Test Plan for Products and Cart Management APIs

Table of Contents

1. Introduction

1.1 Purpose
The purpose of this document is to outline the test plan for evaluating the behavior of the API endpoints related to Products and Cart Management. This includes testing both authorized and unauthorized scenarios to ensure functionality, security, and efficiency.

1.2 Test Objectives
The testing of the system should validate the requirements that:

- All API end points are working are expected in product area and cart area
- Retrieving a list of Products functions work correctly.
- Reading Product details functions work correctly.
- Editing cart functions work correctly.
- Clearing cart functions work correctly.
- The system is easy to use by the end-users.
- Security controls are in place to prevent unauthorized system access.
- All points of integration within the system work as defined in requirements.

The main goals of this test plan are to ensure that the API works as intended, follows security and performance standards, provides broad test coverage, prioritizes tests based on potential risks and functionality, and creates automated test cases for applicable scenarios.

1.3 Scope of Testing
This test plan covers the following API endpoints:

- Retrieving a list of Products
- Reading Product details
- Adding/Editing Cart
- Clearing Cart

2. Approach

2.1 Assumptions
Each build of the Picnic system will have passed unit tests before it is transferred to the testing environment.

2.2 Coverage
All available APIs Product and Carts will be tested.

Test coverage will be measured by:

- Endpoint Coverage
  Measure endpoint coverage by tracking how many API endpoints or methods have been tested. This involves creating test cases for each API endpoint and ensuring that all endpoints are covered.
- Error Handling Coverage
  Measure how well the API responds to different error conditions and ensure that these cases are thoroughly tested.
- Test Automation coverage
  Measure how many tests are automated to achieve consistent and repeatable test coverage.

2.3 Test Type
The following types of testing will be performed:

- Functional testing, by performing test cases based on business functions.
- Regression testing, to ensure that a change to the system does not introduce new defects.
- Security testing, by testing each end-user's security access levels.
- Performance testing, by evaluating performance against business requirements.

2.4 Test Data
To perform testing on APIs, test data will be supplied with APIs from docker image named teampicnic/qa-recruitment.

2.5 Test Tools
To perform testing, following test tools as part of solution:

- Robot Framework
- Defect Tracker
- Postman
- JMeter

3. Test Strategy
The test strategy involves a combination of manual and automated testing. Manual testing is crucial for exploratory and edge-case scenarios, while automation helps with regression and repetitive testing.

4. APIs to be tested

4.1 Description
The API provides services for managing products and carts. It allows users to retrieve product information, edit their cart, and clear the cart.

4.2 API Endpoints
The API has the following endpoints:

- `GET  /products`: Retrieve a list of products.
- `GET  /products/{product_id}`: Read product details.
- `GET  /cart`: Read the user's cart.
- `PATCH  /cart`: Edit the user's cart.
- `POST  /cart/clear`: Clear the user's cart.

5. APIs not to be tested
There is no scenario applicable now.

6. Test Scenarios and Prioritization

6.1 Retrieving a List of Products

Test Scenario 1 – Fetch product details list (High Priority)

- Description: Verify that a list of products is successfully retrieved.
- Test Type: Manual and Automated
- Preconditions: None
- Steps:
    - Send a GET request to ` /products`.

Expected Result: A list of products is returned with a 200 status code.

Test Scenario 2 – Fetch product details list with a limit (Medium Priority)

- Description: Verify that a limited list of products is successfully retrieved.
- Test Type: Manual and Automated
- Preconditions: None
- Steps:
  - Send a GET request to ` /products`. with a valid value in product limit
- Expected Result: A limited list of products is returned with a 200 status code.

Test Scenario 3 – Fetch product details list with skip parameter (Medium Priority)

- Description: Verify that a list of products successfully retrieved with given number of skipped products.
- Test Type: Manual and Automated
- Preconditions: None
- Steps:
  - Send a GET request to ` /products`. with a valid value in skip parameter
- Expected Result: A list of products is returned with a 200 status code.

6.2 Reading Product Details

Test Scenario 4 – Read single product detail successfully (High Priority)

- Description: Verify that product details are successfully read.
- Test Type: Manual and Automated
- Preconditions: None
- Steps:
  - Send a GET request to ` /products/{product_id}`
- Expected Result: Product details are returned with a 200 status code.

