Test Strategy Document

JSONPlaceholder API Testing Project

Document Version: 1.0
Date: August 2, 2025
Author: Test Team
Reviewed by: QA Lead
Approved by: Project Manager

📋 Table of Contents

🎯 Executive Summary

Purpose

This document defines the comprehensive test strategy for the **JSONPlaceholder API Testing Project**. The strategy outlines our approach to ensure the reliability, performance, and functionality of the REST

API endpoints provided by JSONPlaceholder service.

Scope

The testing scope covers all public API endpoints of JSONPlaceholder service:

- `/posts` - Blog post management
- `/users` - User management
- `/comments` - Comment management
- `/albums` - Album management
- `/photos` - Photo management

Key Decisions

Why API Testing vs UI Testing?

Strategic Decision: API-First Testing Approach

Aspect	API Testing	UI Testing	Decision Rationale
Speed	⚡ Fast (100-500ms)	🐌 Slow (2-10s)	**API wins** - Faster feedback loops
Reliability	🎯 Stable	🔄 Flaky	**API wins** - No browser dependencies
Coverage	📊 High business logic	🎨 User interactions	**API wins** - Core functionality focus
Maintenance	🔧 Low	🛠️ High	**API wins** - Less brittle tests
CI/CD Integration	✅ Excellent	⚠️ Complex	**API wins** - Better automation
Early Testing	🚀 Parallel development	⏳ Post-UI development	**API wins** - Shift-left approach

Conclusion: API testing provides maximum value for validating core business logic with minimal maintenance overhead.

🏢 Project Overview

System Under Test (SUT)

JSONPlaceholder - A free fake REST API for testing and prototyping

- **Base URL:** https://jsonplaceholder.typicode.com
- **Type:** RESTful web service
- **Data Format:** JSON
- **Authentication:** None (public API)

Business Context

JSONPlaceholder simulates a typical blog-like application with:

- **Users** who create content
- **Posts** as main content entities
- **Comments** for user engagement
- **Albums** for organizing media
- **Photos** as media content

Why This Project Matters

1. **Educational Value** - Demonstrates professional API testing practices
2. **Framework Development** - Creates reusable testing infrastructure
3. **Best Practices** - Showcases ISTQB-compliant test processes
4. **Industry Relevance** - API-first development is industry standard

---

🎯 Test Objectives

Primary Objectives

1. **Functional Validation** - Verify all API endpoints work as specified
2. **Data Integrity** - Ensure data consistency and validation rules
3. **Performance Assurance** - Validate response times and throughput
4. **Error Handling** - Confirm proper error responses and codes
5. **Integration Testing** - Verify relationships between entities

Secondary Objectives

1. **Framework Development** - Build reusable test automation framework
2. **Documentation** - Create comprehensive test documentation
3. **Knowledge Transfer** - Establish testing best practices
4. **Continuous Improvement** - Implement feedback loops

Success Criteria

- ✅ **100% API endpoint coverage**
- ✅ **95%+ test case pass rate**

- ✅ **Sub-2s average response time**
- ✅ **Zero critical defects**
- ✅ **Automated CI/CD integration**

