SOFTWARE TEST PLAN

PetStore Swagger UI Site

Team Members

Project Manager: Yair Amon

Test Team Lead: Tzahi Anedghr

Test Engineers: Wael Otman

TABLE OF CONTENTS

INTRODUCTION

1 The purpose of this Software Test Plan (STP) is to comprehensively outline the meticulous testing approach, methodologies, and strategies devised for the thorough evaluation of the Swagger Petstore API. This document is meticulously crafted to serve as an indispensable guide for the testing team, ensuring not only the quality, reliability, and functionality of the API but also its adherence to industry standards and best practices. By delineating clear testing objectives and procedures, this STP aims to facilitate a systematic and rigorous testing process that guarantees the delivery of a robust and dependable API to end-users.

2 TEST STRATEGY

2.1 Scope of Testing

The scope of this STP encompasses the testing of the Swagger Petstore API, focusing on its functionality, usability, performance, security, and compatibility across various platforms and environments. The testing activities will cover all API endpoints and features specified in the Swagger documentation.

2.2 Glossary

- **Swagger Petstore API:** Web service managing pet store inventory, enabling pet-related operations like adding, deleting, and updating pets, as well as managing users and orders.
- **Endpoint:** URL representing a specific action or resource within the API, such as /pets or /users.
- **RESTful:** Architecture emphasizing stateless communication via HTTP methods (GET, POST, PUT, DELETE) for data exchange between systems.
- **API:** Set of rules and tools facilitating communication and data exchange between software systems.
- **JSON:** Lightweight data-interchange format for easy human readability and machine parsing.
- **HTTP:** Protocol for web communication, used by clients to request resources and exchange data with servers.
- **CRUD:** Fundamental operations of persistent storage: Create, Read, Update, Delete.
- **Authorization:** Process determining if a user, application, or system is permitted to perform a requested action or access a resource.
- **Authentication:** Process verifying the identity of a user, application, or system through credentials like username and password.
- **Response Code:** Numeric code indicating the status of a client request, such as 200 for success or 404 for not found.
- **STP (Software Test Plan):**

  Definition: Document outlining testing goals, methods, resources, and schedule.

  Explanation: Guides testing process, detailing what, how, and by whom testing will be conducted.

- **STD (Software Test Description):**

  Definition: Detailed document specifying individual test cases.

  Explanation: Describes test scenarios, inputs, expected outcomes, and execution steps.

- **STR (Software Test Report):**

  Definition: Summary document detailing testing results.

  Explanation: Provides insights into testing outcomes, including defects and overall software quality.

- **Bugs:**

  Definition: Software errors causing unexpected behavior.

  Explanation: Range from minor to critical issues, affecting functionality or security.

- **Test Case:**

  Definition: Set of steps to verify software functionality.

  Explanation: Guides systematic testing of software features.

2. TESTS TREE

2.1 test tree of functional tests

A) Functional tests:

1) Pet Operations

a. Add a new pet.
   - Verify successful addition of a pet with valid information
   - Verify error handling for invalid pet data (e.g., missing required fields)

b. Update an existing pet
   - Verify successful update of pet information
   - Verify error handling for updating non-existent pet

c. Find pet by ID
   - Verify successful retrieval of pet by valid ID
   - Verify error handling for non-existent pet ID

d. Find pet by status
   - Verify successful retrieval of pets by valid status (e.g., available, pending)
   - Verify error handling for invalid status input

2) User Operations

a. Create a new user:
   - Verify successful creation of a user with valid information
   - Verify error handling for invalid user data (e.g., missing required fields)

b. Update an existing user
   - Verify successful update of user information
   - Verify error handling for updating non-existent user

c. Find user by username
    - Verify successful retrieval of user by valid username
    - Verify error handling for non-existent username

3) Place an order

a. Click on a search result to view item details.
    - Verify successful placement of an order with valid information
    - Verify error handling for invalid order data (e.g., missing required fields)

b. Get order by ID
    - Verify successful retrieval of order by valid ID
    - Verify error handling for non-existent order ID

c. Delete an order
    - Verify successful deletion of an order by valid ID
    - Verify error handling for deleting non-existent order

2.2 test tree of functional tests

B) Non-Functional tests

1) Performance Testing

a. Load Testing:
    - Test the API's response time under various loads (low, medium, high).
    - Evaluate the API's throughput and resource utilization under different load conditions.

b. Stress Testing:
    - Assess the API's stability and behavior under extreme loads.
    - Determine the breaking point and performance degradation threshold.

c. Scalability Testing:
    - Evaluate the API's ability to handle increased loads by scaling horizontally or vertically.
    - Test the API's performance with increased concurrent users or requests.

