API Test Scenarios

Booking Room

TC1: Verify Successful GET /booking (List Bookings)

- **Description**: Ensure GET /booking returns a list of booking IDs.
- **Preconditions**: None.
- **Steps**:
  1. In “Restful Booker API” collection, create a new request named “Get All Bookings”.
  2. **Request Setup**:
     - Method: GET
     - URL: {{base_url}}/booking
     - Headers: None
  3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response is an array of booking IDs", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData).to.be.an("array");
  if (jsonData.length > 0) {
    pm.expect(jsonData[0]).to.have.property("bookingid");
  }
});
pm.test("Content-Type is JSON", () => {
  pm.response.to.have.header("Content-Type", "application/json");
});
```

  4. Save and click “Send”.
- **Expected Result**:
  - Status: 200 OK
  - Response: Array like [{ "bookingid": 1 }, { "bookingid": 2 }, ...]
  - Tests pass.

TC2: Verify Successful GET /booking/:id (Get Booking Details)

- **Description**: Ensure GET /booking/:id returns the correct booking details.
- **Preconditions**: A booking exists (create one via POST /booking or use an existing booking_id).
- **Steps**:
  1. Create a new request named “Get Booking by ID”.
  2. **Request Setup**:
     - Method: GET
     - URL: {{base_url}}/booking/{{booking_id}}
     - Headers: None
  3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response has booking details", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData).to.have.property("firstname");
  pm.expect(jsonData).to.have.property("lastname");
  pm.expect(jsonData).to.have.property("totalprice");
  pm.expect(jsonData).to.have.property("depositpaid");
  pm.expect(jsonData).to.have.property("bookingdates");
});
```

  4. Save and send (ensure booking_id is set in the environment).
- **Expected Result**:
  - Status: 200 OK
  - Response: { "firstname": "John", "lastname": "Doe", ... }
  - Tests pass.

TC3: Verify Successful POST /booking (Create Booking)

- **Description**: Ensure POST /booking creates a booking and returns the booking details.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named “Create Booking”.
  2. **Request Setup**:
     - Method: POST
     - URL: {{base_url}}/booking
     - Headers: Content-Type: application/json
     - Body (Raw, JSON):

```
{
  "firstname": "John",
  "lastname": "Doe",
  "totalprice": 123,
  "depositpaid": true,
  "bookingdates": {
    "checkin": "2023-01-01",
    "checkout": "2023-01-05"
  },
  "additionalneeds": "Breakfast"
}
```

  3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response has bookingid and details", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData).to.have.property("bookingid");
  pm.expect(jsonData.booking).to.have.property("firstname").and.equal("John");
  pm.expect(jsonData.booking).to.have.property("lastname").and.equal("Doe");
});
pm.test("Store booking ID", () => {
  const jsonData = pm.response.json();
  pm.environment.set("booking_id", jsonData.bookingid);
});
```

  4. Save and send.
- **Expected Result**:
  - Status: 200 OK (Note: Docs suggest 200, though 201 is typical for creation)
  - Response: { "bookingid": <id>, "booking": { "firstname": "John", ... } }
  - booking_id stored in environment.
  - Tests pass.

TC5: Verify Successful POST /auth (Authentication)

- **Description**: Ensure POST /auth returns a valid token for correct credentials.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named "Authenticate".
  2. **Request Setup**:
     - Method: POST
     - URL: {{base_url}}/auth
     - Headers: Content-Type: application/json
     - Body (Raw, JSON):

```
{
  "username": "admin",
  "password": "password123"
}
```

  3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response has token", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData).to.have.property("token");
});
pm.test("Store auth token", () => {
  const jsonData = pm.response.json();
  pm.environment.set("auth_token", jsonData.token);
});
```

  4. Save and send.
- **Expected Result**:
  - Status: 200 OK
  - Response: { "token": "<token>" }
  - auth_token stored in environment.
  - Tests pass.

