Table of Contents

1. Introduction 2
  1.1. Scope 2
    1.1.1 Features In-Scope 2
    1.1.2 Features Not Tested 5
    1.1.3 Features Out of Scope 5
  1.2. API Endpoint Explanation 6
  1.3. Quality Objective 12
  1.4. Roles and Responsibilities 12
2. Testing Strategy 13
  2.1. Overview 13
  2.2. Test Levels 13
  2.2. Test Types 14
  2.3. Bug Triage 15
    2.3.1. Severity 15
    2.3.2. Priority 15
  2.4. Suspension Criteria and Resumption Requirements 15
  2.5. Test Completeness 16
3. Test Case Design Techniques 17
  3.1 Equivalence Partitioning (EP) 17
  3.2. Negative Testing 17
  3.3. State Transition 18
4. Testing Artifacts 19
5. Resources and Environment Needs 19
6. Test Execution 19
7. Defect Management 20
8. Reporting and Metrics 21
9. Test Schedule 21
10. Risks and Mitigation 22
  10.1. Risks Associated with the Testing Phase 22
  10.2. Mitigation Strategies for identified Risks 22

1. Introduction

This test plan outlines my approach and strategy for testing the Note API documented in Swagger. The purpose of this document is to ensure that all functionalities are thoroughly validated, providing confidence in the API's reliability and security.

I will use Postman as my primary tool for API testing, which will allow me to efficiently execute test cases and validate the functionality of each endpoint. Additionally, I will utilize Newman Reporter Generator to execute my Postman collection and log the results in HTML format, as well as JMeter for load testing.

The API includes three main sections: Check Health, Users, and Notes. With Postman's intuitive interface, I can easily check responses, status codes, identify errors, and ensure that each section operates as expected. My goal is to ensure that the API meets all specified requirements and performs reliably under expected load conditions.

1.1. Scope

1.1.1 Features In-Scope

The scope of this test plan includes the following aspects of API testing using Postman:

- Testing the functionality of all endpoints within the API sections (Health, User, Note). This includes verifying that each endpoint returns the expected responses like status codes, response time, response body etc.
- Testing the API's ability to handle various input types, including valid, invalid, and equivalent partitioning, to ensure robust validation mechanisms are in place and achieve high test coverage.
- Validating that the API correctly enforces security measures, ensuring that only authorized users can access certain endpoints.
- Verifying that data is correctly created, updated, and deleted across the Users and Notes sections, ensuring that operations do not lead to data corruption or loss.
- Assessing the API's performance under expected load conditions. Each API endpoint must respond within a range of **50 to 1000 milliseconds**, depending on its complexity and functionality. This will include load testing to simulate a minimum of **50 concurrent users** and measure response times effectively.

Breakdown of Last point (Load Testing): Due to the lack of required documentation, this estimate is generated based on assumptions for all the API endpoints.

ENDPOINT	**ESTIMATED AVERAGE RESPONSE TIME WITH 50 VUS (MS)**
Health check	50 – 150 ms
Register user	300 – 700 ms
Login user	300 – 700 ms
Get user profile	150 – 400 ms
Forget password	300 – 700 ms
Update user profile	300 – 500 ms
Logout	50 - 200 ms
Delete account	300 - 600 ms
Create note	300 – 600 ms
Get all Notes	300 - 1000 ms (depending on the number of notes)
Get the note by ID	150 - 400 ms
Update an existing note	300 – 600 ms
Update the completed status of note	300 - 600 ms
Delete note by ID	300 - 600 ms

Brief Explanation

Health Check:

- Ideal Response Time: 50 – 150 ms
- This endpoint should respond almost instantaneously as it merely checks the status of the service.

Register User:

- Ideal Response Time: 300 – 700 ms
- This involves creating a new user, which may include validation and database interactions.

Login User:

- Ideal Response Time: 300 – 700 ms
- Similar to registration, this requires authentication checks and possibly database queries.

Get User Profile:

- Ideal Response Time: 150 – 400 ms
- Fetching user data typically involves querying the database.

Update User Profile:

- Ideal Response Time: 300 – 500 ms
- Updating user data requires validation and database updates.

Forget Password:

- Ideal Response Time: 300 – 700 ms
- This may involve sending an email or other verification processes.