2) **Usability Testing**

a. Documentation Review:
    - Evaluate the clarity and completeness of API documentation.

b. API Usability Testing
    - Assess the ease of use and intuitiveness of API endpoints and parameters.

c. Error Handling Usability
    - Evaluate the user-friendliness of error messages and troubleshooting guidance.

3) **Compatibility Testing**
    a. Browser Compatibility:
        - Test the API's compatibility with different web browsers (e.g., Chrome, Firefox, Safari).

    b. Platform Compatibility:
        - Verify the API's compatibility with different operating systems (e.g., Windows, Linux, macOS).

4) **Reliability Testing**
    a. Stability Testing:
        - Assess the API's ability to maintain consistent performance over a prolonged period.
        - Test for memory leaks or resource exhaustion under continuous usage.

    b. Error Handling Testing:
        - Verify the API's response to unexpected inputs or errors.
        - Test for graceful degradation and appropriate error messages.

    c. Recovery Testing
        - Evaluate the API's recovery mechanisms after a system failure or interruption.
        - Test for data consistency and integrity after recovery procedures.

2.3. Feature to be tested

Feature	Description
Pet Management	- Verify the ability to add new pets with correct information. Test updating existing pet details. Confirm retrieval of pet information by ID.
User Management	- Ensure successful creation of new users. Test updating user information. Verify user authentication and login functionality.
Order Management	- Confirm the ability to place new orders. Test updating order details. Verify order retrieval by ID.
Inventory Management	- Check inventory status for available pets. Test adding new inventory items. Verify inventory deletion.
Authentication and Authorization	- Ensure proper authentication mechanisms. Test authorization for different user roles. Verify access control for sensitive operations.
Error Handling	- Test error messages for clarity and helpfulness. Ensure proper handling of unexpected errors or exceptions.
API Documentation	- Verify completeness and clarity of API documentation. Ensure accurate description of API endpoints and parameters.
Security	- Conduct security testing to identify and mitigate vulnerabilities. Verify secure transmission of sensitive data.
Pet Status Management	- Test updating pet status (e.g., available, pending, sold). Verify retrieval of pets by status.
Order Fulfillment	- Confirm order processing (e.g., fulfillment, shipping). Test order cancellation and refund handling.

2.4 Feature not to be tested

- UI Components: User interface elements may not be extensively tested due to the API's focus on backend functionality.
-
- Obsolete Functionality: Features marked for removal might not receive comprehensive testing.
-
- Platform-specific Features: Testing on specific platforms may be limited if the API aims for broad compatibility.
-
- Advanced Security Features: Extensive testing of advanced security measures might be limited to essential aspects.
-
- Non-essential Error Cases: Testing of non-critical error scenarios may be minimal, with focus on critical errors.

3 Test Type

1. **Functional Testing**:
   - **API Endpoint Testing**: Verify each API endpoint's functionality (e.g., adding a pet, updating a user).
   - **Data Validation Testing**: Ensure correct data validation (e.g., proper handling of invalid input).
   - **Error Handling Testing**: Test how the API responds to various error conditions (e.g., invalid requests, server errors).
2. **Non-functional Testing**:
   - **Performance Testing**: Assess response times and throughput under normal and peak loads.
   - **Security Testing**: Check for vulnerabilities like injection attacks and unauthorized access.
   - **Usability Testing**: Evaluate the API's ease of use, documentation clarity, and error messages.
3. **Integration Testing**:
   - **Third-party Integration Testing**: Ensure smooth interaction with external services like payment gateways.
   - **Compatibility Testing**: Verify compatibility across different browsers, platforms, and devices.
4. **Regression Testing**:
   - **API Regression Testing**: Re-test previously validated functionalities after updates or changes.

- **Database Regression Testing**: Ensure that changes do not adversely affect data integrity or performance.