Test Scenario 5 – Try to fetch product detail for a nonexistent product ID (Medium Priority)

- Description: Test the API's response response when the requested product does not exist.
- Test Type: Manual and Automated
- Preconditions: None
- Steps:
  - Send a GET request to `/products/{nonexistent_product_id}`.
- Expected Result: A 404 Not found status.

Test Scenario 6 - Try to fetch product detail for invalid product ID (Medium Priority)

- Description: Test the API's response with invalid product ID type.
- Test Type: Manual and Automated
- Preconditions: None
- Steps:
  - Send a GET request to ` /products/{nonexistent_product_id}`.
- Expected Result: A 422 Error response is returned with error message.

Test Scenario 7 - Try to fetch product detail with non-supported request method (Medium Priority)

- Description: Test the API's response with invalid product ID type.
- Test Type: Manual and Automated
- Preconditions: None

- Steps:
    - Send a request other than GET method to ` /products/{product_id}`.
- Expected Result: Method not allowed message returned with 4XX status

6.3 Updating Cart

Test Scenario 8 - Add to cart successfully (High Priority)

- Description: Verify that the user is added the product to cart successfully.
- Test Type: Manual and Automated
- Preconditions:
    - Authentication token is valid.
    - Cart is empty.
- Steps:
    - Send a PATCH request to ` /cart` with valid data.
- Expected Result: The cart is successfully updated with a 200 status code.

Test Scenario 9 – Update cart successfully (High Priority)

- Description: Verify that the user's cart is successfully edited.
- Test Type: Manual and Automated
- Preconditions:
    - Authentication token is valid.
    - A single item is available in the cart.
- Steps:
    - Send a PATCH request to ` /cart` with valid data.
- Expected Result: The cart is successfully updated with a 200 status code.

Test Scenario 10 - Remove product from cart successfully (Medium Priority)

- Description: Verify that the user can remove an item from the cart successfully.
- Test Type: Manual and Automated
- Preconditions:
    - Authentication token is valid.
    - A single item is available in the cart.
- Steps:
    - Send a PATCH request to ` /cart` with valid data.
- Expected Result: The item is removed from the cart successfully edited with a 200 status code.

Test Scenario 11 – Update card with Invalid Input (High Priority)

- Description: Test the API's response to invalid input for cart editing.
- Test Type: Manual and Automated
- Preconditions:
    - Authentication token is valid.
- Steps:
    - Send a PATCH request to ` /cart` with invalid data.
- Expected Result: 422 Validation Error response is returned with error message.

Test Scenario 12 – Update card with Empty data (High Priority)

- Description: Test the API's response to empty data for cart editing.
- Test Type: Manual and Automated
- Preconditions:
    - Authentication token is valid.
- Steps:

    - Send a PATCH request to ` /cart` with empty data.

  Expected Result: 422 Validation Error response is returned with error message.

Test Scenario 13 – Update cart with Unauthorized Access (High Priority)

- Description: Test unauthorized access to the ` /cart` endpoint.
- Test Type: Manual and Automated
- Preconditions:
    - Authentication token is invalid.
- Steps:
    - Send a PATCH request to ` /cart` with valid data.
- Expected Result: A 422 Validation error response is returned.

Test Scenario 14 – Update cart with expired token (High Priority)

- Description: Test the API's response to invalid input for cart editing.
- Test Type: Manual and Automated
- Preconditions:
    - Authentication token is valid.
    - Add item to cart with valid token.
    - Delete the current token.
- Steps:
    - Send a PATCH request to ` /cart` with expired token.

Expected Result: Unauthorized access error message is returned.

Test Scenario 15 – Update cart with Invalid Request Method (Medium Priority)

- Description: Test the API's response to invalid request method.
- Test Type: Manual and Automated
- Preconditions: Authentication token is valid
- Steps:
    - Send a POST request to ` /cart` with valid data.

Expected Result: Method not allowed error message is returned.

Test Scenario 16 – Update cart with unavailable product (Medium Priority)

- Description: Test the API's response to cart editing
- Test Type: Manual and Automated
- Preconditions: Authentication token is valid
- Steps:
    - Send a POST request to ` /cart` with data containing ID of unavailable product.

Expected Result: The response contains 200 status with message that the product cannot be added due to unavailability.