Change Password:

- Ideal Response Time: 300 – 700 ms
- Similar to reset password but usually involves fewer steps if already authenticated.

Logout:

- Ideal Response Time: 50 - 200 ms
- This typically involves invalidating a session or token.

Create Note:

- Ideal Response Time: 200-500 ms
- Involves saving data to the database.

Get All Notes:

- Ideal Response Time: 300 - 1000 ms (depending on the number of notes)
- Fetching multiple records may take longer depending on the number of notes.

Get Note by ID:

- Ideal Response Time: 150 - 400 ms
- A single record fetch should be relatively quick.

Update Entire Note / Update Partial Modification:

- Ideal Response Time: 300 – 600 ms
- Both operations involve writing to the database, though partial updates may be slightly faster.

Delete Note by ID / Delete User:

- Ideal Response Time: 200-400 ms
- Deleting records typically involves database interactions similar to updates.

1.1.2 Features Not Tested

Reset Password Endpoint: The functionality related to resetting user passwords will not be tested because the API does not currently send the password reset link to the registered email. Until this issue is resolved, testing of these endpoints cannot be effectively conducted. The endpoint responsible for validating reset password tokens plus resetting password itself is also excluded from this testing phase. This is due to the dependency on the reset password functionality, which is not operational at this time.

1.1.3 Features Out of Scope

The scope of this testing plan doesn’t include the following key areas:

- This plan focuses solely on API functionality and does not cover any front-end or user interface testing.
- In-depth security testing, such as vulnerability scanning and others, are beyond the scope.
- Performance testing will be limited to basic load testing; no additional performance testing will be conducted.

1.2. API Endpoint Explanation

Check Health: Monitors the availability and operational status of the application.

- Verify that it returns a status code of 200 OK.
- Ensure that the response message includes “Notes API is running.”

Register User: Allows new users to create an account. **Required**: The request must include the following payload in request body in JSON format [name, email and password] where all of them are mandatory.

- Verify that the registration endpoint accepts valid input data and successfully creates a new user account.
- Ensure that appropriate status codes (201 Created) are returned upon successful registration.
- Validate input handling, including checks for required fields, email format, and password complexity requirements and other negative and edge cases.
- Assess error handling by testing various invalid inputs to ensure meaningful error messages are returned.

Login User: Allows registered users to authenticate themselves and gain access to their accounts. **Required**: The request must include the following payload in the request body in JSON format [email, password].

- Verify that the login endpoint accepts valid credentials and successfully authenticates users, returning a session token or access token upon successful login.
- Ensure that appropriate status codes (200 OK for successful logins, 401 Unauthorized for failed attempts) are returned based on the outcome of the authentication process.
- Validate input handling by testing various scenarios, including correct and incorrect username/password combinations and other negative and edge cases.
- Assess error handling to ensure meaningful error messages are provided for invalid credentials.

Retrieve User Profile Information: Allows authenticated users to retrieve their profile information, ensuring secure access to personal data. **Required:** The request must include a valid `x-auth-token` in the request header.

- Verify that the endpoint requires a valid ‘x-auth-token ‘in the request header for access.
- Ensure that the endpoint returns the correct user profile details (e.g. user ID, email, name, email) when provided with a valid token.

- Validate that appropriate status codes (200 OK for successful retrieval, 401 Unauthorized for invalid tokens) are returned based on the authentication status.
- Assess error handling by testing scenarios with invalid or expired tokens and other negative and edge cases to ensure meaningful error messages are returned.

Update User Profile Information: Allows authenticated users to modify their existing profile information, essential for keeping personal data up to date. **Required:** The request must include the following payload in the request body in JSON format [name, phone, company] where 'name' is mandatory, but phone and company are optional. The request must include a valid `x-auth-token` in the request header.

- Verify that the endpoint requires a valid 'x-auth-token 'in the request header for access.
- Validate that the request body must include a mandatory field for the name, while fields for phone and company are optional.
- Ensure that the endpoint successfully updates user profile details (e.g. name, phone, company) when provided with a valid token and correct data.
- Check that appropriate status codes are returned based on the update success (200 OK for successful updates, 401 Unauthorized for invalid tokens).
- Assess error handling by testing scenarios with invalid or expired tokens and other negative and edge cases to ensure meaningful error messages are returned.
- Verify that the system enforces the character limits for the name field (4 to 30 characters) and returns appropriate error messages when these limits are exceeded or not met.

