Restful Booker API – Test Automation Documentation

Author: Kenan Bejtić
Date: 26.04.2025

Table of Contents

1. How to run the automated tests **3**
2. Objectives **4**
Verify Core Functionality 4
Ensure Service Reliability 4
Exercise Realistic User Flows 4
3. Core Functionality Tests **5**
3.1. Authentication Tests 5
3.2. Create Booking Tests 5
3.3. Read Booking Tests 6
3.4. Update Booking Tests 6
3.5. Delete Booking Tests 7
4. Smoke Tests **8**
5. Realistic User Flow Tests **9**
5.1. Full-Lifecycle Booking Flow 9
5.2. Search & Filtering Flow 10
6. Bugs **11**
6.1. Steps to reproduce the bug with checkin attribute 11
6.3. Table of bugs relating tocheckin attribute 13
6.4. Steps to reproduce the bug relating to the totalprice attribute 15
6.5. Table of bugs relating to the totalprice attribute 16

1. How to run the automated tests

Option 1:

Provided you have Python installed, open powershell and run the following command:
"Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass", then run the script "run_tests.ps1" inside the root of the folder where the script is located using the command:
".\run_tests.ps1" .
This will automatically install the necessary python libraries and run the tests.
The tests display 10 warnings at the end which are addressed in chapter 6 titled "Bugs" on page 11.

Option 2:

Provided you have Python installed, you will need to boot up any terminal and run the following command:

"pip install pytest requests"

After that is done you simply need to run " pytest -v "inside the root of the folder and all of the tests will start.

2. Objectives

Verify Core Functionality

- **CRUD Operations**: Validate Create (POST /booking), Read (GET /booking and GET /booking/{id}), Update (PUT & PATCH /booking/{id}), and Delete (DELETE /booking/{id}) behave as specified, returning correct status codes and data.
- **Authentication & Authorization**: Ensure valid credentials through POST /auth yield a usable token, and protected endpoints reject requests without a valid token or with an expired/invalid token.
- **Data Validation & Error Handling**: Confirm malformed or incomplete payloads return appropriate 4xx errors, and operations on non-existent resources return 404 Not Found (or agreed codes).

Ensure Service Reliability

- **Smoke Tests**: Establish a minimal suite (ping, authentication, create, read, delete) enabling quick detection of major service failures.

Exercise Realistic User Flows

- **Full-Lifecycle Booking**: Simulate a user journey from login, booking creation, full replace, partial update, retrieval, to deletion to ensure end-to-end integrity.
- **Search & Filtering**: Emulate multi-user scenarios by creating multiple bookings with shared attributes and querying via query parameters (e.g., firstname, lastname, checkin/out) to verify correct filtering.

3. Core Functionality Tests

3.1. Authentication Tests

Test Case	Description	Steps	Expected Result	Type
TC-Auth-01	Valid credentials	1. POST /auth with correct username/password	200 OK and response JSON contains a non-empty token	Positive
TC-Auth-02	Invalid credentials	1. POST /auth with incorrect username/password	200 OK and JSON body contains reason=\"Bad credentials\	Negative

3.2. Create Booking Tests

Test Case	Description	Steps	Expected Result	Type
TC-CR-01	Create booking (valid)	1. Obtain token via TC-Auth-01 2. POST /booking with a complete, valid JSON body including: firstname, lastname, totalprice, depositpaid, bookingdates, additionalneeds	200 OK and response body contains bookingid plus a booking object matching the request payload	Positive
TC-CR-02	Create booking (missing data)	1. POST /booking omitting a required field (e.g., lastname)	500 Bad Request with descriptive error message	Negative

3.3. Read Booking Tests

Test Case	Description	Steps	Expected Result	Type
TC-RD-01	Retrieve existing booking by ID	1. Create a booking via TC-CR-01 2. GET /booking/{id}	200 OK and response body contains booking details exactly as created	Positive
TC-RD-02	List bookings with filter (no match)	1. GET /booking?firstname=NonExistentName	200 OK and empty JSON array ([])	Negative
TC-RD-03	Retrieve non-existent booking ID	1. GET /booking/999999	404 Not Found	Negative