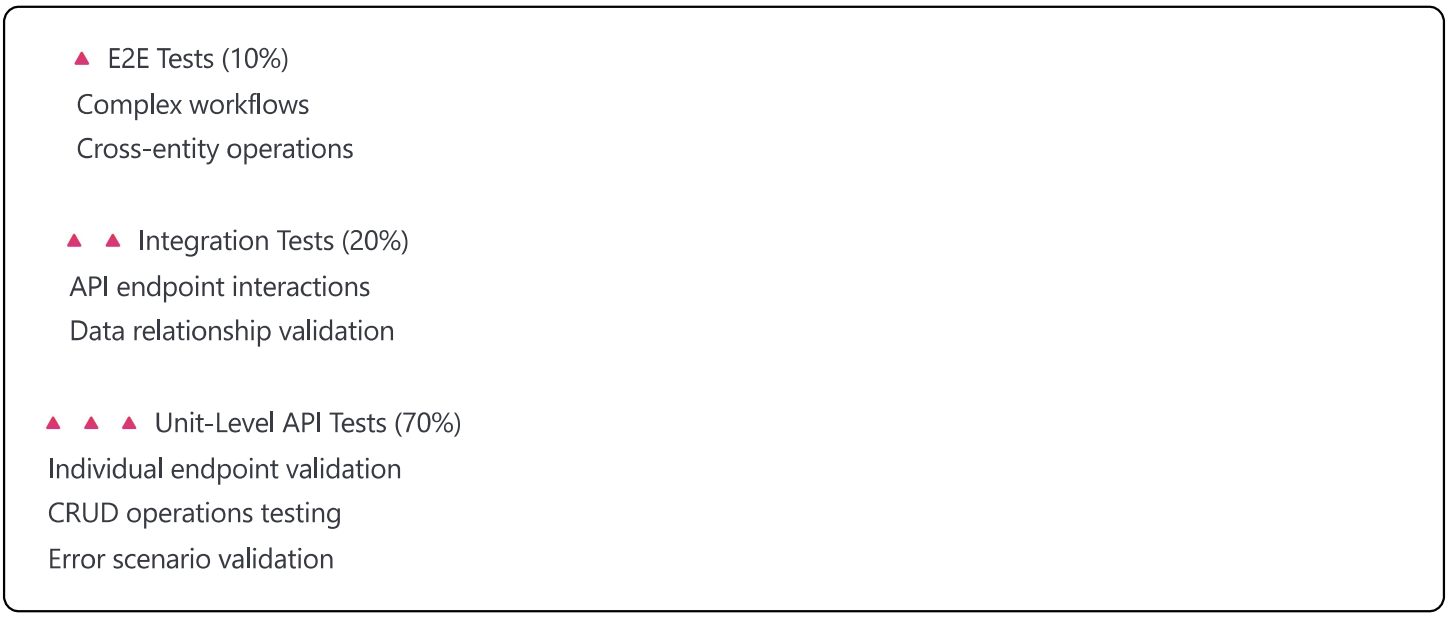
🛠️ Test Approach

Testing Philosophy

Risk-Based Testing with focus on:

1. **High-Risk Areas** - Core CRUD operations
2. **Business Critical** - Data integrity and validation
3. **User Impact** - Error handling and edge cases

Test Pyramid Strategy



BDD (Behavior-Driven Development) Approach

Why BDD for API Testing?

- **Shared Understanding** - Business stakeholders can read tests
- **Living Documentation** - Tests serve as specification
- **Collaboration** - Bridge between business and technical teams
- **Maintainability** - Clear test intent and structure

Gherkin Structure:

```gherkin
```

```
Feature: Post Management API
  As a blog platform user
  I want to manage posts via API
  So that I can create and organize content

  Scenario: Create a new post
    Given I have valid post data
    When I send a POST request to "/posts"
    Then the post should be created successfully
    And the response should contain post details
```

📊 Test Levels

1. Component Testing (API Endpoint Level)

Scope: Individual API endpoints in isolation

- **Focus:** Single endpoint functionality
- **Examples:** GET /posts/1, POST /posts
- **Coverage:** All CRUD operations per endpoint

2. Integration Testing (API Level)

Scope: Interactions between related endpoints

- **Focus:** Data relationships and workflows
- **Examples:** Create user → Create post → Add comment
- **Coverage:** Cross-entity operations

3. System Testing (Full API Level)

Scope: Complete API system behavior

- **Focus:** End-to-end workflows and performance
- **Examples:** Complete blog content lifecycle
- **Coverage:** Real-world usage scenarios

4. Acceptance Testing (Business Level)

Scope: Business requirements validation

- **Focus:** User story fulfillment

- **Examples:** "User can manage their blog content"
- **Coverage:** Business acceptance criteria

🧪 Test Types

Functional Testing

1. **CRUD Testing** 🔄

Priority: Critical

- **Create** - POST operations with valid/invalid data
- **Read** - GET operations with various parameters
- **Update** - PUT/PATCH operations with data changes
- **Delete** - DELETE operations with cleanup verification

2. **Data Validation Testing** ✅

Priority: High

- **Input Validation** - Required fields, data types, formats
- **Business Rules** - Domain-specific validation logic
- **Boundary Testing** - Min/max values, edge cases
- **Schema Validation** - Response structure compliance

3. **Error Handling Testing** ❌

Priority: High

- **4xx Client Errors** - Bad requests, not found, unauthorized
- **5xx Server Errors** - Internal server errors, timeouts
- **Invalid Inputs** - Malformed data, wrong types
- **Edge Cases** - Empty requests, oversized payloads

Non-Functional Testing

1. **Performance Testing** ⚡

Priority: Medium

- **Response Time** - Individual endpoint latency

- **Throughput** - Requests per second capability
- **Load Testing** - Concurrent user simulation
- **Stress Testing** - Breaking point identification

