RYANAIR API TEST PLAN

Portfolio Project

Prepared by: Jean-Yves Garcin
Date: 13th November 2023

1. Introduction and Overview

- **Objective**: To comprehensively test and validate the functionalities, reliability, security, and performance of the Ryanair API.

- **Scope**: Covering major API functionalities related to flight search, booking, authentication, user interactions, error handling, and data integrity.

2. API Overview

- **Description**:

1. **Description of Ryanair API:**

Primary Functionalities:
- The API primarily caters to functionalities revolving around flight management, bookings, and passenger-related operations within the Ryanair ecosystem.

Supported Endpoints:
- Flight Booking: The API encompasses endpoints for flight booking operations, enabling users to search, select, and reserve flights for specific routes, dates, and passenger details.

- Passenger Management:
It provides endpoints related to passenger information management, allowing users to add, modify, or retrieve passenger details associated with booked flights.

Supported Methods:
- GET: Utilized for retrieving information, such as available flights, booking details, or passenger information.

- POST: Used for creating new bookings, adding passengers, or submitting flight reservation requests.

- PUT: Enables modification of existing bookings or passenger details.

- DELETE: Provides functionality to remove or cancel bookings or passenger information.

2. Authentication Mechanisms:

The Ryanair API does not utilize an API key for authentication. It's worth noting that while historically this API was accessible without the need for an API key, it has been discontinued. Therefore, it is possible that the current version or access to this API might not be officially supported by Ryanair.

Given the absence of an API key requirement and considering its discontinued status, it's essential to verify the authenticity and reliability of the endpoints, payloads, and responses during testing. Furthermore, additional communication or documentation might be necessary to understand the current status and access permissions for utilizing this API.

As part of the testing strategy, special attention will be given to validating the endpoints, data structures, error handling, and the overall behavior of the API without relying on an API key for authentication.

3. Supported Protocols:

1. RESTful Architecture:

- The API is likely built following RESTful principles, utilizing HTTP methods (GET, POST, PUT, DELETE) to interact with resources (endpoints) via uniform resource identifiers (URIs).

2. HTTP/HTTPS:

- The API predominantly operates over the HTTP or HTTPS protocols, offering secure and encrypted communication channels between clients and the server.

3. Test Strategy and Approach

- **Testing Types**: Specify functional testing, security testing, performance testing, error handling, and recovery testing.
- **Tools and Technologies**: Utilize tools like Postman, Newman, JMeter for functional, load, and security testing, JIRA Bug Tracking Tool

4. Test Scenarios, Test Cases & Inclusions

1. **Functional Testing:**
   - Verify the correctness and functionality of all API endpoints as per the API documentation.
   - Test various scenarios for booking creation, modification, and cancellation.
   - Validate user authentication and authorization mechanisms for protected endpoints.

2. **Data Validation Testing**:
   - Ensure that the API correctly validates input data, rejecting invalid requests.

- Test boundary values for input fields to check for any unexpected behavior.
- Validate the accuracy of data returned in responses.

3. **Error Handling Testing**:
   - Verify that appropriate error codes and messages are returned for invalid requests.
   - Check error responses for sensitive information disclosure.
   - Validate the API's ability to handle unexpected errors gracefully.

4. **Performance Testing**:
   - Assess the API's response time under normal and peak loads to identify potential bottlenecks.
   - Measure the API's throughput and scalability to handle concurrent requests.

5. **Security Testing**:
   - Conduct security assessments to identify vulnerabilities such as SQL injection, XSS, etc.
   - Validate the API's compliance with secure data transmission practices (e.g., HTTPS).
   - Check for proper access controls to prevent unauthorized access to sensitive resources.

6. **Integration Testing**:
   - Verify interactions between different API endpoints and services.
   - Test data consistency across related endpoints.

7. **Compatibility Testing**:
   - Test the API on different platforms, browsers, and devices to ensure cross-compatibility.