3.4. Update Booking Tests

Test Case	Description	Steps	Expected Result	Type
TC-UP-01	Full update (PUT) (valid)	1. Create booking via TC-CR-01 2. PUT /booking/{bookingid} with a completely new valid booking payload and valid token	200 OK and response JSON matches the new payload	Positive
TC-UP-02	Partial update (PATCH) (valid)	1. Create booking via TC-CR-01 2. PATCH /booking/{bookingid} with JSON containing one or two fields (e.g., firstname) and valid token	200 OK and only the patched fields are updated in the response	Positive
TC-UP-03	Unauthorized update	1. Create booking via TC-CR-01 2. PATCH /booking/{bookingid} without sending token	403 Forbidden	Negative

3.5. Delete Booking Tests

Test Case	Description	Steps	Expected Result	Type
TC-DEL-01	Delete existing booking (valid)	1. Create a booking via TC-CR-01 2. DELETE /booking/{bookingid} with valid token 3. GET /booking/{bookingid}	201 or 204 on DELETE; subsequent GET returns 404 Not Found	Positive
TC-DEL-02	Delete non-existent booking	1. DELETE /booking/999999 with valid token	404 Not Found or 405 Method Not Allowed	Negative
TC-DEL-03	Delete existing booking using basic authorization header	1. Create a booking via TC-CR-01 2.DELETE/booking/{ bookingid} with authentication header as an alternative to Cookie	201 or 204 on DELETE	Positive

4. Smoke Tests

Minimal critical-path validations to ensure service health.

Test Case	Description	Steps	Expected Result	Type
TC-SM-01	Health Check & Authentication	1. Send GET /ping 2. Send POST /auth with valid username/password	GET/ping returns Status 201, POST/auth returns Status 200, response contains token	Smoke
TC-SM-02	Create, Retrieve, and Delete Booking	1. Obtain token 2. Send POST /booking with valid payload and capture bookingid 3. Send GET /booking/{bookingid} 4. Send DELETE /booking/{bookingid} with token	POST returns Status 200 and bookingid, GET returns Status 200, payload matches creation, DELETE returns Status 201/204, subsequent GET returns 404	Smoke
TC-SM-03	List Bookings (Basic)	1. Send GET /booking	Status 200, returns a JSON array of booking IDs (at least including newly created ones)	Smoke

5. Realistic User Flow Tests

5.1. Full-Lifecycle Booking Flow

Test Case	Description	Steps	Expected Result	Type
TC-E2E-01	Full-Lifecycle Booking Flow	1. Auth: POST /auth (valid credentials) capture token 2. Create: POST /booking (valid payload, with token) capture bookingid 3. Verify Create: GET /booking/{bookingid} payload matches creation. 4. Full Replace: PUT /booking/{bookingid} (new payload, with token) 5. Verify Replace: GET /booking/{bookingid} the payload matches replacement 6. Partial Update: PATCH /booking/{bookingid} (e.g., change firstname, with token) 7. Verify Patch: GET /booking/{bookingid} only patched fields changed 8. Delete: DELETE /booking/{bookingid} (with token) 9. Verify Delete: GET /booking/{bookingid} returns 404 Not Found	Auth returns 200 +token, Create returns 200 +bookingid GET returns 200 + correct data PUT returns 200 + new data PATCH returns 200 + patched data DELETE returns 201/204 Final GET returns 404 Not Found	End-to-End

5.2. Search & Filtering Flow

Test Case	Description	Steps	Expected Result	Type
TC-E2E-02	Search & Filtering of Bookings	1. Create A: POST /booking (firstname="NameA", bookingdates A) 2. Create B: POST /booking (firstname="NameA", bookingdates B) 3. Create C: POST /booking (firstname="NameC") 4. Filter by firstname: GET /booking?firstname=NameA 5. Filter by lastname: GET /booking?lastname=<B_lastname> 6. Filter by dates: GET /booking?checkin=<A.checkin>&checkout=<A.checkout>	Filter firstname=NameA retruns an array that contains IDs of A & B only Filter lastname=B_lastname returns an array that contains the ID of B only Filter by A's dates returns an array that contains the ID of A only	End-to-End

6. Bugs

Issue: The API's ?checkin= filter does not return newly created bookings. Only the checkin-based query is affected. All other filters (firstname, lastname, checkout) work as expected.There is also a bug relating to the totalprice attribute that is returned when we query a booking. The tests relating to the bugs can be found inside /tests/test_bug.py.