6.4 Clearing Cart

Test Scenario 17 - Successful Clearing (High Priority)

- Description: Verify that the user's cart is successfully cleared.
- Test Type: Manual and Automated
- Preconditions:
    - Authentication token is valid.
    - Items are added to the cart.
- Steps:
    - Send a POST request to `/cart/clear`.
- Expected Result: The cart is successfully cleared with a 200 status code.

Test Scenario 18 – Clear cart with Unauthorized Access (High Priority)

- Description: Test unauthorized access to the ` /cart` endpoint.
- Test Type: Manual and Automated
- Preconditions:
  - Authentication token is invalid.
- Steps:
  - Send a POST request to ` /cart/clear`.
- Expected Result: A 422 Validation error response is returned.

Test Scenario 19 – Clear cart with Invalid Request Method (Medium Priority)

- Description: Test invalid request support to the ` /cart/clear` endpoint.
- Test Type: Manual and Automated
- Preconditions:
  - Authentication token is valid.
- Steps:
  - Send a request other than POST to ` /cart/clear`.
- Expected Result: Method not allowed error response is returned with 405 status.

7. Definition of Ready

To ensure an effective testing process, make sure to take the following steps beforehand:

- Set up and configure the test environments.
- Prepare the necessary test data, including both valid and invalid test cases.
- Ensure that the authorization tokens required for authenticated scenarios are available.
- Review the API documentation thoroughly to gain a comprehensive understanding of the expected behaviour.

8. Acceptance Criteria

- Test cases should be considered successful if they meet the defined expected results.
- Automated tests should be integrated into the continuous integration pipeline.
- All high-priority manual test scenarios should be executed before release.
- Make sure that the performance of the APIs are met with business requirements

9. Defect management

9.1 Defect Categories

- Functional Defects: Any deviation from the expected functionality of the API.
- Performance Issues: Problems related to API speed, scalability, or resource utilization.
- Security Vulnerabilities: Identification of security-related issues or vulnerabilities.
- Data Accuracy Issues: Problems with the accuracy of data returned by the API.
- Usability and Documentation Issues: Problems related to usability or inconsistencies in API documentation.

9.2 Defect Reporting
- Testers are responsible for documenting defects they encounter during testing.
- Defect reports should include clear and detailed information to aid in resolution.

9.3 Defect Tracking
- Use a defect management tool like Jira to track and manage defects.
- Testers, developers, and stakeholders will have access to the tool for transparency.

9.4 Defect Lifecycle
Defects will go through the following stages:
- New: A new defect that has been reported.
- Open: Confirmed and accepted by the development team.
- In Progress: Currently being addressed by a developer.
- Resolved: The defect has been fixed by the developer.
- Verified: The tester has verified that the defect is resolved.
- Closed: The defect has been successfully resolved and verified.

9.5 Defect Prioritization
Defects will be categorized by priority:
- Critical: Defects that must be addressed immediately due to severe impact.
- High: Defects with significant impact but not immediately critical.
- Medium: Defects with a moderate impact on functionality.
- Low: Defects with minimal impact and low priority for immediate resolution.

9.6 Defect Assignment
The development team will be responsible for assigning defects to specific developers.

9.7 Defect Verification
Testers will verify that resolved defects are indeed fixed. Verified defects will be marked as closed in the defect management tool.

10. Risks and Contingencies
- API Changes:
  The API might change unexpectedly, leading to test failures.
  Contingency - Regularly review API documentation and conduct version control to ensure that you are aware of upcoming changes. Automate tests to detect API changes as early as possible.
- Test Environment Availability:
  The test environment may not be available or may be unstable.
  Contingency: Implement continuous integration and continuous delivery (CI/CD) practices to ensure the availability of stable test environments. Use infrastructure as code (IaC) to automate environment provisioning.
- Test Automation Challenges:
  Developing and maintaining API test automation can be challenging.
  Contingency: Implement version control and continuous integration for test scripts. Regularly review and update automated tests.
- Documentation Gaps:
  Incomplete or outdated API documentation can lead to misunderstandings.
  Contingency: Collaborate with the development team to ensure documentation is accurate and up-to-date.

11. Conclusion

This test plan outlines the testing strategy for the API endpoints that are related to Products and Cart Management. The goal is to ensure the correct operation and security of the API by prioritizing tests based on critical functionality. The automated test cases that defined will serve as the basis for continuous testing and validation.