8. **Documentation Review**:
   - Assess the clarity, completeness, and accuracy of the API documentation.
   - Verify that the API documentation is in sync with the actual API behavior.

9. **Load Testing**:
   - Evaluate the API's behavior under high concurrent user loads to ensure stability.

10. **Regression Testing:**
    - Conduct regression testing after bug fixes or updates to ensure existing functionality remains intact.

11. **Edge Case Testing:**
    - Test extreme and boundary scenarios to identify potential issues.

12. **Concurrency Testing:**
    - Assess the API's behavior when multiple users attempt to access and modify bookings simultaneously.

13. **Ad Hoc Testing:**
    - Perform exploratory testing to identify any hidden defects or usability issues.

14. **Usability Testing:**
   - Evaluate the API's user-friendliness and ease of use from a developer's perspective.

15. **Continuous Integration and Deployment (CI/CD) Testing:**
   - Validate the API's behavior within the CI/CD pipeline to ensure smooth deployments.

16. **Performance Monitoring:**
   - Implement monitoring to track API performance in real-time.

17. **Backup and Recovery Testing:**
   - Validate data backup and recovery procedures to ensure data integrity.

18. **Internationalization Testing:**
   - Test the API's behavior with different language settings.

19. **Rate Limiting Testing:**
   - Check the API's adherence to rate-limiting rules to prevent abuse.

20. **Third-Party Integration Testing:**
   - Validate any third-party integrations for smooth functioning.

It's important to note that the scope of the test plan may evolve during the testing process based on feedback, changing requirements, or discoveries during testing. The scope should be reviewed and adjusted accordingly throughout the testing phase to ensure comprehensive coverage of the Ryanair API.

Inclusions:

Create (POST) Operations:

Test the API's ability to create new bookings using valid input data.
Verify that appropriate error responses are returned for invalid or missing data.
Validate that newly created bookings are stored correctly in the system.

Read (GET) Operations:

Test the API's ability to retrieve booking information by various criteria (e.g., booking ID, date range, etc).
Verify that the API returns the correct data in response to read requests.
Test for correct handling of non-existent or invalid booking IDs.

Update (PUT) Operations:

Test the API's ability to update existing bookings with valid data.
Verify that the API rejects invalid update requests with appropriate error responses.
Validate that the booking data is correctly modified in the system after updates.

Delete (DELETE) Operations:

Test the API's ability to delete bookings by providing valid booking IDs.
Verify that the API returns appropriate responses after successful deletion.
Validate that the deleted bookings are removed from the system.

Boundary Testing:

Test the API with minimum and maximum allowed values for input fields.
Validate the behavior of the API with values close to the boundaries.

Concurrency Testing:

Test the API's behavior when multiple users try to perform CRUD operations simultaneously.
Verify data consistency and handling of concurrent modifications.

Data Validation:

Test the API's response to various data validation scenarios (e.g., invalid characters, data types, mandatory fields).
Verify that the API handles validation errors appropriately.

Authentication and Authorization:

Test CRUD operations for both authenticated and unauthenticated users.
Verify that only authorized users can perform certain CRUD operations.

Error Handling:

Test the API's response when invalid or malformed requests are made for CRUD operations.
Validate that appropriate error codes and messages are returned.

Security Testing:

Test for security vulnerabilities during CRUD operations (e.g., SQL injection, XSS).
Verify that sensitive data is not exposed in responses.

Performance Testing:

Evaluate the API's response time for CRUD operations under normal and peak loads.
Measure the throughput and scalability of the API.

Integration Testing:

Verify the interaction and data consistency between CRUD operations and other API components.

Regression Testing:

Perform regression tests after bug fixes or updates to ensure existing CRUD functionalities remain intact.

Documentation Review:

Assess the accuracy of API documentation related to CRUD operations.

Load Testing:

Evaluate the API's behavior and performance during CRUD operations under high concurrent user loads.