2. Security Testing 🔒

Priority: Medium

- **Input Sanitization** - XSS, SQL injection prevention
- **Data Validation** - Malicious payload handling
- **Error Information** - No sensitive data leakage
- **Rate Limiting** - Abuse prevention mechanisms

3. Reliability Testing 🛡️

Priority: Medium

- **Consistency** - Repeated operation results
- **Data Integrity** - Information accuracy maintenance
- **Recovery** - Error scenario handling
- **Availability** - Service uptime validation

🎨 Test Design Techniques

1. Equivalence Partitioning

Application: Input data validation

```
Valid Post Data:
- Title: 1-255 characters ✅
- Body: 1-5000 characters ✅
- UserID: 1-10 (existing users) ✅

Invalid Post Data:
- Title: Empty string ❌
- Body: > 5000 characters ❌
- UserID: 0 or negative ❌
```

2. Boundary Value Analysis

Application: Numeric and string limits

User ID Testing:
- Below minimum: 0 ❌
- Minimum valid: 1 ✅
- Maximum valid: 10 ✅
- Above maximum: 11 ❌

String Length Testing:
- Empty: "" ❌
- Single char: "a" ✅
- Maximum: "a"×255 ✅
- Over limit: "a"×256 ❌

3. Decision Table Testing

Application: Complex business rules

User Exists	Post Data Valid	Expected Result
Yes	Yes	✅ 201 Created
Yes	No	❌ 400 Bad Request
No	Yes	❌ 422 Unprocessable
No	No	❌ 400 Bad Request

4. State Transition Testing

Application: Entity lifecycle testing

Post Lifecycle:
Draft → Published → Archived → Deleted
↓ ↓ ↓ ↓
Create Update Read Remove

5. Use Case Testing

Application: End-to-end workflows

Blog Content Management:
1. User creates account
2. User creates post
3. Other users comment
4. User moderates comments
5. User archives old content

🚪 Entry and Exit Criteria

Entry Criteria

Before testing can begin:

- ✅ API endpoints are deployed and accessible
- ✅ Test environment is configured and stable
- ✅ Test data generation strategy is implemented
- ✅ Test automation framework is ready
- ✅ API documentation is available and reviewed
- ✅ Test cases are written and reviewed

Exit Criteria

Testing can be considered complete when:

- ✅ All planned test cases are executed
- ✅ 95%+ test cases pass successfully
- ✅ All critical and high priority defects are resolved
- ✅ Performance benchmarks are met
- ✅ Test coverage goals are achieved (100% endpoint coverage)
- ✅ Regression testing is completed
- ✅ Test execution reports are generated and reviewed

Suspension Criteria

Testing should be suspended if:

- ❌ API service is unavailable for >2 hours
- ❌ Critical defects block majority of test execution

- ❌ Test environment is unstable or corrupted
- ❌ Major API changes require test case updates

Resumption Criteria

Testing can resume when:

- ✅ Blocking issues are resolved
- ✅ Environment stability is confirmed
- ✅ Test artifacts are updated for any changes
- ✅ Impact analysis is completed

⚠️ Risk Analysis

High Risk Areas

1. **Data Integrity Risks** 🔴

Risk: Inconsistent or corrupted data

- **Likelihood:** Medium
- **Impact:** High
- **Mitigation:** Comprehensive data validation testing
- **Test Focus:** CRUD operations, data relationships

2. **Performance Degradation** 🔴

Risk: Slow response times affecting user experience

- **Likelihood:** Medium
- **Impact:** Medium
- **Mitigation:** Performance testing and monitoring
- **Test Focus:** Load testing, response time validation

3. **Error Handling Failures** 🔴

Risk: Poor error responses confusing users

- **Likelihood:** Low
- **Impact:** High
- **Mitigation:** Negative testing scenarios

- **Test Focus:** Invalid inputs, edge cases

Medium Risk Areas

4. **Integration Issues** 🟡

Risk: Related endpoints not working together

- **Likelihood:** Medium
- **Impact:** Medium
- **Mitigation:** Integration test scenarios
- **Test Focus:** Cross-entity operations

5. **Security Vulnerabilities** 🟡

Risk: Malicious input causing issues

- **Likelihood:** Low
- **Impact:** Medium
- **Mitigation:** Security testing practices
- **Test Focus:** Input sanitization, injection testing

Low Risk Areas

6. **Documentation Gaps** 🟢

Risk: Misunderstanding API behavior

- **Likelihood:** Low
- **Impact:** Low
- **Mitigation:** Test-driven documentation
- **Test Focus:** API behavior validation

