**Test Plan – Restful-Booker REST API**

**Created by: Dubbaka Lipi Goud**

**Objective**

The objective of this REST API test plan is to validate the functionality, security, and reliability of the Restful-Booker API (<https://restful-booker.herokuapp.com>). The API manages hotel bookings via endpoints for authentication, creating/retrieving/updating/deleting bookings, and health checks. It accepts HTTP requests (e.g., JSON/XML payloads) and returns responses in JSON or XML. Testing will ensure the API handles valid/invalid inputs correctly, enforces authentication, and maintains data integrity—returning appropriate HTTP status codes (e.g., 200 OK for success, 401 Unauthorized for auth failures).

**Scope**

Scope of Test Plan for the Restful-Booker REST API:

**Functional Testing**

* Validate correctness of API responses for all endpoints (e.g., token creation, booking CRUD).
* Ensure responses adhere to expected formats (JSON/XML) and data structures.

**Data Validation Testing**

* Validate input constraints (e.g., date formats, required fields like firstname / lastname).
* Test correct and incorrect data (e.g., invalid dates, missing fields, non-numeric prices).

**Error Handling Testing**

* Validate error responses for malformed requests, invalid parameters, or unauthorized access.
* Ensure API returns correct HTTP status codes (e.g., 400 Bad Request, 404 Not Found, 403 Forbidden).

**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.

**Security Testing**

* Verify authentication via token or Basic Auth for protected endpoints (PUT, PATCH, and DELETE).
* Check for vulnerabilities like unauthorized access or injection attacks on inputs.

**Integration Testing**

* Test end-to-end flows, such as creating a booking, updating it, and deleting it.

**Compatibility Testing**

* Validate API calls with different formats (JSON/XML) and tools (e.g., curl, Postman).

**Documentation Review**

* Ensure API behaviour aligns with the provided documentation (e.g., endpoint details, examples).

**Load Testing**:

* Evaluate the API's behaviour under high concurrent user loads to ensure stability.

**Regression Testing:**

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

**Edge Case Testing:**

* Test extreme and boundary scenarios to identify potential issues.

**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, guest name).
* 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 behaviour of the API with values close to the boundaries.

**Concurrency Testing:**

* Test the API's behaviour 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:**

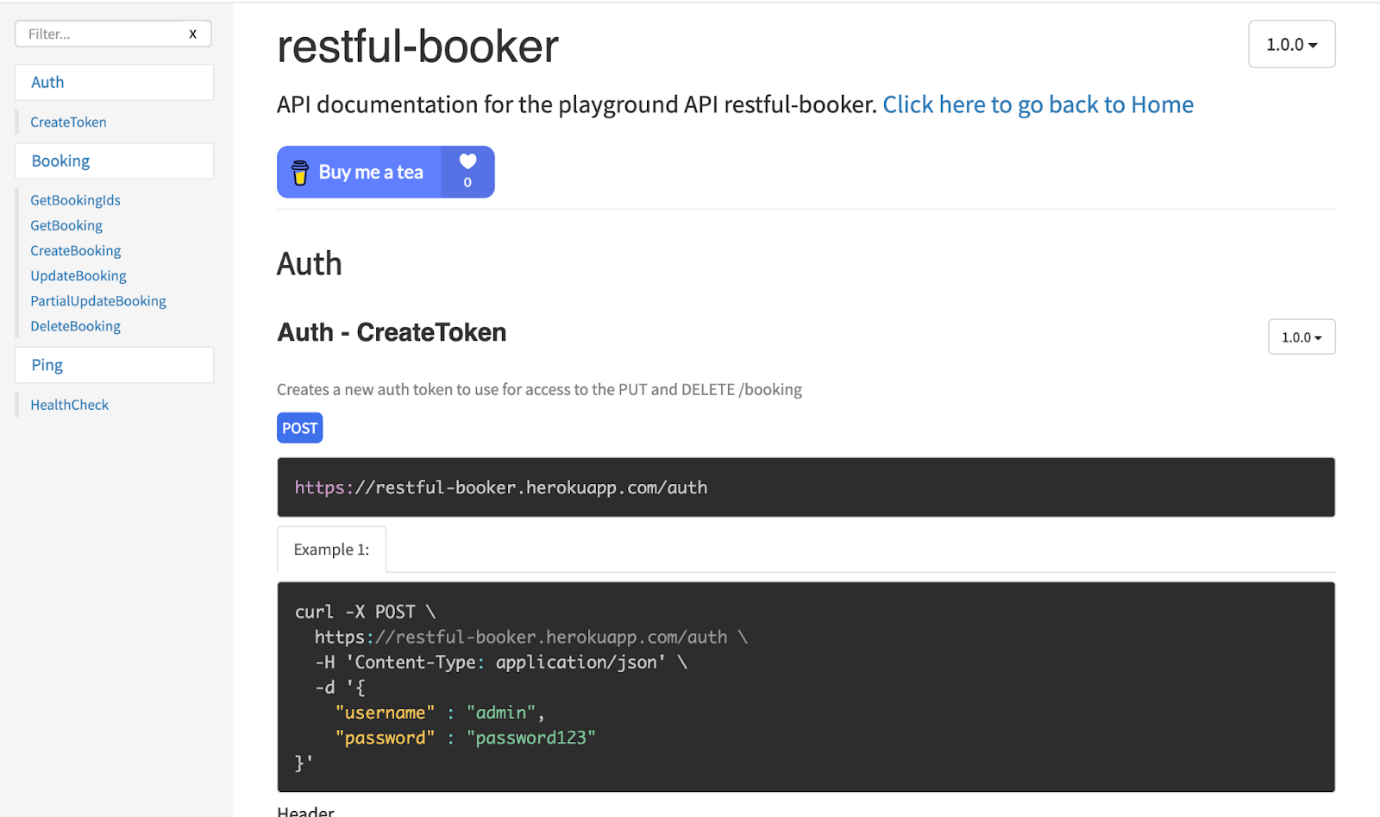
* Evaluate the API's behaviour and performance during CRUD operations under high concurrent user loads.

