### (c). Load Testing Tools (e.g., JMeter):

Tools like JMeter will be used for simulating high traffic and stress testing. These tools allow the team to test how the system performs under different load conditions, ensuring that the server can handle large numbers of concurrent users.

### (d). Static Code Analysis Tools:

Tools such as SonarQube or ESLint will be used to identify potential vulnerabilities, coding inefficiencies, or non-compliance with coding standards. These tools help ensure that the codebase remains clean and secure.

## (iii). Configuration Management:

* **Version Control:** All code and test artifacts will be managed using GitHub to ensure proper version control and collaboration.
* **Environment Management:** All test configurations (e.g., database connections, environment variables) will be documented and stored in configuration files to guarantee reproducibility and consistency across testing environments.

## (iv). Metrics to be Collected:

* **Test Coverage Percentage:** Measure the percentage of the codebase covered by automated tests to ensure comprehensive testing.
* **Number and Severity of Defects Identified:** Track the number of defects discovered during testing and categorize them based on severity (critical, major, minor).
* **Average Response Times during Performance Testing:** Monitor the average response times under load to identify performance bottlenecks.
* **Number of Regression Defects Detected:** Record any defects that were reintroduced during regression testing, which could indicate issues with new code.

## (v). Hardware and Software Configurations:

### (a). Hardware:

Development and testing servers will be equipped with at least 8 GB of RAM and quad-core processors to ensure the system is capable of handling the expected load during testing phases.

### (b). Software

Node.js will be used for backend development, MongoDB Atlas will serve as the cloud database, and React.js will be used for frontend development.

## (vi). Testing Environments:

* **Staging Environment:** A staging environment will closely replicate the production system, enabling testing in an environment that mimics real-world conditions.
* **Production-like Environment:** This will simulate a production setup with identical server configurations to validate the system's performance and behavior under production conditions.

## (vii). Special Requirements:

* **Testers Training:** Testers will be required to undergo training on the use of tools like Jest, Postman, and JMeter. This training will ensure that they are familiar with the tools’ functionalities and can use them efficiently.
* **Internet Connectivity:** Since the application uses cloud-based services like MongoDB Atlas, testers will require a stable internet connection for connecting to cloud resources during testing.

## (viii). Meetings and Processes:

* **Biweekly Meetings:** Regular progress meetings will be held every two weeks to track the status of testing efforts, identify blockers, and discuss next steps.
* **Emergency Meetings:** Critical issues that need immediate attention will be addressed through emergency meetings to prioritize fixes and ensure that testing resumes without delays.
* **Defect Triage Meetings:** These meetings will prioritize identified defects, determine their severity, and assign resources to fix critical bugs in a timely manner.

This test strategy aims to provide a comprehensive and structured approach to testing, ensuring that the Movie Management System is thoroughly validated, meets user expectations, and is delivered without major defects.

# 7. Item Pass/Fail Criteria

The pass/fail criteria define the conditions under which the testing effort can be deemed complete and the system can be considered ready for deployment. These criteria are structured to ensure that the system meets all functional, performance, security, and usability requirements.

## (i). General Completion Criteria:

### (a). Unit Test Level:

* All test cases defined for individual components or functions must be executed.
* Code coverage must meet or exceed 90%, ensuring critical paths are fully tested.
* Minor defects (e.g., UI alignment issues, spelling errors) may be accepted if they do not disrupt primary functionality or workflows.
* Static code analysis must show no critical vulnerabilities in the codebase.

### (b). Integration Test Level:

* All integrated modules must function cohesively with no interface mismatches or data loss.
* Data flow between modules (e.g., user actions affecting movie recommendations, subscriptions, and notifications) must be validated for correctness and consistency.
* Any critical or major integration issues must be resolved before proceeding to system-level testing.
* Error-handling mechanisms must be validated to ensure graceful recovery from failures.

### (c). System Test Level:

* The system must satisfy all functional requirements, including core features like movie management, user authentication, subscription processing, and recommendations.
* Response times for API calls must be below 2 seconds under a load of 100 concurrent users.
* Stress tests must show no server crashes, excessive memory usage, or instability during 24-hour peak load simulations.
* Authentication and authorization workflows must prevent unauthorized access to secure endpoints.
* No sensitive data (e.g., passwords, payment information) should be stored in plain text or logged inadvertently.
* Security vulnerabilities (e.g., SQL injection, XSS) must be remediated.
* User Interface (UI) tests must ensure responsiveness and usability across multiple devices and browsers.

### (d). Master Test Plan Level:

* All lower-level test plans (unit, integration, and system) must be completed, with results documented and signed off.
* End-to-end workflows (e.g., user registration, subscription payment, watching a movie) must pass 100% of test cases without failures.
* No critical or major defects should remain unresolved.
* A maximum of 5 minor or cosmetic defects can remain, provided they do not affect core functionality, user experience, or deployment readiness.