Forget Password: Allows users to initiate a password reset process when they have forgotten their login credentials, crucial for account recovery. **Required**: The request must include the following payload in request body in JSON format [email].

- Verify that the endpoint accepts a valid registered email address in the request body to initiate the password reset process.
- Ensure that an email is sent containing a password reset link or confirmation code when a valid email address is provided.
- Validate that appropriate error messages are returned for invalid email addresses.

- Test that the reset link sent via email directs users to a secure page where they can enter a new password.
- Confirm that the reset link includes a secure token that expires after a specified time (e.g. 1 hour) to prevent unauthorized use.
- Ensure that the new password meets specified security requirements (e.g., minimum length, complexity) when users attempt to set it.
- Verify that users receive a confirmation message or email after successfully resetting their password.

Change Password: Allows authenticated users to update their current password, essential for maintaining account security. **Required:** The request must include the following payload in request body in JSON format [currentPassword, newPassword]. The request must include a valid \`x-auth-token\` in the request header.

- Verify that the endpoint requires a valid ‘x-auth-token ‘in the request header for access.
- Validate that the user must provide their current password as part of the request to verify their identity before allowing a password change.
- Ensure that the new password meets specified security requirements (e.g. minimum length, complexity) and is different from the current password.
- Check that appropriate status codes are returned based on the outcome of the change password request (200 OK for successful changes, 401 Unauthorized for invalid tokens, 400 Bad Request for validation errors).
- Assess error handling by testing scenarios with incorrect current passwords, weak new passwords, expired tokens and other negative and edge cases to ensure meaningful error messages are returned.

Logout: Allows authenticated users to terminate their session and invalidate their access token. **Required:** The request must include a valid \`x-auth-token\` in the request header for access.

- Verify that the endpoint requires a valid ‘x-auth-token ‘in the request header for access.
- Validate that upon successful logout, the provided token is invalidated, preventing any further requests from being processed using that token.
- Check that appropriate status codes are returned based on the outcome of the logout request (200 OK for successful logout, 401 Unauthorized for invalid tokens).

- Assess error handling by testing scenarios with invalid or expired tokens and other negative and edge cases to ensure meaningful error messages are returned.
- Verify that users receive a confirmation message indicating that they have successfully logged out.

Account Deletion: Allows authenticated users to permanently remove their account. **Required:** The request must include a valid `x-auth-token` in the request header for access.

- Verify that the endpoint requires a valid 'x-auth-token 'in the request header for access.
- Validate that the endpoint successfully processes the account deletion request when provided with a valid token.
- Check that appropriate status codes are returned based on the outcome of the delete account request (200 OK for successful deletion, 401 Unauthorized for invalid tokens).
- Assess error handling by testing scenarios with invalid or expired tokens and other negative and edge cases to ensure meaningful error messages are returned.
- Verify that users receive a confirmation message indicating that their account has been successfully deleted.

Create Note: Allows authenticated users to create a new note, enabling better organization of personal information. **Required:** The request must include the following payload in request body in JSON format [title, description, category (Home, Work and Personal)] where all of them are mandatory. The request must include a valid `x-auth-token` in the request header for access.

- Verify that the endpoint requires a valid 'x-auth-token 'in the request header for access.
- Validate that the request body includes all mandatory fields (title, description, and category) and that category is one of the specified options.
- Ensure that the endpoint successfully creates a new note when provided with valid data and a valid token.
- Check that appropriate status codes are returned based on the outcome of the creation request (201 Created for successful creation, 401 Unauthorized for invalid tokens, 400 Bad Request for validation errors).
- Assess error handling by testing scenarios with missing fields, invalid category values, or invalid tokens and other negative and edge cases to ensure meaningful error messages are returned.
- Verify that users receive a confirmation message indicating that 'note' have been created successfully.

Get all Notes: Allows authenticated users to retrieve a list of all their notes, facilitating easy access to personal information. **Required:** The request must include a valid `x-auth-token` in the request header for access.