**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.



**Examples Covered:**

**Individual Test Cases**

* **Valid booking creation:** Send POST /booking with all info (like name "Jim", dates, price 111, paid true, extra needs "Breakfast"). Expect 200 OK with booking ID and details.
* **Invalid booking creation:** Send POST /booking missing key info (like no name or price). Expect 400 Bad Request or similar error.
* **Valid authentication:** Send POST /auth with right username ("admin") and password. Expect 200 OK with token.
* **Invalid authentication:** Send POST /auth with wrong username or password. Expect 200 OK with failure message or no token (API may return 200 even on failure per docs).
* **Token-based authorization:** Add a good token to update or delete (like in PUT). Expect 200 OK on success.
* **Unauthorized requests:** Try update or delete without token. Expect 403 Forbidden.
* Invalid or expired token: Use wrong/old token for update. Expect 403 Forbidden or 401 Unauthorized.
* **Filtering bookings:** Use GET /booking with filters (like date after "2025-07-29" or name + date). Expect 200 OK with matching list.
* **Retrieving single booking:** Use GET /booking/id with real ID. Expect 200 OK with full details.
* **Invalid ID handling:** Use GET /booking/fake-id. Expect 404 Not Found.
* **Full Change:** Use PUT /booking/id with token to change everything. Expect 200 OK with updated info.
* **Partial update**: Use PATCH /booking/id with token to change just one thing (like name). Expect 200 OK with changes.
* **Special characters and input sanitization:** Send names like "John@Doe" or with extra spaces. Expect 200 OK if handled, or 400 Bad Request if invalid.
* **Inputs exceeding limits:** Send very long text (like huge extra needs). Expect 200 OK if okay, or 400 Bad Request if too long.
* **Non-numeric or invalid type input:** Send price as "ABC" instead of number. Expect 400 Bad Request.
* **Invalid date formats:** Send wrong date like "not-a-date". Expect 400 Bad Request.
* **Malformed payloads:** Send messy JSON or XML. Expect 400 Bad Request.
* **Boundary testing:** Send extreme values like checking: "1900-01-01" or far-future dates. Expect 200 OK if within limits, or 400 Bad Request if not.
* **Boolean validation:** Send deposit paid as non-boolean (like string "true"). Expect 400 Bad Request or error if not handled.
* **Response format switching:** Set Accept: application/xml in GET /booking. Expect 200 OK with XML response instead of JSON.
* **Health check:** Use GET /ping. Expect 201 Created confirming API availability.

**End-to-End Flow Test Cases**

* **Full Booking Lifecycle:**

(1) Send POST /auth for token;

(2) Send POST /booking to create (like with "Test User");

(3) Use GET /booking/id to verify;

(4) Use PUT /booking/id (authenticated) to update (like total price);

(5) Use DELETE /booking/id (authenticated);

(6) Use GET /booking/id to confirm deletion.

Expect all steps 200/201 OK, final GET 404 Not Found.

* **Filtered Booking Search After Creation:**

(1) Send POST /booking to create multiple bookings;

(2) Use GET /booking with filters (like firstname = Test User & checking = 2025-07-29);

(3) Use GET /booking/id for a matched ID to verify details.

Expect 200 OK with accurate filtered results.

* **Authenticated Partial Update and Verification:**

(1) Send POST /auth for token;

(2) Send POST /booking to create;

(3) Use PATCH /booking/id (authenticated) to change additional needs (like to "Lunch");

(4) Use GET /booking/id to confirm;

(5) Try unauthorized PATCH.