Compatibility Testing:

Test the API's CRUD operations on different platforms, browsers, and devices.

Usability Testing:

Evaluate the ease of using CRUD functionalities from a developer's perspective.

Continuous Integration and Deployment (CI/CD) Testing:

Validate the CRUD operations within the CI/CD pipeline to ensure smooth deployments.

Rate Limiting Testing:

Check the API's adherence to rate-limiting rules for CRUD operations to prevent abuse.
Backup and Recovery Testing:

Validate data backup and recovery procedures for CRUD-related data.

5. Test Data, Environment Setup & Tools

- **Test Data Requirements**: Prepare test data for various scenarios (valid, invalid, boundary).

- **Environment Setup**: The operating systems and versions that will be used for testing, such as Windows 10, macOS, or Linux.

  The browsers and versions that will be tested, such as Google Chrome, Mozilla Firefox, or Microsoft Edge.

  The device types and screen sizes that will be used for testing, such as desktop computers, laptops, tablets, and smartphones.

  The network connectivity and bandwidth that will be available for testing, such as Wi-Fi, cellular, or wired connections.

The hardware and software requirements for running the test cases, such as a specific processor, memory, or storage capacity.

The security protocols and authentication methods that will be used to access the test environment, such as passwords, tokens, or certificates.

The access permissions and roles of the team members who will be using the test environment, such as testers, developers, or stakeholders.
Windows 10 – Chrome, Firefox and Edge

- Mac OS – Safari Browser
- Android Mobile OS – Chrome
- iPhone Mobile OS - Safari

6. Testing Approach

- **Test Execution Flow**:

Entry Criteria:
• Test Scenarios and Test Cases Documents are signed-off by the Client
• Application is ready for Testing

Exit Criteria:
• Test Case Reports, Defect Reports are ready

- **Automation Approach**: Automate repetitive test scenarios using Postman or other suitable tools.

- **Manual Testing Approach**: Identify areas that require manual testing, such as UI interactions or complex scenarios.

7. Reporting and Metrics

- **Test Reporting**:

1. Format of Test Reports:
   - Test reports will comprise detailed logs of executed test cases, including inputs, expected results, and actual outcomes.

2. Contents of Test Reports:

- Test execution summaries outlining the pass/fail status of test cases, along with detailed descriptions of defects encountered.

3. Defect Tracking:
   - Defect tracking will include comprehensive documentation of identified issues, their severity, steps to reproduce, and their resolution status.

4. Defect Reporting:

The criteria for identifying a defect, such as deviation from the requirements, user experience issues, or technical errors.

The **steps for reporting a defect**, such as using a designated template, providing detailed reproduction steps, and attaching screenshots or logs.

The **process for triaging and prioritizing defects, s**uch as assigning severity and priority levels, and assigning them to the appropriate team members for investigation and resolution.

The **tools and systems** that will be used for tracking and managing defects, such as a defect tracking software or a project management tool.

The **roles and responsibilities of the team members** involved in the defect reporting process, such as testers, developers, and the test lead.

The **communication channels** and frequencies for updating stakeholders on the progress and status of defects.

The metrics and metrics that will be used to measure the effectiveness of the defect reporting process, such as the number of defects found, the time taken to resolve them, and the percentage of defects that were successfully fixed.

DEFECT PROCESS	**POC**
New Frontend	Dev 1
Backend	Dev 2
Dev Ops	Dev 3

Tools: JIRA

- **Metrics**:

1. Response Times:
   - Measure and report API response times for various endpoints under different load conditions, indicating performance benchmarks and adherence to SLAs.

2. Error Rates:
   - Report the frequency and types of errors encountered during testing, categorizing them based on severity and impact.

3. Test Coverage:
   - Measure and report test coverage metrics, detailing the percentage of API endpoints and functionalities covered by test cases.