## (ii). Defect Severity Guidelines:

* **Critical Defects:** These represent a complete failure of core functionality (e.g., system crashes, data corruption). None are acceptable for system release.
* **Major Defects:** These represent failures in key features (e.g., payment not processed, notifications not delivered). All must be resolved before release.
* **Minor Defects:** These are less severe (e.g., minor UI misalignment, slow non-critical features). Up to 5 may remain unresolved.
* **Cosmetic Defects:** These include minor visual issues or spelling errors that do not affect usability. Unresolved cosmetic issues are acceptable unless they significantly degrade the user experience.

## (iii). Metrics for Pass/Fail Evaluation:

### (a). Test Case Execution:

* At least 95% of all defined test cases must pass.
* 100% of critical and major test cases must pass.

### (b). Defect Thresholds:

* Critical defects: 0 allowed.
* Major defects: ≤ 2 allowed.
* Minor defects: ≤ 5 allowed.
* Cosmetic defects: No strict limit but should be tracked and triaged.

### (c). Performance and Load Testing:

* API response times ≤ 2 seconds for 100 concurrent users.
* Error rate ≤ 1% under peak load.

### (d). Code Quality:

* Code coverage ≥ 90% with tools like Jest and Supertest.
* No unresolved critical vulnerabilities in static code analysis.

## (iv). Additional Criteria:

### (a). Regression Testing:

Any changes introduced after defect resolution must not impact previously tested functionality.

### (b). Documentation:

All test cases, defects, and test results must be documented and reviewed for completeness and accuracy.

### (c). Stakeholder Sign-Off:

The release must be approved by project stakeholders, including the test manager, development lead, and product owner.

By defining these detailed criteria, we ensure that the Movie Management System meets quality, reliability, and user expectations before it is released to production.

# 8. Testing Deliverables

* Test Plan Document (this document).
* Test Cases and Test Scripts.
* Test Execution Reports.
* Defect Logs and Reports.
* Final Summary Report (including Pass/Fail statistics).

# 9. Testing Environment

* Backend: Node.js with Express, MongoDB Atlas.
* Frontend: React.js (assumed based on MERN stack).
* Test Server: Local or cloud-hosted server.
* Test Data: Sample users, movies, and subscriptions.

# 10. Roles and Responsibilities

This section defines the roles and responsibilities of the key personnel involved in the testing process for the Movie Management System (MMS). Each individual is assigned specific duties to ensure effective and efficient execution of the testing phases, from planning to defect resolution.

## (i). Test Manager

The Test Manager is responsible for overseeing the entire testing process and ensuring that all required elements for testing are in place. Their key duties include:

* **Coordinating Testing Efforts:** Ensure that testing is aligned with the overall project schedule, and manage resource allocation for testing activities.
* **Risk Management:** Identify potential risks associated with testing and implement strategies to mitigate them. This includes assessing the impact of any potential issues on the project's schedule and resources.
* **Feature Selection:** Oversee the selection of features to be tested, ensuring that high-risk features are prioritized.
* **Scheduling:** Resolve scheduling conflicts, especially when testing on production systems, and ensure that all test activities are completed on time.
* **Training and Resource Management:** Ensure that the testing team is well-trained in the tools, processes, and methodologies required for the project. This includes organizing or providing training sessions where needed.
* **Decision Making:** Make critical go/no-go decisions for items not covered by test plans and ensure that all testing efforts are completed to meet the project’s quality standards.
* **Reporting and Communication:** Prepare test reports, track progress, and communicate testing status to stakeholders, including project managers, developers, and product owners.

## (ii). Test Engineers

Test Engineers are responsible for executing the tests as outlined in the test plan and reporting defects. Their duties include:

* **Writing Test Cases:** Develop detailed test cases based on the requirements and specifications provided for the Movie Management System.
* **Test Execution:** Run manual and automated tests, documenting results accurately.
* **Bug Reporting:** Identify defects and log them in the tracking system, providing necessary details for developers to address the issues.
* **Collaboration:** Work closely with the developers to clarify defects, verify fixes, and re-test corrected functionality.
* **Test Automation:** Where applicable, write automated test scripts to increase test coverage and ensure repeatable test execution.

## (iii). Developers

Developers play a key role in ensuring the system’s functionality meets the expected quality standards by addressing defects and supporting the testing process. Their responsibilities include:

* **Bug Fixing:** Upon receiving defect reports, developers are responsible for investigating and resolving the issues in the codebase. They must ensure that the system meets the requirements for functionality, performance, and security.
* **Providing Test Builds:** Developers provide the necessary builds of the system to the testing team for validation, ensuring that the builds are stable and contain the necessary fixes for testing.
* **Collaboration with Test Engineers:** Work closely with Test Engineers to understand defects and help resolve issues efficiently. Developers may be required to run their own tests to reproduce issues and validate fixes.
* **Code Review and Quality Assurance:** Collaborate in code reviews to ensure that the code meets quality standards and adheres to the project's coding guidelines.

## (iv). Decision Making and Training:

* **Go/No-Go Decisions:** The Test Manager, in collaboration with key stakeholders, makes critical decisions on whether to proceed with the testing or release phases based on the test results. This includes determining the state of the system’s readiness for deployment, especially when certain items fall outside the scope of the initial test plans.
* **Training Requirements:** Testers, including Test Engineers and developers, will be provided training on testing tools (e.g., Jest, Postman) and methodologies (e.g., Agile testing practices) to ensure a high standard of testing.