Expect update 200 OK, unauthorized 403 Forbidden, changes visible in GET.

* **Error Handling in Flow with Invalid Data:**

(1) Send POST /auth for token;

(2) Send POST /booking with invalid data (like non-numeric total price);

(3) Use GET /booking with filters to check for data leaks;

(4) create valid booking then PUT with malformed JSON. Expect invalid steps return 400/500 errors, no unintended data.

* **Health Check Integrated with Booking Flow:**

(1) Use GET /ping to confirm up;

(2) Send POST /booking to create;

(3) Use DELETE /booking/id if created;

(4) Re-ping. Expect 201 Created on pings, flow aborts if ping fails, ensuring reliability.

**Test Environments**

|  |  |
| --- | --- |
| **Component** | **Description** |
| API URL | https://restful-booker.herokuapp.com/ |
| OS | Windows 10, macOS, Linux |
| Browsers | Chrome, Firefox, Edge, Safari |
| Devices | Desktop, Mobile (for API testing tools) |
| Tools Used | Postman, curl, JSON Validator |
| Network Connectivity and Bandwidth | Wi-Fi, cellular, or wired connections |
| Security Protocols and Authentication Methods | Passwords, Tokens, or Certificates |

**Defect Reporting Procedure**

• **Tool:** JIRA or similar (e.g., GitHub Issues if not specified).

• **Include:** Request method/payload/headers, response body/status code, timestamps, environment details, and screenshots.

• Prioritize by severity (e.g., Critical: API crashes; High: Auth failure; Medium: Invalid response format; Low: Cosmetic issues) and reproducibility.

• Defects reviewed by Test Lead and assigned to developers.

• **Severity & Priority** defined during triage meeting

**Test Strategy**

1. **Test Case Design**

* Use Equivalence partitioning (group similar inputs) & Boundary Value Analysis (test edges like min/max values).
* Include both positive (e.g., valid token creation) and negative (e.g., invalid ID in GET /booking/:id) test scenarios.

1. **Execution Flow**

* Smoke test: Check /ping for API availability.
* Proceed with Functional → Security & Negative Testing → Integration & Regression → Closure.

1. **Types of Testing**

* Smoke (basic endpoint reachability), Sanity (core functions like booking creation), Functional (all endpoints), Regression (re-test after fixes), Usability (ease of use per docs), Exploratory (ad-hoc invalid inputs).

1. **Best Practices**

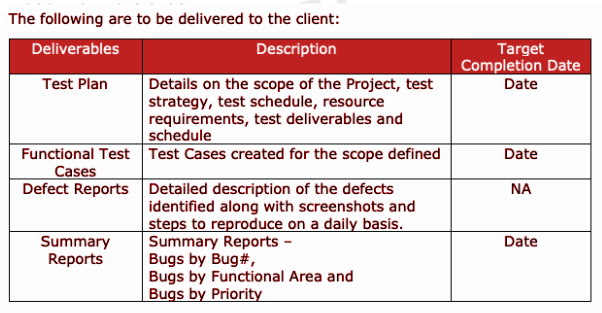
* Context-Driven Testing (adapt to API's playground nature, where data resets every 10 minutes).
* Shift-Left Testing (involve early reviews).
* End-to-End Flow Validation with Sample Requests (e.g., curl examples from docs).

**Test Schedule**

|  |  |
| --- | --- |
| **Task** | **Duration** |
| Test Plan Creation | Day 1 |
| Test Case Preparation | Day 2 |
| Execution & Defects | Day 3 – Day 4 |
| Summary Report | Day 5 |
| Sprint Duration | 1 Week Total |

**2 Sprints to Test the Application**

**Test Deliverables**



**Entry and Exit Criteria**

**Requirement Analysis**

* Entry: Access to API documentation & endpoint details.
* Exit: All requirements understood and documented.

**Test Execution**

* Entry: Test Cases Approved, API Reachable (via /ping).
* Exit: Test Logs & Defect Reports submitted, 100% coverage of in-scope items.

**Test Closure**

* Entry: Final Test Results Compiled.
* Exit: Test Summary Report Shared, all critical defects resolved.

**Tools**

* JIRA (Bug Tracking).
* Postman / curl (Test Execution for REST calls).
* Word/Excel (Test Case Documentation).
* Snipping Tool / Screenshot Tool.

**Risks and Mitigations**

|  |  |
| --- | --- |
| **Risk** | **Mitigation** |
| API Service Down / Endpoint downtime | Notify dev team, test on staging |
| Unexpected input format behaviour | Add comprehensive validation |
| Limited Access | Communicate with service owner early |
| Tight Timelines | Prioritize high-risk tests first |

**Approvals**

Team will send below documents for client or stakeholder approval:

* Test Plan.
* Test Scenarios & Cases.
* Execution Reports.
* Final Summary.