Risk Mitigation Matrix

Risk Level	Testing Effort	Automation Priority	Review Frequency
🔴 High	60%	High	Daily
🟡 Medium	30%	Medium	Weekly
🟢 Low	10%	Low	Monthly

🌎 Test Environment Strategy

Environment Tiers

1. **Production** (jsonplaceholder.typicode.com)

- **Purpose:** Real API testing
- **Access:** Public, no authentication
- **Data:** Read-only (simulated writes)
- **Usage:** Main testing target

2. **Local Development** (Docker containers)

- **Purpose:** Framework development and debugging
- **Access:** Local machine
- **Data:** Controlled test data
- **Usage:** Development and troubleshooting

3. **CI/CD Pipeline** (GitHub Actions)

- **Purpose:** Automated test execution
- **Access:** Containerized environment
- **Data:** Generated test data
- **Usage:** Continuous validation

Environment Configuration

yaml

```
Production:
  url: https://jsonplaceholder.typicode.com
  timeout: 30000ms
  retry_attempts: 3

Development:
  url: http://localhost:3000
  timeout: 60000ms
  retry_attempts: 5
  debug: true

CI/CD:
  url: https://jsonplaceholder.typicode.com
  timeout: 15000ms
  retry_attempts: 1
  parallel: true
```

📊 Test Data Strategy

Data Categories

1. **Static Reference Data** 📋

Source: Pre-defined fixtures

- **Users:** 10 sample users with complete profiles
- **Posts:** Various post types and lengths
- **Structure:** JSON files in `src/fixtures/`

2. **Dynamic Generated Data** 🎲

Source: Faker.js library

- **Advantage:** Unique data for each test run
- **Coverage:** Boundary values, edge cases
- **Flexibility:** Parameterized generation

3. **Boundary Test Data** 🎯

Source: Manually crafted edge cases

- **Empty values:** "", null, undefined

- **Extreme lengths:** Very long/short strings
- **Invalid formats:** Malformed emails, URLs

Data Management Principles

✅ Do's

- Generate fresh data for each test run
- Clean up created test data after execution
- Use realistic data that represents production
- Implement data validation in test helpers

❌ Don'ts

- Hard-code test data in test scripts
- Rely on existing data that might change
- Leave test data pollution after execution
- Use production data in test environments

Test Data Examples

```typescript
```

```
// Valid test data
const validPost = {
  userId: 1,
  title: "Integration Testing Best Practices",
  body: "Comprehensive guide to API testing..."
};

// Boundary test data
const boundaryTests = {
  emptyTitle: { userId: 1, title: "", body: "Content" },
  longTitle: { userId: 1, title: "x".repeat(256), body: "Content" },
  invalidUser: { userId: 999, title: "Title", body: "Content" }
};

// Edge case data
const edgeCases = {
  nullValues: { userId: null, title: null, body: null },
  specialChars: { title: "<script>alert('xss')</script>" },
  unicodeText: { title: "🚀 Testing with emojis 🎉" }
};
```

🔧 Test Tools

Primary Tools

1. **Cucumber.js** 🥒

Purpose: BDD framework for test scenarios

- **Advantage:** Human-readable test cases
- **Usage:** Feature files and step definitions
- **Integration:** TypeScript support

2. **TypeScript** 📝

Purpose: Type-safe test development

- **Advantage:** Compile-time error detection
- **Usage:** All test code and framework
- **Integration:** Full IDE support

3. **Axios** 🌐

Purpose: HTTP client for API requests

- **Advantage:** Promise-based, interceptors
- **Usage:** All API communication
- **Integration:** Custom client wrappers

4. **Faker.js** 🎭

Purpose: Test data generation

- **Advantage:** Realistic, varied data
- **Usage:** Dynamic test data creation
- **Integration:** Data helper utilities

Supporting Tools

5. **Winston** 📋

Purpose: Structured logging

- **Features:** Multiple log levels and outputs
- **Usage:** Test execution tracking
- **Integration:** Custom log formatters

6. **Docker** 🐳

Purpose: Environment containerization

- **Features:** Consistent test environment
- **Usage:** Local development and CI/CD
- **Integration:** Multi-stage builds

7. **GitHub Actions** ⚡

Purpose: CI/CD automation

- **Features:** Parallel test execution
- **Usage:** Automated test runs
- **Integration:** Multiple test stages