4. Security Vulnerabilities:
   - Report findings from security assessments, highlighting identified vulnerabilities, their severity levels, and steps taken for mitigation.

5. Compliance Metrics:
   - Measure the extent of compliance with industry standards and regulations (e.g., GDPR, API security best practices) and report any deviations.

8. Test Strategy

Step 1: Is to create test scenarios and test cases for the various features in
Scope.

While developing test cases, we'll use a number of test design techniques.

- Equivalence Class Partition
- Boundary Value Analysis
- Decision Table Testing
- State Transition Testing
- Use Case Testing

We also use our expertise in creating Test Cases by applying the below:

- Error Guessing
- Exploratory Testing
- We prioritize the Test Cases

Step 2: Our testing procedure when we receive a request for testing:

• First, we'll conduct smoke testing to see if the various and important functionalities of the application are working.

• We reject the build, if the Smoke Testing fails and will wait for the stable build before performing in depth testing of the application functionalities.

• Once we receive a stable build, which passes Smoke Testing, we perform in depth testing using the Test Cases created.

• Multiple Test Resources will be testing the same Application on Multiple Supported Environments simultaneously.

We then report the bugs in bug tracking tool and send dev. management the defect found on that day in a status end of the day email.

As part of the Testing, we will perform the below types of Testing:

- Smoke Testing and Sanity Testing
- Regression Testing and Retesting
- Usability Testing, Functionality & UI Testing
- We repeat Test Cycles until we get the quality product.

Step 3: We will follow the below best practices to make our Testing better:

• **Context Driven Testing** – We will be performing Testing as per the context of the given application.

• **Shift Left Testing** – We will start testing from the beginning stages of the development itself, instead of waiting for the stable build.

• **Exploratory Testing** – Using our expertise we will perform Exploratory Testing, apart from the normal execution of the Test cases.

• **End to End Flow Testing** – We will test the end-to-end scenario which involve multiple functionalities to simulate the end user flows.

9. Test Schedule

Following is the test schedule planned for the project –
Task Time Duration

TASK	DATES
▪ Creating Test Plan	
▪ Test Case Creation	
▪ Test Case Execution	
▪ Summary Reports Submission Date	

2 Sprints to Test the Application

10. Deliverables

The following are to be delivered to the client!

DELIVERABLES	DESCRIPTION	TARGET COMPLETION DATE
Test Plan	Details on the scope of	Date

	the Project, test strategy, test schedule, resource requirements, test deliverables and schedule	
Functional Test Cases	Test Cases created for the scope defined	Date
Defect Reports	Detailed description of the defects and identified along with screenshots and steps to reproduce on a daily basis	N/A
Summary Reports	Summary Reports – Bugs by Bug# Bugs by Function Area and Bugs by Priority	Date

11. Risks and Mitigation

- **Risk Analysis**:

1. API Downtime Risk:
   - Identify the risk of API service disruptions or downtime due to server issues, maintenance, or unexpected failures.

2. Security Vulnerabilities:
   - Recognize the potential for security vulnerabilities, such as injection attacks, authentication weaknesses, or insufficient data encryption.

3. Data Breaches:
   - Assess the risk of unauthorized access, data leaks, or breaches that could compromise sensitive information transmitted via the API.

- **Risk Mitigation**:

1. API Downtime Mitigation:
   - Implement redundancy measures, such as load balancing or failover mechanisms, to minimize the impact of API downtime.

2. Security Measures:
   - Employ robust authentication protocols (e.g., OAuth2, API keys) and implement encryption standards (TLS/SSL) to secure data transmission.

3. Data Breach Prevention:
   - Enforce access controls, encryption for sensitive data, and regular security audits to proactively prevent data breaches.

4. Contingency Planning:
   - Develop contingency plans outlining steps to mitigate risks promptly if they materialize, including rapid response procedures and backup/restoration protocols.

5. Regular Security Assessments:
   - Conduct regular security assessments and penetration testing to identify and rectify vulnerabilities before they are exploited.

