**Test Report: REST API Automation**

**Overview**

This document serves as a simple example of a Test Report for a real project. A similar structure can be used in an actual project, but this report is prepared specifically for a test task.

This document outlines the REST API automation test framework and executed test scenarios for <https://petstore.swagger.io/#/>. The framework is implemented using **TestNG** to ensure reliable and flexible execution scenarios. There are verifications of three entities API endpoints: **Pet, Store, and User**.

**1. Test Framework**

* **Framework**: REST API Test Automation using **TestNG**
* **Tools & Libraries**: REST Assured - io.rest-assured (send/get request/response, for JSON serialization/deserialization), TestNG, Log4j
* **Logging & Reporting**: Log4j for logging
* **Test Structure**:
  + Each entity has a dedicated API client class
  + Test cases follow a structured approach: setup, execution, verification, and cleanup
  + Common validation methods for consistency across tests
  + There are comments/suggestions about improving framework or scenarios.

**2. Test Scenarios & Status**

**Pet API**

|  |  |
| --- | --- |
| **Test Case** | **Status** |
| Upload Image | Passed |
| Create Pet | Passed |
| Update Pet | Passed |
| Find by Status | Passed |
| Find by ID | Passed |
| Update in Store | Passed |
| Delete Pet | Passed |
| Find by Tag (Deprecated) | Skipped |

**Note:** there is afluent bug:

Actual Result: {code=1, type=error, message=**Pet not found**}

java.lang.AssertionError: Pet verification failed:

Category ID mismatch: expected=1, actual=0

Category Name mismatch: expected=Dogs, actual=null

Pet Name mismatch: expected=Buddy, actual=null

Photo URLs mismatch: expected=[https://example.com/photo1.jpg], actual=[]

Pet Status mismatch: expected=AVAILABLE, actual=null

Tag count mismatch: expected=2, actual=0

***Researching:*** Need to read server logs/speak with dev to investigate the issue.

**Store API**

|  |  |
| --- | --- |
| **Test Case** | **Status** |
| Inventory Check | Pending |
| Create New Order | Passed |
| Find by Order ID | Passed |
| Delete Order by ID | Passed |

**User API**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Status** | **Comments** |
| Create with List | Failed | java.lang.AssertionError: User verification failed:  Firstname mismatch: expected=firstname-1, actual=null  Lastname mismatch: expected=lastname-1, actual=null |
| Get by Username | Passed |  |
| Update User | Pending |  |
| Delete User | Passed |  |
| Login | Passed |  |
| Logout | Passed |  |
| Create with Array | Pending |  |
| Create User | Failed | java.lang.AssertionError: User verification failed:  Firstname mismatch: expected=firstname-1, actual=null  Lastname mismatch: expected=lastname-1, actual=null |

**3. Notes & Observations**

1. **TestNG Annotations Used:**
   * @BeforeMethod and @AfterMethod ensure cleanup (e.g., deleting created pets/users after tests).
   * @DataProvider for parameterized testing (e.g., multiple users/pets).
2. **Assertions & Validation:**
   * Custom verification methods for comprehensive field comparison.
   * Capturing all assertion failures instead of failing on the first mismatch.
3. **Pending Scenarios:**
   * **Update in Store (Pet API)**: Yet to be implemented.
   * **Inventory Check (Store API)**: API response format needs verification – need to have isolate system to test it in clear way.
   * **Update User (User API)**: Needs API payload clarification.
4. **Skipped Tests:**
   * find by tag for Pet API is deprecated and omitted from execution.

**4. Execution Summary**

* **Total Test Cases:** 20
* **Passed:** 14
* **Failed:** 2
* **Pending:** 3
* **Skipped:** 1

**Next Steps:**

* Create traceability matrix – map scenario to requirement/functionality.
* Add all scenarios into Jira.
* Implement pending test cases.
* Implement negative test cases: without mandatory fields, extra/duplicate fields, incorrect fields/values/ids/structure, etc. All these scenarios need to prepare by requirements, discuss with dev. team, business.
* Improve error handling and logging.
* Generate a full test execution report with Extent Reports.
* Cover scenarios via Selenium (GUI layer).
* CI/CD Integration: Automate test execution in pipelines.
* Performance Testing: Evaluate API response times under load.