By clearly defining these roles and responsibilities, the testing process will be structured and well-coordinated, minimizing risks and ensuring that the system is ready for production.

# 11. Risks and Contingencies

The Movie Management System (MMS) testing process may encounter various risks that could impact the successful execution and timely delivery of the system. Identifying and planning for these risks is essential to mitigate their effects and ensure that testing continues smoothly. This section outlines the potential risks to the testing process and provides contingency strategies to address each identified risk.

## (i). Late Delivery of Features

**Risk:** Late delivery of essential features can delay the testing process, particularly when core functionalities are incomplete or unavailable for testing. This may result in testing being rushed or missing critical components.

**Contingency:** In the event that features are delayed, the testing team will prioritize critical modules that impact the overall system's functionality. High-risk features, such as user authentication, movie management, and payment processing, will be tested first to ensure that these core components are functioning as expected before proceeding with less critical functionalities.

## (ii). Unavailability of Test Data

**Risk:** Access to realistic and comprehensive test data is crucial for validating the system's functionality. If test data is unavailable or incomplete, testing may be hindered, leading to delays or incomplete test coverage.

**Contingency:** If required test data is not available, synthetic data will be generated to simulate real-world conditions. This data will mirror the structure and characteristics of the actual data expected in the system, ensuring that testing can proceed without delays.

## (iii). Dependency on External APIs (e.g., Stripe)

**Risk:** External dependencies, such as third-party services like Stripe for payment processing, may face availability issues, downtime, or changes in functionality, which could impact the testing process.

**Contingency:** To mitigate the risks posed by dependencies on external APIs, mock API responses will be used in the testing environment. This allows the testing of related functionalities, such as payment processing, without relying on the availability or performance of the actual third-party services. These mocks will be designed to simulate realistic responses and edge cases, ensuring comprehensive testing of integration points.

## (iv). Lack of Personnel Resources

**Risk:** A shortage of skilled testing personnel or a mismatch between resource availability and the project timeline may cause delays or impact the quality of testing efforts.

**Contingency:** To address potential personnel shortages, additional testers may be recruited, or existing resources may be reallocated to prioritize critical testing activities. If additional resources are not available, the test team may work overtime or adjust the scope of testing to focus on the most essential features. However, this may have an impact on team morale, and overtime work will be managed to avoid burnout.

## (v). Delays in Training

**Risk:** Testing tools and techniques require proper training. If testers are not trained on new tools or the application, it could cause delays or reduce the effectiveness of the testing process.

**Contingency:** The test manager will ensure that training sessions are scheduled well in advance of the testing phase, with buffers to accommodate any changes in tool usage. In case of delays in training, the testing schedule will be adjusted to ensure that testers have the necessary skills and knowledge before they begin executing tests.

## (vi). Changes to the Original Requirements or Designs

**Risk:** Changes in the project’s requirements or design during the development or testing phases could introduce scope creep, requiring adjustments to test plans, test cases, and schedules.

**Contingency:** Any changes to the requirements or designs will trigger a review of the test plan and schedule. If changes occur after the requirements freeze date, the testing team will assess the impact on testing and may reduce the number of tests, increase the number of acceptable defects, or adjust the test scope to meet the new requirements. The team will collaborate closely with developers and stakeholders to ensure that the test cases are updated to reflect the changes.

## (vii). Delays in Availability of Hardware or Software

**Risk:** Testing environments may depend on specific hardware or software configurations that could be delayed or unavailable, affecting the ability to conduct tests on time.

**Contingency:** In case of hardware or software unavailability, the testing team will work with available resources or simulate environments to continue testing. If necessary, resources will be procured or additional testing environments will be set up to ensure continuity.

## (viii). Performance and Load Testing Failures

**Risk:** During performance testing, the system may fail to meet required response times or handle the expected load, which could delay the release schedule.

**Contingency:** If performance or load testing shows that the system cannot handle the expected user load, the development team will optimize the system and re-test after addressing the issues. If performance issues are not resolved within the set timeline, the number of users or acceptable response time thresholds may be adjusted based on project priorities.

## (ix). Incomplete Regression Testing

**Risk:** Regression testing is critical to ensure that new code changes do not introduce defects in existing functionality. If regression testing is incomplete or missed, critical bugs may go undetected.

**Contingency:** Regression testing will be prioritized as part of the continuous integration process. The test manager will ensure that all critical functionalities are covered by automated regression tests, and additional manual regression tests will be executed if time permits.

## (x). Unforeseen Technical Issues or Bugs

**Risk:** During testing, unforeseen technical issues or bugs in the test environment may arise, leading to delays or errors in test execution.

**Contingency:** The testing team will have a dedicated support system for resolving technical issues quickly. In case of prolonged issues, the testing schedule will be adjusted to account for the delay, and priority will be given to critical tests that need to be completed before the release.

By identifying these risks and establishing clear contingency plans, the testing process can remain flexible and resilient, ensuring the Movie Management System meets its quality and delivery goals despite unforeseen challenges.