TC6: Verify Successful PUT /booking/:id (Update Booking)

- **Description**: Ensure PUT /booking/:id updates a booking with valid token.
- **Preconditions**: Valid token (from POST /auth), existing booking (from POST /booking).
- **Steps**:
  1. Create a new request named “Update Booking”.
  2. **Request Setup**:
     - Method: PUT
     - URL: {{base_url}}/booking/{{booking_id}}
     - Headers:
       - Content-Type: application/json
       - Accept: application/json
       - Cookie: token={{auth_token}}
     - Body (Raw, JSON):

```
{
  "firstname": "Jane",
  "lastname": "Doe",
  "totalprice": 456,
  "depositpaid": false,
  "bookingdates": {
    "checkin": "2023-02-01",
    "checkout": "2023-02-05"
  },
  "additionalneeds": "Dinner"
}
```

  3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response has updated booking details", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData.firstname).to.equal("Jane");
  pm.expect(jsonData.totalprice).to.equal(456);
});
```

  4. Save and send.
- **Expected Result**:
  - Status: 200 OK
  - Response: { "firstname": "Jane", "lastname": "Doe", ... }
  - Tests pass.

TC7: Verify Successful PATCH /booking/:id (Partial Update)

- **Description**: Ensure PATCH /booking/:id partially updates a booking.
- **Preconditions**: Valid token, existing booking.
- **Steps**:
    1. Create a new request named “Partial Update Booking”.
    2. **Request Setup**:
        - Method: PATCH
        - URL: {{base_url}}/booking/{{booking_id}}
        - Headers:
            - Content-Type: application/json
            - Accept: application/json
            - Cookie: token={{auth_token}}
        - Body (Raw, JSON):

```
{
  "firstname": "Johnny",
  "totalprice": 789
}
```

    3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response has partially updated details", () => {
  const json Normandy = pm.response.json();
  pm.expect(jsonData.firstname).to.equal("Johnny");
  pm.expect(jsonData.totalprice).to.equal(789);
});
```

    4. Save and send.
- **Expected Result**:
    - Status: 200 OK
    - Response: { "firstname": "Johnny", "lastname": "Doe", "totalprice": 789, ... }
    - Tests pass.

Edge Case Test Cases

TC8: Verify POST /booking with Missing Required Field

- **Description**: Ensure POST /booking with missing firstname returns 500 (per docs).
- **Preconditions**: None.
- **Steps**:
    1. Create a new request named “Create Booking Missing Firstname”.
    2. **Request Setup**:
        - Method: POST
        - URL: {{base_url}}/booking
        - Headers: Content-Type: application/json
        - Body (Raw, JSON):

```
{
  "lastname": "Doe",
  "totalprice": 123,
  "depositpaid": true,
  "bookingdates": {
    "checkin": "2023-01-01",
    "checkout": "2023-01-05"
  },
  "additionalneeds": "Breakfast"
}
```

    3. **Tests**:

```
pm.test("Status code is 500", () => {
  pm.response.to.have.status(500);
});
```

    4. Save and send.
- **Expected Result**:
    - Status: 500 Internal Server Error (per docs)
    - Response: Unspecified error message
    - Tests pass.

TC9: Verify POST /booking with Invalid Date Format

- **Description**: Ensure POST /booking with invalid checkin date returns 500.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named “Create Booking Invalid Date”.
  2. **Request Setup**:
     - Method: POST
     - URL: {{base_url}}/booking
     - Headers: Content-Type: application/json
     - Body (Raw, JSON):

```
{
  "firstname": "John",
  "lastname": "Doe",
  "totalprice": 123,
  "depositpaid": true,
  "bookingdates": {
    "checkin": "invalid-date",
    "checkout": "2023-01-05"
  },
  "additionalneeds": "Breakfast"
}
```

  3. **Tests**:

```
pm.test("Status code is 500", () => {
  pm.response.to.have.status(500);
});
```

  4. Save and send.
- **Expected Result**:
  - Status: 500 Internal Server Error
  - Tests pass.

TC10: Verify GET /booking/:id with Non-Existent ID

- **Description**: Ensure GET /booking/:id with invalid ID returns 404 Not Found.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named “Get Non-Existent Booking”.
  2. **Request Setup**:
     - Method: GET
     - URL: {{base_url}}/booking/999999
     - Headers: None
  3. **Tests**:
     ```
     pm.test("Status code is 404", () => {
       pm.response.to.have.status(404);
     });
     pm.test("Response body is Not Found", () => {
       pm.expect(pm.response.text()).to.equal("Not Found");
     });
     ```
  4. Save and send.
- **Expected Result**:
  - Status: 404 Not Found
  - Response: "Not Found"
  - Tests pass.