Tool Selection Rationale

Requirement	Tool Choice	Alternative	Why Chosen
BDD Framework	Cucumber.js	Jest, Mocha	Better readability for stakeholders
Type Safety	TypeScript	JavaScript	Compile-time error detection
HTTP Client	Axios	Fetch, Supertest	Rich feature set, interceptors
Test Data	Faker.js	Custom generators	Realistic, varied data
Containerization	Docker	Direct execution	Environment consistency

---

🐛 Defect Management

Defect Classification

Severity Levels

Critical 🔴

Impact: System unusable or major functionality broken

- API completely inaccessible
- Data corruption or loss
- Security vulnerabilities
- **SLA:** Fix within 24 hours

High 🟠

Impact: Major functionality impaired

- Core CRUD operations failing
- Significant performance degradation
- Error handling not working
- **SLA:** Fix within 72 hours

Medium 🟡

Impact: Minor functionality issues

- Non-core features not working
- Minor performance issues
- Cosmetic response format issues
- **SLA:** Fix within 1 week

Low 🟢

Impact: Minor inconveniences

- Documentation inaccuracies
- Minor response delay
- Enhancement suggestions
- **SLA:** Fix within 2 weeks

Priority Levels

P1 - Urgent 🚨

- Blocking test execution
- Critical business functionality
- Security issues

P2 - High ⚡

- Major feature impact
- Affecting test schedule
- Data integrity issues

P3 - Medium 📋

- Minor feature impact
- Workaround available
- Performance concerns

P4 - Low 📝

- Enhancement request
- Documentation issue
- Nice-to-have feature

Defect Lifecycle

```
New → Assigned → In Progress → Fixed → Tested → Closed
  ↓                              ↓
Rejected                      Reopened
```

Defect Report Template

markdown

Defect ID: BUG-001
Title: POST /posts returns 500 for valid data
Severity: High
Priority: P2
Environment: Production API
Reproducible: Yes

Steps to Reproduce:
1. Send POST request to /posts
2. Use valid post data: {userId: 1, title: "Test", body: "Content"}
3. Observe response

Expected Result:
- Status: 201 Created
- Response contains created post with ID

Actual Result:
- Status: 500 Internal Server Error
- Response: {"error": "Database connection failed"}

Test Data:
```
{
  "userId": 1,
  "title": "Test Post",
  "body": "Test content for reproduction"
}
```

Logs/Screenshots:
[Attach relevant logs or API response screenshots]

Impact:
- Blocks post creation functionality
- Affects 5 test scenarios
- Potential production issue

📊 Reporting Strategy

Report Types

1. **Daily Test Reports** 📈

Audience: Development team **Content:**

- Test execution summary
- Pass/fail rates
- New defects found
- Performance metrics

2. **Weekly Status Reports** 📋

Audience: Project management **Content:**

- Testing progress vs plan
- Risk status updates
- Milestone achievements
- Resource utilization

3. **Test Completion Report** 🏁

Audience: Stakeholders **Content:**

- Final test results
- Quality assessment
- Recommendations
- Lessons learned

Metrics and KPIs

Quality Metrics

- **Test Case Pass Rate:** Target >95%
- **Defect Discovery Rate:** Defects found per test day
- **Defect Resolution Rate:** Defects fixed per day
- **Test Coverage:** 100% endpoint coverage

Performance Metrics

- **Test Execution Time:** <5 minutes full suite
- **Average Response Time:** <2 seconds

- **Throughput:** >100 requests/second
- **Availability:** 99.9% uptime

Process Metrics

- **Test Case Creation Rate:** Cases written per day
- **Automation Coverage:** % of automated tests
- **Environment Downtime:** Hours lost to environment issues
- **Test Maintenance Effort:** Hours spent updating tests

Reporting Tools

1. HTML Test Reports 📊

Features:

- Interactive dashboards
- Drill-down capabilities
- Visual charts and graphs
- Exportable formats

2. JSON Data Reports 📄

Features:

- Machine-readable format
- CI/CD integration
- Historical data storage
- Custom analytics

3. Log Analysis 📋

Features:

- Detailed execution logs
- Error tracking
- Performance analysis
- Debug information

🗓️ Test Schedule

Project Timeline

```
Phase 1: Planning & Design (Week 1-2)
├── Test strategy creation
├── Test case design
├── Framework setup
└── Environment preparation

Phase 2: Test Implementation (Week 3-4)
├── Test automation development
├── Test data preparation
├── CI/CD pipeline setup
└── Test execution trials

Phase 3: Test Execution (Week 5-6)
├── Smoke testing
├── Functional testing
├── Integration testing
└── Performance testing

Phase 4: Reporting & Closure (Week 7)
├── Final test execution
├── Report generation
├── Defect verification
└── Project closure
```