6.1. Steps to reproduce the bug with checkin attribute

- **Authenticate**
  First we need to authenticate our session regularly.

  curl -s -X POST https://restful-booker.herokuapp.com/auth \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"password123"}

- **Create a Booking and capture the returned bookingid**

  curl -s -X POST https://restful-booker.herokuapp.com/booking \

  -H "Content-Type: application/json" \

  -H "Cookie: token=<token>" \

  -d '{

  "firstname":"BugTest",

  "lastname":"User",

  "totalprice":111,

  "depositpaid":false,

  "bookingdates":{

  "checkin":"2025-11-01",

  "checkout":"2025-11-05"

  },

  "additionalneeds":"None"

  }'

- **Apply the “checkin” filter**

  curl -s -X GET "https://restful-booker.herokuapp.com/booking?checkin=2025-11-01"

- **Inspect the response**

  There will be an array of { "bookingid": ... } objects. The newly created bookingid will not appear.

6.3. Table of bugs relating tocheckin attribute

Test Case	Description	Steps	Expected Result	Type
TC-BUG-01	Filter by checkin only	1.Create booking A 2.GET /booking?checkin=2025-11-01	200 OK, response array does NOT contain { "bookingid": <A> } (bug); booking A still exists	Bug
TC-BUG-02	Filter by firstname + checkin	1. Create booking A 2. GET /booking?firstname=ComboTest&checkin=2025-11-01	200 OK, response array does NOT contain { "bookingid": <A> } (bug); booking A still exists	Bug
TC-BUG-03	Filter by lastname + checkin	1. Create booking A 2. GET /booking?lastname=User&checkin=2025-11-01	200 OK, response array does NOT contain { "bookingid": <A> } (bug); booking A still exists	Bug
TC-BUG-04	Filter by firstname + lastname + checkin	1. Create booking A 2. GET /booking?firstname=ComboTest&lastname=User&checkout=2025-11-05	200 OK, response array does NOT contain { "bookingid": <A> } (bug); booking A still exists	Bug
TC-BUG-05	Filter by firstname + checkin + checkout	1. Create booking A 2. GET /booking?firstname=ComboTest&checkin=2025-11-01&checkout=2025-11-05	200 OK, response array does NOT contain { "bookingid": <A> } (bug); booking A still exists	Bug

TC-BUG-06	Filter by lastname + checkin + checkout	1. Create booking A 2. GET /booking?lastname=User&checkin=2025-11-01&checkout=2025-11-05	200 OK, response array does NOT contain { "bookingid": <A> } (bug); booking A still exists	Bug
TC-BUG-07	Filter by firstname + lastname + checkin + checkout	1. Create booking A 2. GET /booking?firstname=ComboTest&lastname=User&checkin=2025-11-01&checkout=2025-11-05	200 OK, response array does NOT contain { "bookingid": <A> } (bug); booking A still exists	Bug

6.4. Steps to reproduce the bug relating to the totalprice attribute

- **Authenticate**
  First we need to authenticate our session regularly.

  curl -s -X POST https://restful-booker.herokuapp.com/auth \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"password123"}

- **Create a Booking with a Floating-Point price**

  curl -s -X POST https://restful-booker.herokuapp.com/booking \

  -H "Content-Type: application/json" \

  -H "Cookie: token=<token>" \

  -d '{

  "firstname":"BugTest",

  "lastname":"Precision",

  "totalprice":123.45,

  "depositpaid":true,

  "bookingdates":{ "checkin":"2025-11-01","checkout":"2025-11-05" },

  "additionalneeds":"None"

  }'

  Capture the returned bookingid.

- **Retrieve the Booking**

  curl -s -X GET https://restful-booker.herokuapp.com/booking/{bookingid}

- **Inspect the response**

  Expected: "totalprice":123.45
  Actual: precision lost (e.g. "totalprice":123 or other integer)

6.5. Table of bugs relating to the totalprice attribute

Test Case	Description	Steps	Expected Result	Type
TC-BUG-08	Floating-point cents dropped (123.45 turns into 123)	1. Create booking with totalprice:123.45 2. GET /booking/{id}	200 OK, response JSON contains "totalprice":123.45	Bug
TC-BUG-09	Small fractional amount lost (0.99 turns into 0)	1. Create booking with totalprice: 0.99 2. GET /booking/{id}	200 OK, response JSON contains "totalprice":0.99	Bug