TC11: Verify POST /booking with Empty Body

- **Description**: Ensure POST /booking with empty body returns 500.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named “Create Booking Empty Body”.
  2. **Request Setup**:
     - Method: POST
     - URL: {{base_url}}/booking
     - Headers: Content-Type: application/json
     - Body: None (empty)
  3. **Tests**:
     ```
     pm.test("Status code is 500", () => {
       pm.response.to.have.status(500);
     });
     ```
  4. Save and send.
- **Expected Result**:
  - Status: 500 Internal Server Error
  - Tests pass.

Security Test Cases

TC12: Verify POST /auth with Invalid Credentials

- **Description**: Ensure POST /auth with wrong password returns 200 with empty response (per docs).
- **Preconditions**: None.
- **Steps**:
    1. Create a new request named “Auth Invalid Credentials”.
    2. **Request Setup**:
        - Method: POST
        - URL: {{base_url}}/auth
        - Headers: Content-Type: application/json
        - Body (Raw, JSON):

```
{
  "username": "admin",
  "password": "wrongpassword"
}
```

    3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response has no token", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData).to.not.have.property("token");
});
```

    4. Save and send.
- **Expected Result**:
    - Status: 200 OK
    - Response: {} or {"reason": "Bad credentials"}
    - Tests pass.

TC13: Verify PUT /booking/:id without Token

- **Description**: Ensure PUT /booking/:id without token returns 403 Forbidden.
- **Preconditions**: Existing booking.
- **Steps**:
  1. Create a new request named “Update Booking No Token”.
  2. **Request Setup**:
     - Method: PUT
     - URL: {{base_url}}/booking/{{booking_id}}
     - Headers: Content-Type: application/json
     - Body (Raw, JSON):

```
{
  "firstname": "Jane",
  "lastname": "Doe",
  "totalprice": 456,
  "depositpaid": false,
  "bookingdates": {
    "checkin": "2023-02-01",
    "checkout": "2023-02-05"
  },
  "additionalneeds": "Dinner"
}
```

  3. **Tests**:

```
pm.test("Status code is 403", () => {
  pm.response.to.have.status(403);
});
```

  4. Save and send.
- **Expected Result**:
  - Status: 403 Forbidden
  - Tests pass.

Admin Login

TC14: Verify Successful Admin Login with Valid Credentials

- **Description**: Ensure POST /auth with valid admin credentials returns a token.
- **Preconditions**: None.
- **Steps**:
  1. In the “Booking Room” folder of your “RB” collection, create a new request named “POST TC18: Verify Successful Admin Login”.
  2. **Request Setup**:
     - Method: POST
     - URL: {{base_url}}/auth
     - Headers: Content-Type: application/json
     - Body (Raw, JSON):

```
{
   "username": "admin",
   "password": "password123"
}
```

  3. **Tests**:

```
pm.test("Status code is 200", () => {
   pm.response.to.have.status(200);
});
pm.test("Response has token", () => {
   const jsonData = pm.response.json();
   pm.expect(jsonData).to.have.property("token");
   pm.expect(jsonData.token).to.be.a("string").and.not.empty;
});
pm.test("Store auth token", () => {
   const jsonData = pm.response.json();
   pm.environment.set("auth_token", jsonData.token);
});
```

  4. Save and send.
- **Expected Result**:
  - Status: 200 OK
  - Response: { "token": "<token>" }
  - auth_token stored in the environment.
  - Tests pass.

TC15: Verify Admin Can Access Bookings After Login

- **Description**: Ensure the admin can use the token to access the list of bookings via GET /booking.
- **Preconditions**:
    - Valid token (auth_token) stored in the environment (from TC18).
- **Steps**:

1. Create a new request named “GET TC19: Verify Admin Access to Bookings”.
2. **Request Setup**:
    - Method: GET
    - URL: {{base_url}}/booking
    - Headers: Cookie: token={{auth_token}} (Note: While GET /booking doesn’t require auth in this API, we include the token to simulate admin access as per the requirement).
3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response is an array of booking IDs", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData).to.be.an("array");
  if (jsonData.length > 0) {
    pm.expect(jsonData[0]).to.have.property("bookingid");
  }
});
```

4. Save and send.

- **Expected Result**:
    - Status: 200 OK
    - Response: Array like [{ "bookingid": 1 }, { "bookingid": 2 }, ...]
    - Tests pass.