- Verify that the endpoint requires a valid x-auth-token in the request header for access.
- Ensure that the endpoint successfully retrieves all notes associated with the authenticated user when provided with a valid token.
- Check that appropriate status codes are returned based on the outcome of the request (200 OK for successful retrieval, 401 Unauthorized for invalid tokens).
- Validate that the response includes an array of notes, each containing relevant fields such as id, title, description, category, completed, created_at, updated_at and user_id.
- Assess error handling by testing scenarios with invalid or expired tokens and other negative and edge cases to ensure meaningful error messages are returned.

Get Note by ID: Allows authenticated users to retrieve a specific note using its unique identifier [note_id], enabling access to detailed information about that note. **Required:** The request must include the following parameter in the URL [id]: The request must include a valid `x-auth-token` in the request header for access.

- Verify that the endpoint requires a valid x-auth-token in the request header for access.
- Ensure that the endpoint successfully retrieves the note associated with the provided id when given a valid token.
- Check that appropriate status codes are returned based on the outcome of the request (200 OK for successful retrieval, 401 Unauthorized for invalid tokens, 404 Not Found or 400 Bad requests if the note does not exist).
- Validate that the response includes the note details, such as id, title, description, category, completed, created_at, updated_at and user_id.
- Assess error handling by testing scenarios with invalid or expired tokens, as well as non-existent note IDs, to ensure meaningful error messages are returned.

Update Note by ID: Allows authenticated users to modify an existing note using its unique identifier [id], ensuring that users can keep their notes up to date. **Required:** The request must include the following parameter in the URL [id]: The request must include a valid `x-auth-token` in the request header for access. The request must include the following payload in request body in JSON format [title, description, category (Home, Work and Personal), completed (true and false)] where all of them are mandatory.

- Verify that the endpoint requires a valid x-auth-token in the request header for access.
- Validate that the request body includes only valid fields and that if provided, the category is one of the specified options.
- Ensure that the endpoint successfully updates the note associated with the provided id when given a valid token and correct data.
- Check that appropriate status codes are returned based on the outcome of the update request (200 OK for successful updates, 401 Unauthorized for invalid tokens, 400 Bad Request for validation errors).
- Assess error handling by testing scenarios with invalid or expired tokens, non-existent note IDs, and invalid category values to ensure meaningful error messages are returned.

Update the completed status of the note by id: Allows authenticated users to partially modify the completion status of an existing note using its unique identifier. [id], ensuring that users can keep their notes up to date. **Required:** The request must include the following parameter in the URL [id]: The request must include a valid `x-auth-token` in the request header for access. The request must include the following payload in request body in request body in JSON format [completed (true and false)].

- Verify that the endpoint requires a valid x-auth-token in the request header for access.
- Ensure that the request body includes the mandatory field completed and that it is Boolean value.
- Validate that the endpoint successfully updates the completion status of the note associated with the provided id when given a valid token and correct data.
- Check that appropriate status codes are returned based on the outcome of the update request (200 OK for successful updates, 401 Unauthorized for invalid tokens, 400 Bad Request for validation errors).

Delete note by ID: Allows authenticated users to delete an existing note using its unique identifier, ensuring proper management of their notes. **Required:** The request must include the following parameter in the URL [id]: The request must include a valid `x-auth-token` in the request header for access.

- Verify that the endpoint requires a valid x-auth-token in the request header for access.
- Ensure endpoint successfully processes the deletion request when provided with both valid token and id.
- Check that appropriate status codes are returned based on the outcome of the update request (200 OK for successful updates, 401 Unauthorized for invalid tokens, 400 Bad Request for validation errors).

1.3. Quality Objective

The primary quality objective is to ensure that the Swagger Note API is reliable, secure, and performs well under expected load conditions between **50 to 1000** milliseconds, with a target of 80% of test cases passing successfully. Additionally, maintain a maximum of 5% critical and 20% medium severity bugs if they arise during testing.

1.4. Roles and Responsibilities

- **Tester**
    - Design, execute and document test cases based on requirements and specifications.
    - Perform functional, regression, integration, and performance testing as needed.
    - Report bugs and issues found during testing in a clear and concise manner.
    - Ensure that all test documentation is up-to-date and accessible.