5. **End-to-End Testing**:
   - **Order Processing**: Test the entire process of placing an order, including pet selection and payment.
   - **User Registration Flow**: Validate the entire user registration process, from account creation to login.
6. **Load Testing**:
   - **Stress Testing**: Evaluate how the API performs under extreme loads or resource constraints.
   - **Scalability Testing**: Assess the API's ability to handle increased load by scaling horizontally or vertically.
7. **Security Testing**:
   - **Authentication Testing**: Verify the effectiveness of authentication mechanisms (e.g., OAuth).
   - **Authorization Testing**: Ensure that access controls are properly enforced for different user roles.
8. **Usability Testing**:
   - **API Documentation Review**: Evaluate the clarity and completeness of API documentation.
   - **Error Message Usability**: Assess the user-friendliness and comprehensibility of error messages.

4 Risk and Issues

Risk	Explanation
API Changes	Changes in the API specifications may lead to outdated test cases, affecting the accuracy of testing.
Dependency on External Services	The API relies on external services; if they're unavailable or unpredictable, it may affect test execution and results.
Security Vulnerabilities	Weaknesses in the API's security may result in data breaches or unauthorized access, posing risks to sensitive information.
Poor Performance Under Load	The API may slow down or become unresponsive when subjected to heavy usage or high traffic, impacting user experience.
Lack of Documentation	Inadequate or unclear documentation may hinder understanding of API functionality, leading to testing challenges.
Compatibility Issues	Compatibility issues with different client libraries or frameworks may arise, affecting the API's usability in various environments.
Data Integrity Concerns	Changes to the API could compromise data integrity, leading to inaccurate test results or potential data loss.

5 Test Logistics

5.1 Who will test?

The project will be tested by the Beyondev QA Software Engineering Team.

5.2 When will the test occur?

The timing of API testing can vary depending on the development lifecycle, project requirements, and team preferences

1. **Unit Testing**: Tests individual components or functions in isolation to ensure they work as intended.
2. **Integration Testing**: Verifies interactions between integrated components, including APIs, databases, and external services.
3. **System Testing**: Evaluates the entire system's functionality and behavior to ensure it meets requirements.
4. **Acceptance Testing**: Validates whether the system meets business requirements and user expectations.
5. **Regression Testing**: Ensures that changes or updates to the system do not introduce new bugs or regressions.
6. **Performance Testing**: Evaluates the system's responsiveness, scalability, and reliability under different conditions.
7. **Security Testing**: Identifies and mitigates security vulnerabilities in the system, such as data breaches or unauthorized access.

6 TEST OBJECTIVE

These test objectives aim to ensure the reliability, functionality, security, and usability of the Swagger Petstore API, providing a clear direction for testing efforts and helping stakeholders understand the expected outcomes of the testing activities.

1. **Accurate Pet Retrieval**: Verify the API retrieves pet information correctly based on parameters.
2. **Complete Pet Details**: Ensure all pet details, like name, category, and status, are complete.
3. **Order Management Functionality**: Test users' ability to place orders and view order details.
4. **User Authentication and Security**: Validate secure access to user accounts and data.
5. **Error Handling**: Assess the clarity of error messages for better user understanding.
6. **Performance Under Load**: Check API responsiveness and scalability under varying user loads.
7. **Compatibility**: Test API compatibility with different client libraries and environments.
8. **Documentation Clarity**: Ensure API documentation is clear, accurate, and comprehensive.
9. **Regression Testing**: Confirm existing functionalities remain intact after updates.

7 TEST CRITERIA

7.1 Suspension Criteria

- If critical API functionalities fail in more than 30% of test cases, testing will be suspended until the development team resolves the issues.
- Any security vulnerabilities discovered during testing must be addressed immediately before further testing can proceed.

7.2 Exit Criteria

- All core functionalities of the API, as outlined in the requirements, must have been tested.
- The pass rate for functional test cases should be at least 90%, with no critical defects remaining unresolved.

