Booking API Testing Plan

Contents

[Test Plan 2](#_Toc198983616)

[1. Test Plan Overview 2](#_Toc198983617)

[1.1 Objectives 2](#_Toc198983618)

[1.2 Test Scenarios 2](#_Toc198983619)

[2. Tools and Setup 2](#_Toc198983620)

[3. Postman Setup and Implementation 3](#_Toc198983621)

[3.1 Test Idempotency (new collection folder) 3](#_Toc198983622)

[4. JMeter Setup and Implementation 4](#_Toc198983623)

[4.1 Detailed JMeter Script Design 4](#_Toc198983624)

[5.Database Consistency Check 5](#_Toc198983625)

[6. Run Load Test and Monitor 5](#_Toc198983626)

[7. Export Results & Reporting 5](#_Toc198983627)

[8. Metrics to Measure 5](#_Toc198983628)

# Test Plan

# 1. Test Plan Overview

## 1.1 Objectives

The primary objectives of this testing task are to validate the functionality, performance, and reliability of the booking API endpoint (POST /places) using both functional and load testing techniques.

## 1.2 Test Scenarios

The following test scenarios will be covered:

* **Valid Payloads:** Successful booking creation.
* **Invalid Payloads:** Missing fields, invalid date formats, duplicate bookings.
* **Load Testing:** Simulating 100 concurrent users for 5 minutes with a 30 s ramp-up.
* **Edge Cases:** Large payloads and special characters in user\_id.
* **Idempotency:** Re-sending the *exact same* booking request must **not** create a second record (expect a conflict error).
* **Database Consistency:** After execution, the total count of successful creations must equal the total number of records returned by a GET /places.

# 2. Tools and Setup

* **Postman:** Functional and error testing of the API.
* **Apache JMeter:** Load testing to simulate concurrent users and perform a high-level consistency check.

# 3. Postman Setup and Implementation

|  |  |
| --- | --- |
| Requirement | Details |
| Endpoint | POST https://682d014e4fae1889475497b9.mockapi.io/v1/api/senior-qc-test/booking/places |
| Tools | Postman (collection runner + environment vars) |
| Test Cases | - TC01: Valid Payload - TC02: Empty-String Fields - TC03: Invalid Date Format - TC04: Duplicate Booking - TC05: Large & Special Characters |
| Data Handling | Environment & collection variables; dynamic generation of IDs and timestamps |
| Assertions | - Valid → expect **201 Created** - Invalid → expect **400 Bad Request** or **409 Conflict** - JSON schema validation |
| Automation | Pre-request scripts to dynamic timestamps and randomized IDs |

## 3.1 Test Idempotency (new collection folder)

* **Request #1 – Duplicate Booking:**
  + Method: POST /places
  + Body: fixed user\_id & place\_id
  + Tests: pm.expect(pm.response.code).to.eql(201)
* **Request #2 – Send Duplicate Booking Again**
  + Re-uses the *exact same* body from Request #1
  + Tests: pm.expect(pm.response.code).to.eql(409)

# 4. JMeter Setup and Implementation

|  |  |
| --- | --- |
| Requirement | Details |
| Endpoint | Same POST /places URL |
| Users (Threads) | 100 |
| Ramp-up | 30 seconds |
| Duration | 5 minutes (300 seconds) |
| CSV Data | Columns: (user\_id, place\_id, date, VALID) ~97 valid rows + 3–4 invalid rows |
| Controllers | If-Controllers branching on VALID column from CSV file (Invalid vs. Valid) |
| Assertions | - Valid → expect 200/201 - Invalid → expect 400/409 |

## 4.1 Detailed JMeter Script Design

The JMeter script uses conditional logic based on input data from a CSV file to simulate multiple testing scenarios within one execution run, targeting valid and invalid payloads dynamically.

**Key components and flow:**

* **CSV Data Set Config:** Reads rows containing (user\_id, place\_id, createdAt) and a test type field indicating scenario type.
* **If Controllers:** Branch execution based on test type value:
  + 'valid' branch for valid booking requests
  + 'missing\_fields' branch for requests missing required fields
  + 'invalid\_date' branch for malformed date formats
  + 'duplicate' branch for duplicate booking attempts
* **HTTP Request Samplers:** Separate POST /places requests configured per branch using CSV variables.
* **Assertions:** Response validations scoped within each If Controller to check expected HTTP status codes (201 for valid, 400/409 for invalid).
* **Listeners:** View Results Tree and Summary Report collect test results per scenario.

# 5.Database Consistency Check

* **Track successful creations**:Attach a **JSR223 Post-Processor** to each **POST** sampler:
  + If response code is **201**, increment a thread-safe counter variable createdCount.
* **Fetch all records** by adding an **HTTP Request Sampler**: GET /places.
* **Count returned IDs** by Using a **Regular Expression Extractor** on the GET response body:
  + Match No.: **-1** (to count all matches) → store in id\_matchNr.
* **Compare counts** by Attaching a **JSR223 Assertion** to the GET sampler:
  + Compare createdCount vs. id\_matchNr.

# 6. Run Load Test and Monitor

* Average and 90th percentile response times.
* Error rates (HTTP 4xx/5xx).
* Throughput (requests per second).

# 7. Export Results & Reporting

* Summary.csv
* Aggregate.csv
* Results.jtl
* Screenshots for functional flows and idempotency

# 8. Metrics to Measure

* **Latency:** 95% of requests should respond within 2 seconds.
* **Error Rate:** Less than 3% for valid payloads.
* **Throughput:** Number of requests processed per second.
* **Idempotency:** 100 % of repeated POSTs return conflict
* **Database Consistency:** createdCount = id\_matchNr