Test Documentation - Petstore API Test Automation Framework

Project: RestAssured TestNG Petstore API Tests
Author: Utkarsh-Takmoge
Framework: RestAssured + TestNG + Maven
Target API: Swagger Petstore API (https://petstore.swagger.io/v2)
Date: August 11, 2025

Table of Contents

Project Overview

This project implements comprehensive API test automation for the Swagger Petstore API using Java, RestAssured, and TestNG. The framework follows a modular architecture using the Page Object Model (POM) design pattern for maintainability and scalability.

Key Features

- ✅ Multi-layer authentication support (API Key, Bearer Token, Basic Auth)
- ✅ Configurable test environment via properties files
- ✅ Extent Reports integration for detailed test reporting
- ✅ TestNG for test management and parallel execution
- ✅ JSON payload creation and validation
- ✅ Modular architecture with separation of concerns

Test Framework Architecture

Project Structure

bash
CopyEdit
```
src/
├── main/java/com/sprint/test/
│   └── App.java
└── test/
    ├── java/com/
    │   ├── baseSteps/
    │   │   ├── BaseSteps.java
    │   │   └── RequestSteps.java
    │   ├── parameter/
    │   │   └── PropertyReader.java
    │   ├── tests/
    │   │   └── Tests.java
    │   └── utils/
    │       └── ExtentManager.java
    └── PropertyFiles/
        └── Property.properties
```

Design Patterns Implemented

- **Page Object Model (POM)**
- **Factory Pattern**
- **Singleton Pattern**
- **Builder Pattern**

Test Environment Configuration

Base Configuration (Property.properties)

ini
CopyEdit
```ini
baseURL=https://petstore.swagger.io/v2

basepathPost=/pet
basepathGet=/pet/
basepathPut=/pet
basepathDelete=/pet/

auth.type=apikey
auth.apikey=special-key
auth.apikey.header=api_key
```

Supported Authentication Types

1. API Key Authentication ✅
2. Bearer Token Authentication ✅
3. Basic Authentication ✅
4. No Authentication ✅

Authentication Implementation

Current Implementation

- **Type**: API Key Authentication
- **Header**: `api_key`
- **Value**: `special-key`
- **Scope**: Applied to all API requests automatically

Authentication Flow

1. PropertyReader loads configuration
2. BaseSteps calls `setupAuthorization()`
3. Appropriate authentication applied
4. Auth headers included in all requests

Security Features

- ✅ Configurable authentication types
- ✅ Secure credential management
- ✅ Automatic header injection
- ✅ Error handling for missing credentials

Test Scenarios & Test Cases

Test Suite: Pet Management API Tests

Test Scenario 1: Pet Creation and Management

Test Case 1.1: Add New Pet to Store

- **Method**: `testAddPet()`
- **Priority**: 1
- **Type**: Functional Test
- **Description**: Verify pet creation
- **Pre-conditions**: Valid auth, API is up
- **Test Steps**:
    1. Create pet payload:
        - ID: 12345
        - Name: "Buddy"
        - Status: "available"
        - Category: Dogs (ID: 1)
        - Photo URLs: ["string"]
        - Tags: [{id: 1, name: "tag1"}]
    2. Send POST to `/pet`
    3. Validate response
- **Expected Result**: Status code 200, pet created
- **Test Data**:

json
CopyEdit
```json
{
  "id": 12345,
  "name": "Buddy",
  "status": "available",
  "category": { "id": 1, "name": "Dogs" },
  "photoUrls": ["string"],
  "tags": [{ "id": 1, "name": "tag1" }]
}
```

Test Scenario 2: Pet Retrieval Operations

Test Case 2.1: Get Pets by Status

- **Method**: `testGetPetsByStatus()`
- **Priority**: 4
- **Type**: Functional Test
- **Description**: Retrieve pets by "sold" status
- **Test Data**: status = "sold"

Test Scenario 3: Smoke Testing

Test Case 3.1: Basic Functionality Smoke Test

- **Method**: `smokeTest()`
- **Group**: smoke
- **Type**: Smoke Test
- **Test Data**:

json
CopyEdit
```
{
  "id": 999,
  "name": "SmokeTestDog",
  "status": "available"
}
```

Test Data Management

Static Test Data

- Pet IDs: 12345, 999
- Pet Names: "Buddy", "SmokeTestDog"
- Status Values: "available", "sold"
- Category: Dogs (ID: 1)

Dynamic Test Data Generation

- Via `createPetPayload()`
- Configurable attributes
- JSON schema validation

Reporting & Logging

Extent Reports

- Location: `target/Reports/ExtentReport.html`
- HTML spark reports with:
  - Timeline
  - Pass/fail details
  - Logs and exceptions

Console Logging

- API request/response
- Debug info
- Test progress

TestNG Reporting

- XML: `testng-results.xml`
- HTML: `index.html`, `emailable-report.html`
- JUnit XML: `junitreports/TEST-com.tests.Tests.xml`

Test Execution

Execution Order

1. `testAddPet()`
2. `testGetPetsByStatus()`
3. `smokeTest()` (group = smoke)

TestNG Configuration (testng.xml)

xml
CopyEdit
```
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="com.tests.Tests"/>
    </classes>
  </test>
</suite>
```

Execution Commands

bash
CopyEdit
```
mvn clean test
mvn test -Dtest=Tests
mvn test -Dgroups=smoke
```

Test Lifecycle

- `@BeforeClass`: Init
- `@BeforeMethod`: Setup
- `@Test`: Execute
- `@AfterMethod`: Log result
- `@AfterClass`: Cleanup

Dependencies

Core Libraries

- RestAssured: 5.4.0
- TestNG: 7.9.0
- ExtentReports: 5.1.2
- Jackson: 2.16.1
- Apache Commons Lang3: 3.14.0

Support Libraries

- JSON: 20240303
- Hamcrest: 2.2
- DataFaker: 2.1.0
- SLF4J + Logback

Test Coverage Analysis

API Endpoints Covered

- ✅ POST `/pet`
- ✅ GET `/pet/findByStatus`
- ❌ GET `/pet/{petId}`
- ❌ PUT `/pet`
- ❌ DELETE `/pet/{petId}`

HTTP Methods

- ✅ POST
- ✅ GET
- ❌ PUT
- ❌ DELETE

Test Types

- ✅ Functional
- ✅ Smoke
- ✅ Integration
- ❌ Negative
- ❌ Performance
- ❌ Security

Quality Metrics

Results

- Total Tests: 3
- Categories: Functional (2), Smoke (1)
- Pass Rate: 100%
- Auth Coverage: 100%

Code Quality

- ✅ Modular
- ✅ Configurable
- ✅ Logging
- ✅ Exception Handling

Future Enhancements

Test Cases

1. Full CRUD coverage
2. Negative tests (invalid IDs, bad JSON)
3. Boundary cases
4. Data-driven tests (CSV/Excel)
5. Error scenario simulations (timeouts, 5xx, 4xx)

Framework Enhancements

- DB Integration
- Parallel Testing
- CI/CD Setup
- Performance Tests
- API Contract Tests

Conclusion

This automation framework provides a strong foundation for testing the Petstore API. It supports robust architecture, flexible configuration, and is ready for scaling.