6. Documentation and Training:
   - Ensure comprehensive documentation of security protocols and provide training to the development and testing teams regarding best practices and security measures.

7. Incident Response Plan:
   - Develop a structured incident response plan specifying roles, responsibilities, and actions in case of identified risks or security incidents.

12. Exit Criteria and Go/No Go discussion

- **Exit Criteria**:

1. Functional Coverage and Compliance:
   - Verify that all critical and high-priority functionalities are thoroughly tested, meeting acceptance criteria defined in the test plan and aligning with the API's intended functionality.

2. Regression Test Completion:
   - Ensure completion of regression testing, validating that existing functionalities remain intact after new feature implementations or modifications.

3. Performance and Scalability Assurance:
   - Confirm that the API performance metrics (response time, throughput) meet defined SLAs under varying loads, ensuring scalability and responsiveness.

4. Security and Compliance Verification:
   - Validate adherence to security standards (OWASP guidelines) and compliance with industry-specific regulations (GDPR, HIPAA, etc.), ensuring data protection and privacy.

5. Documentation Accuracy and Completeness:
   - Review and update API documentation to ensure alignment with the implemented functionalities, endpoints, and response formats.

- **Go/No Go discussion**:

1. Internal Stakeholder Review:
   - Internal stakeholders, including QA leads, project managers, and technical leads, conduct a thorough review of test artifacts (Test Plan, Test Scenarios, Test Cases, Reports) for accuracy and completeness.

2. Client Review and Approval:
   - Present the finalized Test Plan, Test Scenarios, Test Cases, and Reports to the client for their thorough review and approval.

3. Client Acceptance Criteria Fulfillment:
   - Testing will proceed to subsequent stages only upon the client's formal approval of the provided test artifacts, ensuring alignment with their acceptance criteria.

4. Formal Go/No Go Meeting:
   - Conduct a formal Go/No Go meeting involving both internal and client stakeholders to collectively acknowledge the completion of testing and readiness for the next phase.

5. Documented Go/No Go Confirmation:
   - Document the client's Go/No Go and approval in a formal document, ensuring a clear record of acceptance and readiness for subsequent development or deployment phases.

13. Glossary and References

- **Glossary**:

API
Definition: API stands for Application Programming Interface. It defines the protocols, tools, and definitions for building and interacting with software applications.

CRUD
Definition: CRUD stands for Create, Read, Update, Delete. It represents the basic operations used to manage data in a database or system.

Response Time

Definition: Response Time refers to the duration between sending a request to an API and receiving the complete response.

Endpoint

Definition: An endpoint refers to a specific URL or URI (Uniform Resource Identifier) that an API exposes to interact with particular functionalities or resources.

Payload

Definition: Payload represents the data transmitted in a request or response of an API. It contains the information being sent.

Authentication

Definition: Authentication is the process of verifying the identity of users or systems attempting to access the API, ensuring they have the required permissions.

- **References**:

Technical Standard: REST API Best Practices - by Swagger
https://swagger.io/resources/articles/rest-api-best-practices/
Description: This resource offers best practices for designing and testing REST APIs, influencing the design and validation approach in this test plan.

Security Guidelines: OWASP API Security Top 10
https://owasp.org/www-project-api-security/
Description: The OWASP API Security Top 10 provides insights into common API security risks and mitigation strategies, guiding the security testing approach for the API.

Testing Methodology: ISTQB Agile Testing Foundation
https://www.istqb.org/certification-path-root/agile-tester-extension-root.html
Description: ISTQB's Agile Testing Foundation influenced the Agile testing methodologies integrated into the API testing plan.

Tool Documentation: Newman - Postman Command Line Integration
*https://learning.postman.com/docs/running-collections/using-newman-*cli/command-line-integration-with-newman/
Description: Newman documentation guided the integration of Postman collections with command-line testing, enhancing automation strategies.