#	Test Name / Scenario	Input	Expected Result	Actual Status	Type of Test	Positive / Negative	Notes / Theory
1	GET /booking - should retrieve all bookings	GET request to `/booking`	200, JSON array of bookings	200	API	Positive	Public route, no auth required
2	GET /booking/{id} - should retrieve specific booking	GET request to `/booking/84`	200, JSON of booking with id 84	200	API	Positive	Valid ID returns booking object
3	GET /booking/{id} - should return 404 for non-existent booking	GET request to `/booking/99999`	404, Booking not found	404	API	Negative	Handles non-existent IDs
4	POST /booking - should create new booking without authentication	POST request with valid booking JSON	200, response includes `bookingid` and `booking` object	200	API	Positive	Public route, creates new booking
5	POST /booking - should fail with invalid booking data	POST request with invalid booking data (empty firstname,	400, validation error	400	API	Negative	Middleware `validateBooking` triggers validation

		negative totalprice, invalid types)					
6	POST /booking - should fail with missing required fields	POST request missing lastname and booking dates	400, validation error	400	API	Negative	Middleware catches missing required fields
7	PUT /booking/{id} - should update booking with authentication	PUT request with full updated booking JSON and valid auth token	200, updated booking object	200	API	Positive	Protected route, full update
8	PUT /booking/{id} - should fail without authentication	PUT request without token	403, auth token required	403	API	Negative	Middleware authenticateToken blocks request
9	PUT /booking/{id} - should fail with invalid token	PUT request with invalid token	403/404, blocked	200 (Failed)	API	Negative	Test expected 403/404 but endpoint accepted invalid token (implementation issue)
10	PATCH /booking/{id} - should	PATCH request with partial	200, booking updated only for	200	API	Positive	Partial update works

	partially update booking	data (`totalprice` + `additionalneeds`) and valid token	specified fields				
11	DELETE /booking/{id} - should delete booking with authentication	DELETE request with valid auth token	200, booking deleted	201	API	Positive	Endpoint returns 201 “Created” instead of 200 OK
12	DELETE /booking/{id} - should fail without authentication	DELETE request without token	403, auth token required	403	API	Negative	Auth middleware blocks deletion
13	DELETE /booking/{id} - should return 404 for non-existent booking	DELETE request to `/booking/99999`	404, booking not found	404	API	Negative	Correct handling of non-existent booking
14	Booking dates validation - checkin after checkout	POST booking with checkin after checkout	200 or 400 depending on validation	200	API	Positive/ Negative	Your API currently allows invalid dates
15	Complete booking lifecycle	POST → GET → PUT → DELETE	Each step succeeds; final deletion	1 failed (PUT invalid token),	API	Positive/ Negative	Full booking lifecycle tested; DELETE

		sequence	returns 200/verification	14 passed			not fully verified

💡 **Observations / Theory from results**:

- Middleware works correctly for **validation** and **auth** in most cases.
- The **PUT with invalid token** test fails because the API incorrectly allows updates with invalid token.
- DELETE returns **201 instead of 200**, which is unconventional but not critical.
- Invalid dates are accepted, which could be a validation gap.
- Complete booking lifecycle mostly passes, but token handling could be improved.

FLOW FOR TESTING .

EXAMPLE : Test: POST /booking

(From bookings.spec.js)

│

▼

[Playwright test sends POST request to /booking]

│

▼

```
[Express Route Matches]
router.post('/', validateBooking, createBooking)
   │
   ▼
[Middleware: validateBooking]
- Checks required fields:
   - firstname, lastname
   - totalprice, depositpaid
   - bookingdates (checkin, checkout)
- Checks data types (string, number, boolean)
- If invalid → returns 400 response → test asserts 400
- If valid → passes to controller
   │
   ▼
[Controller: createBooking]
- Extracts data from req.body
- Checks for missing fields again
- Determines next bookingid
   - Queries MongoDB for last bookingid
   - nextId = lastBooking.bookingid + 1
- Creates new booking object
- Inserts booking into MongoDB collection
- Returns response JSON:
{
   bookingid: nextId,
   booking: { ...booking details... }
}
   │
```

▼

[Test Receives Response]

- Test checks:

    - Status code (expect 200 in my case)

    - Response body has bookingid

    - Response body.booking fields match input

- Stores bookingid for later tests (update, delete)