2. Testing Strategy

2.1. Overview

The testing strategy will follow a structured approach, combining manual and automated testing techniques to ensure comprehensive coverage of all functionalities.

2.2. Test Levels

- **Unit Testing**
  - Register User: Tests the logic for user registration, ensuring valid input handling.
  - Login User: Validates the authentication process for users.
  - Create Note: Checks that notes can be created correctly with valid data.
  - Update Note by ID: Tests the functionality for updating all fields of a note
  - Update Completed Status of Note by ID: Validates that specific fields of a note can be updated without affecting others.
  - Delete Note: Ensures that notes can be deleted as expected.

- **Integration Testing**
  - Login User: Validates interaction between authentication and session management
  - Get User Profile: Confirms accurate retrieval of user data post-login.
  - Forget Password: Tests integration with email services for sending reset links.
  - Change Password: Confirms seamless updates to user authentication and data storage.
  - Logout: Ensures session invalidation upon logout.
  - Delete Account: Verifies complete removal of user data from the database.
  - Create Note: Ensures notes are correctly associated with authenticated users.
  - Get all Notes: Validates retrieval of all notes linked to a logged-in user.
  - Get Note by ID: Confirms accurate fetching of specific notes by ID.

- **System Testing**
  - All API Endpoints: System testing will validate the entire application as a whole. This includes testing all endpoints to ensure they work together seamlessly.

2.2. Test Types

- **Smoke Testing**
  - Check health: Ensures the API is operational and responsive before conducting other tests.
  - Register User: Confirms that new users can successfully create accounts without errors
  - Login User: Verify that user authentication functions correctly and allows access to the system.
  - Create Note: Ensures that authorized users can create notes and that the note is stored properly.

- **Functional Testing**
  - All API endpoints (Health, User, Note)

- **Load Testing**
  - All API endpoints (Health, User, Note) specially below ones.
  - Login User: Assesses how the login endpoint performs under load, ensuring it handles multiple simultaneous requests efficiently
  - Create Note: Measures of how quickly notes can be created when multiple users are adding notes simultaneously
  - Get all Notes: Tests performance when retrieving a large number of notes to evaluate response times and system behavior under load.

- **Security Testing**
  - Logout: Ensures that sessions are properly invalidated to prevent unauthorized access after logout.
  - Get User Profile: Only Authorized users can fetch user profile.
  - Update User Profile: Ensures only authorized users can update their profile.
  - Change Password: Ensures only authorized users can change the password.
  - Create Note: Ensures only authorized users can create notes.
  - Get all Notes: Validates access control for retrieving user-specific notes only by authorized users.
  - Get Note by ID: Confirms that only authorized users can access specific notes.
  - Update Notes: Ensures only authorized users can update their notes.
  - Delete Note: Validates that deletion is restricted to the note's owner.

- **Regression Testing**
  - All API endpoints (health, users, notes) if major updates happen.

2.3. Bug Triage

Bugs will be categorized based on severity and priority:

2.3.1. Severity

- **Blocker**: Prevents further testing; must be fixed immediately.
- **Critical**: Major functionality failure; urgent fix needed, but testing can continue other non-affected areas.
- **Normal**: Moderate impact; should be fixed in a timely manner but not urgent.
- **Minor**: Low impact; can be resolved later without significant effect.
- **Trivial**: Minor issues; low priority, fix when convenient.

2.3.2. Priority

- **High**: Requires immediate attention; critical to project success.
- **Medium**: Important but not urgent; should be addressed in the near term.
- **Low**: Minor issues; can be resolved at convenience without impacting on the project timeline.

2.4. Suspension Criteria and Resumption Requirements

Testing may be suspended if:

- Swagger documentation is inaccurate or unclear.
- Server doesn’t send the proper response or server is down.
- Critical bugs severely impact functionality or blockers are identified.
- If any major endpoint exceeds the response time of 1000 milliseconds during Load testing.

Testing will resume once:

- Swagger documentation has been corrected or updated to accurately reflect the API's functionality.
- Testing can resume once the server issue is resolved, confirmed operational, returns expected responses
- Blockers and Critical bugs have been resolved and verified through regression testing to ensure they no longer affect API functionality.
- Testing can resume once optimizations are made and verified.