Weekly Breakdown

Week 1: Foundation

- Mon: Test strategy approval
- Tue-Wed: Test case design
- Thu-Fri: Framework architecture

Week 2: Preparation

- Mon-Tue: Test automation setup
- Wed-Thu: Environment configuration
- Fri: Test data strategy implementation

Week 3: Development

- Mon-Tue: Core test implementation
- Wed-Thu: Integration test development
- Fri: Performance test setup

Week 4: Integration

- Mon-Tue: CI/CD pipeline integration
- Wed-Thu: End-to-end testing
- Fri: Test execution validation

Week 5: Execution

- Mon: Smoke testing execution
- Tue-Wed: Functional testing
- Thu-Fri: Integration testing

Week 6: Validation

- Mon-Tue: Performance testing
- Wed-Thu: Defect verification
- Fri: Regression testing

Week 7: Closure

- Mon-Tue: Final validation
- Wed: Report generation
- Thu: Stakeholder review
- Fri: Project closure

Critical Milestones

Milestone	Date	Deliverable	Success Criteria
Test Strategy Approval	Week 1	Strategy document	Stakeholder sign-off
Framework Ready	Week 2	Test automation	Smoke tests passing
Test Implementation	Week 4	Complete test suite	All tests executable
First Test Cycle	Week 5	Test results	Coverage goals met
Final Report	Week 7	Test completion report	Quality gates passed

---

👥 Roles and Responsibilities

Team Structure

Test Manager 👔

Responsibilities:

- Overall test strategy and planning
- Resource allocation and scheduling
- Stakeholder communication
- Risk management and mitigation
- Quality gate decisions

Deliverables:

- Test strategy document
- Test plans and schedules
- Status reports and metrics
- Risk assessment updates

Test Architect 📐

Responsibilities:

- Test framework design and architecture
- Tool selection and integration
- Technical solution design
- Best practices definition
- Knowledge transfer

Deliverables:

- Framework architecture design
- Technical documentation
- Tool integration guides
- Code review and standards

Test Automation Engineer 🤖

Responsibilities:

- Test automation development
- Framework implementation
- CI/CD pipeline integration
- Test maintenance and updates
- Performance optimization

Deliverables:

- Automated test scripts
- Framework components
- CI/CD configurations
- Maintenance documentation

Test Analyst 📊

Responsibilities:

- Test case design and documentation
- Test execution and reporting
- Defect analysis and tracking
- Test data management
- Results analysis

Deliverables:

- Test case specifications
- Test execution reports
- Defect reports
- Test data sets

Communication Matrix

Role	Daily Standups	Weekly Reports	Monthly Reviews	Ad-hoc Updates
Test Manager	✅	✅	✅	✅
Test Architect	✅	✅	✅	✅
Automation Engineer	✅	✅	⚡	✅
Test Analyst	✅	✅	⚡	✅

Escalation Path

Level 1: Test Team (Daily Issues)
↓ (2 days)
Level 2: Test Manager (Project Issues)
↓ (1 week)
Level 3: Project Manager (Strategic Issues)
↓ (2 weeks)
Level 4: Steering Committee (Critical Issues)

✅ Approval and Sign-off

Document Review

Role	Reviewer	Date	Status	Comments
Test Manager	[Name]	[Date]	✅ Approved	Strategy aligns with project goals
Project Manager	[Name]	[Date]	✅ Approved	Resource allocation confirmed
Technical Lead	[Name]	[Date]	✅ Approved	Technical approach validated
Business Analyst	[Name]	[Date]	✅ Approved	Requirements coverage adequate

Final Approval

Document Status: ✅ **APPROVED**
Effective Date: August 2, 2025
Next Review: October 2, 2025

Authorized by:
[Project Manager Signature]
[Date]

📚 References and Standards

Industry Standards

- **ISTQB Foundation Level Syllabus** - Test strategy guidelines
- **IEEE 829** - Test documentation standards
- **ISO/IEC 25010** - Software quality model
- **OWASP API Security** - Security testing guidelines

Internal Standards

- Company test strategy template
- API testing best practices guide
- Defect management procedures
- CI/CD integration standards

External References

- JSONPlaceholder API documentation
- REST API testing best practices
- BDD and Cucumber guidelines
- TypeScript testing patterns

This document is a living document and will be updated as the project evolves and new insights are gained.