8 RESOURCE PLANNING

8.1 System Resource

No.	Resources	Descriptions
1.	API Testing Framework	Utilize a suitable API testing framework (e.g., Postman, RestAssured) to automate API testing and validation.
2.	Test Environment	Set up a dedicated test environment to replicate production conditions, including necessary dependencies.
3.	Test Data	Prepare relevant test data to cover various scenarios, including valid and invalid input data.
4.	Test Automation Tools	Employ automation tools like Postman or Newman to automate API testing processes and ensure consistency.
5.	Version Control System	Implement a version control system (e.g., Git) to manage test scripts, track changes, and facilitate collaboration.
6.	Monitoring Tools	Utilize monitoring tools (e.g., New Relic, Datadog) to track API performance and identify bottlenecks.
7.	Load Testing Tools	Employ load testing tools (e.g., Apache JMeter, LoadRunner) to simulate high traffic and measure API response times under stress.
8.	Security Testing Tools	Utilize security testing tools (e.g., OWASP ZAP, Burp Suite) to identify and address security vulnerabilities.
9.	Documentation Management	Maintain comprehensive documentation to track test cases, results, and any issues encountered during testing.
10.	Collaboration Platforms	Use collaboration platforms (e.g., JIRA, Confluence) to manage testing tasks, communicate issues, and track progress.

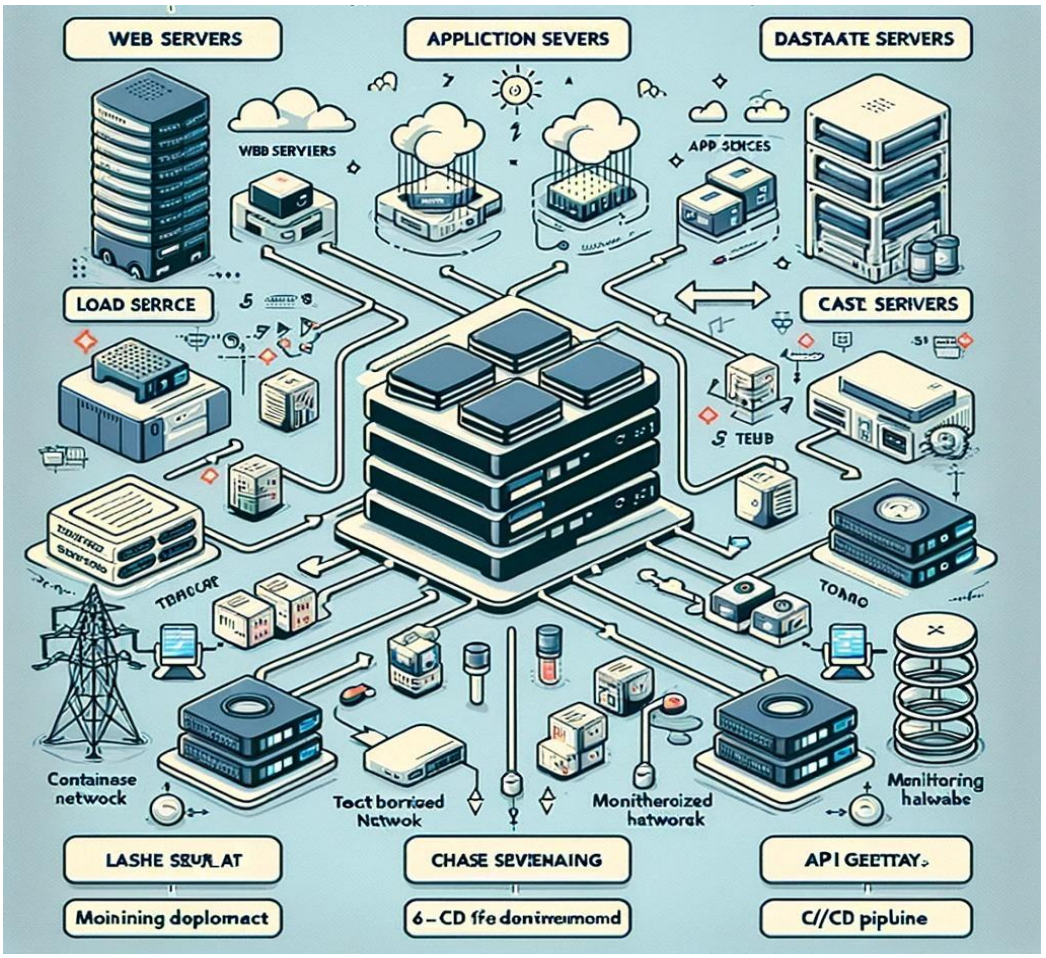
8.2 Human Resource

No.	Member	Tasks
1.	Project Manager	Leads the testing project. Coordinates with team members and ensures resources are available as needed.
2.	Test Team Lead	Leads the testing team. Defines test strategies, selects appropriate tools, and designs test frameworks.
3.	Automation Testers	Write and execute automated tests using selected tools (e.g., Postman,).
4.	Test Environment Manager	Sets up and maintains the test environment, ensuring it's stable and reflects production conditions.
5.	Quality Assurance Analysts	Review test cases to ensure quality and effectiveness. Provide feedback and suggestions for improvement.