2.5. Test Completeness

Test completion will be determined by:

- Ensure that the Swagger documentation accurately reflects the API's functionality, including endpoint descriptions, parameter details, and response formats.
- Verify that all API endpoints defined in the Swagger documentation have been tested achieving a minimum of 80% pass rate on all executed test cases.
- Ensure that the API responses are validated against expected outcomes, including status codes, response formats (JSON), and data structures.
- Confirm that all parameters are tested for valid, invalid, and edge case inputs to ensure proper validation, error handling, and maximum test coverage
- Test the security measures by verifying that endpoints enforce authentication and authorization correctly, ensuring only authorized users can access certain functionalities.
- Validate that the API handles errors gracefully by testing various failure scenarios and confirming that appropriate error messages are returned.
- Implement regression tests to confirm that new changes or fixes do not introduce new defects into previously working functionality.
- Confirm that there are no outstanding blockers or critical bugs that affect the testing of the API. All identified issues should be resolved or documented to ensure they do not hinder testing efforts.
- Conduct basic load tests to assess average response times between **50 to 1000** milliseconds while simulating 50 Virtual users and ensure the API can handle expected load without degradation in performance.

3. Test Case Design Techniques

3.1 Equivalence Partitioning (EP)

This technique divides input data into valid and invalid partitions, ensuring the comprehensive coverage of input scenarios.

Register User Endpoint

- Valid Partition: Passwords that are between 6 and 30 characters.
- Invalid Partitions:
  - Passwords less than 6 characters.
  - Passwords greater than 30 characters.

Change Password Endpoint

- Valid Partition: New password must be between 6 and 30 characters
- Invalid Partitions:
  - Passwords less than 6 characters.
  - Passwords greater than 30 characters.

Create Note

- Valid Partition: Title must be between 4 and 100 characters
- Invalid Partitions:
  - Title less than 4 characters.
  - Title greater than 100 characters.

3.2. Negative Testing

Negative and edge case scenarios are also included for all API endpoints as part of the test design techniques to evaluate how the API handles invalid and unexpected inputs. This approach helps maximize test coverage and ensures that the API can gracefully manage errors, thereby enhancing its robustness and reliability.

3.3. State Transition

It is a powerful technique that improves understanding, enhances test coverage, detects errors effectively, and supports robust test case design, making it invaluable for ensuring software quality in complex applications.

The state of a user or note can change based on various actions, and it's essential to model these transitions effectively. Below is a structured view of the state transitions associated with the operations you mentioned.

State Transition in tabular format for easy understanding.

OPERATION	INITIAL STATE	ACTION	NEW STATE
Register User	Unregistered	Register	Registered
Login User	Registered	Login	Logged In
Get User Profile	Logged In	Fetch Profile	Profile Retrieved
Update User Profile	Logged In	Update Profile	Profile Updated
Reset Password	Registered/Logged In	Reset Password	Password Reset
Logout	Logged In	Logout	Registered
Delete Account	Registered/Logged In	Delete Account	Account Deleted
Create Note	Note Empty	Create Note	Note Created
Get all Notes	Note/s Exist	Fetch All Notes	Note Retrieved
Get Note by ID	Note Exist	Fetch Note by ID	Note Retrieved
Update Note by ID	Note Exists	Update Note	Note Updated
Update Partially	Note Exists	Partial Update	Note Partially Updated
Delete Note by ID	Note Exists	Delete Note	Note Deleted

4. Testing Artifacts

- Test Plan
- Test Cases
- Test Scripts
- Test Execution Reports
- Defect Reports
- Load Testing Report
- Test Summary Report

5. Resources and Environment Needs

- **Swagger UI**: For exploring API endpoints and reviewing documentation.
- **Postman**: For API testing and execution of test cases.
- **JMeter**: Simulating virtual users for Load Testing.
- **Excel**: For tracking test cases and managing tasks.
- **Word**: For reporting Defects.
- **Operating System**: Windows, Mac or Linux (any)
- **Network Configuration:** Above 50 Mbps

6. Test Execution

Manual and Automated Testing with Postman

API endpoints will be tested both manually and automatically through the Collection Runner using the Postman GUI. This approach will allow testers to interactively send requests, inspect responses, and validate the functionality of each endpoint.

Automated Testing with Newman

Automated tests will also be executed using Newman, a CLI tool that runs Postman collections. This will enable seamless integration of automated testing into CI/CD pipelines, enhancing efficiency and consistency in the testing process for future use along with user friendly HTML report generation.

7. Defect Management

Defect Logging Process

Any differences between the expected and actual results will be documented as defects. Each defect will be logged in an Excel file to ensure a systematic approach to tracking issues encountered during testing.

Excel File Structure

The bug log in Excel will include several key columns to effectively track and manage defects. This template will be generated from **Jira**, ensuring that all relevant information is captured systematically.

- Bug ID: A unique identifier for each defect.
- Bug Description: A brief description of the defect, detailing what went wrong.
- Status: Current status of the defect (New, Open, Resolved, To Do).
- Priority: The urgency with which a defect should be addressed (High, Medium, Low).
- Assignee: The team member responsible for addressing the defect.
- Attachments: Supporting files like screenshots, videos or logs.
- Operating System: OS and version where the defect were found.
- Resolution: Summary of how the defect was fixed (Solved, Unsolved).
- Description:
  - Preconditions: Required setup before reproducing the defect.
  - Steps to Reproduce: Actions to replicate the defect.
  - Expected Result: Anticipated correct behavior.
  - Actual Result: Observed outcome when the defect occurred.

8. Reporting and Metrics

Reports will be generated to summarize test execution results using **newman-reporter-generator**, providing detailed insights into the outcomes of all executed test cases, including pass, fail, and skip rates. These reports will highlight key metrics such as the pass rate, which indicates the percentage of test cases that passed successfully; the fail rate, representing the percentage of test cases that failed; and the skip rate, showing the percentage of test cases that were skipped during execution.

In addition to these execution metrics, defect metrics will be compiled to track any issues identified during testing. This will include total defect counts, severity levels that classify defects based on their impact (critical, major, minor etc.), and defect percentages that reflect the ratio of defects relative to total test cases executed.

Furthermore, load test results will be generated in both CSV and HTML formats, facilitating easier data analysis. By leveraging these comprehensive metrics, teams can effectively assess software quality and pinpoint areas for improvement, ultimately enhancing both development and testing processes.

9. Test Schedule

The following table outlines the comprehensive test schedule for my project, detailing the various phases of testing, their respective start and end dates, and the duration allocated for each phase. This structured approach ensures that all critical aspects of the testing process are systematically addressed within a one-month timeframe, from June 1 to June 30. Each phase is designed to facilitate thorough planning, execution, and closure of testing activities, ultimately contributing to the quality and reliability of the final product.

PHASE	START DATE	END DATE	DURATION
Test Planning	June 1, 2024	June 5, 2024	5 Days
Test Case Development	June 6, 2024	June 15, 2024	10 Days
Test Execution	June 16, 2024	June 25, 2024	10 Days
Bug Triage	June 26, 2024	June 28, 2024	3 Days
Test Closure	June 29, 2024	June 30, 2024	2 Days

10. Risks and Mitigation

10.1. Risks Associated with the Testing Phase

- **Risk of Incomplete Test Coverage**: As an individual tester, there may be a tendency to overlook edge cases or less critical functionalities due to limited resources and time.
- **Environment Instability:** The local testing environment may face issues, such as server downtime or configuration problems, which can hinder testing efforts.
- **Lack of Peer Review**: Without a team, there is no opportunity for peer review, which may result in unrecognized errors in test cases or oversight in testing logic.
- **Risks related to performance**: High response times could lead to user dissatisfaction and abandonment.

10.2. Mitigation Strategies for identified Risks

- **Risk of Incomplete Test Coverage**: Create a comprehensive test plan that includes all functionalities and edge cases. Regularly review and update test cases to ensure they cover all aspects of the API.
- **Environment Instability**: Set up a backup testing environment like creating a Mock Server when the sever goes down.
- **Lack of Peer Review**: Utilize online forums, communities, or testing groups to share test cases and receive feedback. Consider documenting the testing process for future reference or self-review.
- **Continuous Monitoring**: Implement continuous monitoring via. Postman to track response times in real-time.