9 TEST ENVIRONMENT

To create a test environment for the Swagger Petstore API:

1. **Server Infrastructure**: Set up servers or cloud instances with sufficient resources.
2. **Database Setup**: Create and populate a database with relevant test data.
3. **API Testing Tools**: Install and configure tools like Postman or RestAssured for testing.
4. **Dependency Management**: Install necessary dependencies and ensure compatibility.
5. **Security Configuration**: Implement security measures and apply patches regularly.
6. **Logging and Monitoring**: Configure logging and monitoring for debugging and analysis.
7. **Network Configuration**: Simulate real-world network conditions for testing.
8. **Version Control and Deployment**: Use version control and automate deployment processes.
9. **Documentation and Collaboration**: Document setup details and collaborate with team members.



Web Servers: While the Swagger Petstore API is not a traditional web application, it may still be hosted on web servers to handle incoming HTTP requests and serve API responses.

Application Servers: The application servers execute the business logic of the Swagger Petstore API. They handle incoming API requests, process data, and generate responses according to the API's specifications.

Database Servers: Database servers store the data related to pets, orders, users, and other entities managed by the Swagger Petstore API. They handle data storage, retrieval, and manipulation operations.

API Gateway: The API gateway may not be explicitly mentioned for the Swagger Petstore API since it's a simple API, but in a more complex setup, an API gateway could be used to manage and route API requests to the appropriate services.

CI/CD Pipeline: A CI/CD pipeline may be used for continuous integration and continuous deployment of changes to the Swagger Petstore API. This pipeline automates the process of building, testing, and deploying updates to the API, ensuring its reliability and quality.

This test environment ensures that the Swagger Petstore API functions correctly, performs well under various conditions, and meets the requirements specified in its documentation.

10 SCHEDULE & ESTIMATION

10.1 All project task and estimation

Task	Members	Estimate effort
Create the test specification	Test Administrator	35 man-hours
Perform Test Execution	Tester, Test Administrator	25 man-hours
Test Report	Tester	15 man-hours
Test Delivery	Tester	20 man-hour
Total		**95 man-hour**

11 TEST DELIVERABLES

Test deliverables for the Swagger Petstore API testing may include:

1. **Test Plan**: A comprehensive document outlining the testing approach, methodologies, test objectives, test scope, and resources required for testing the Swagger Petstore API.
2. **Test Cases**: Detailed test cases covering various functionalities, scenarios, and edge cases of the Swagger Petstore API. These test cases specify inputs, expected outputs, and steps to reproduce.
3. **Test Data**: Relevant data sets used for testing the Swagger Petstore API, including sample pet information, user details, order data, and any other necessary data required to execute test cases.
4. **Test Reports**: Reports summarizing the results of test execution, including test case execution status, defects found, test coverage metrics, and any observations or recommendations for improvement.
5. **Defect Reports**: Documentation of any defects or issues identified during testing, including steps to reproduce, severity, priority, and status of resolution.
6. **Performance Test Results**: Reports on the performance testing conducted for the Swagger Petstore API, including metrics such as response times, throughput, and resource utilization under different load conditions.
7. **Security Test Results**: Findings from security testing activities, including vulnerabilities identified, risk assessments, and recommendations for remediation.
8. **Documentation Updates**: Updates to API documentation based on testing insights, including any changes or enhancements to API endpoints, parameters, or functionality.
9. **Test Environment Setup Guide**: Documentation detailing the setup and configuration of the test environment for the Swagger Petstore API, including software, hardware, and network requirements.
10. **Training Materials**: Training materials or knowledge transfer sessions to share testing methodologies, best practices, and lessons learned from testing the Swagger Petstore API with the testing team or stakeholders