TC16: Verify Invalid Admin Credentials Return an Error

- **Description**: Ensure POST /auth with invalid credentials returns an error response.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named “POST TC20: Verify Invalid Admin Credentials”.
  2. **Request Setup**:
     - Method: POST
     - URL: {{base_url}}/auth
     - Headers: Content-Type: application/json
     - Body (Raw, JSON):

```
{
  "username": "admin",
  "password": "wrongpassword"
}
```

  3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response indicates invalid credentials", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData).to.not.have.property("token");
  pm.expect(jsonData).to.have.property("reason").and.equal("Bad credentials");
});
```

  4. Save and send.
- **Expected Result**:
  - Status: 200 OK (per docs, though 401/403 would be more standard)
  - Response: { "reason": "Bad credentials" }
  - Tests pass.

TC17: Verify Missing Username in Admin Login

- **Description**: Ensure POST /auth with missing username returns an error.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named “POST TC21: Verify Missing Username in Admin Login”.
  2. **Request Setup**:
     - Method: POST
     - URL: {{base_url}}/auth
     - Headers: Content-Type: application/json
     - Body (Raw, JSON):

```
{
  "password": "password123"
}
```

  3. **Tests**:

```
pm.test("Status code is 400", () => {
  pm.response.to.have.status(400);
});
pm.test("Response indicates missing username", () => {
  const responseText = pm.response.text();
  pm.expect(responseText).to.include("Bad Request");
});
```

  4. Save and send.
- **Expected Result**:
  - Status: 400 Bad Request (based on typical API behavior; Restful Booker may vary)
  - Response: "Bad Request" (exact message may differ)
  - Tests pass.

TC18: Verify Missing Password in Admin Login

- **Description**: Ensure POST /auth with missing password returns an error.
- **Preconditions**: None.
- **Steps**:
    1. Create a new request named “POST TC22: Verify Missing Password in Admin Login”.
    2. **Request Setup**:
        - Method: POST
        - URL: {{base_url}}/auth
        - Headers: Content-Type: application/json
        - Body (Raw, JSON):

```
{
  "username": "admin"
}
```

    3. **Tests**:

```
pm.test("Status code is 400", () => {
  pm.response.to.have.status(400);
});
pm.test("Response indicates missing password", () => {
  const responseText = pm.response.text();
  pm.expect(responseText).to.include("Bad Request");
});
```

    4. Save and send.
- **Expected Result**:
    - Status: 400 Bad Request
    - Response: "Bad Request"
    - Tests pass.

TC19: Verify Token Usage for Protected Endpoint (PUT /booking/:id)

- **Description**: Ensure the token from a successful admin login can be used to update a booking via PUT /booking/:id.
- **Preconditions**:
  - Valid token (auth_token) in the environment.
  - Existing booking (booking_id) in the environment.
- **Steps**:

1. Create a new request named "PUT TC23: Verify Token Usage for Update Booking".
2. **Request Setup**:
   - Method: PUT
   - URL: {{base_url}}/booking/{{booking_id}}
   - Headers:
     - Content-Type: application/json
     - Accept: application/json
     - Cookie: token={{auth_token}}
   - Body (Raw, JSON):

```
{
  "firstname": "Jane",
  "lastname": "Doe",
  "totalprice": 456,
  "depositpaid": false,
  "bookingdates": {
    "checkin": "2023-02-01",
    "checkout": "2023-02-05"
  },
  "additionalneeds": "Dinner"
}
```

3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response has updated booking details", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData.firstname).to.equal("Jane");
  pm.expect(jsonData.totalprice).to.equal(456);
});
```

4. Save and send.

- **Expected Result**:
  - Status: 200 OK
  - Response: { "firstname": "Jane", "lastname": "Doe", ... }
  - Tests pass.

Cancelling a Booking

TC20: Verify Successful Cancellation of a Booking

- **Description**: Ensure DELETE /booking/:id with a valid booking ID and auth token deletes the booking and returns 201 or 204.
- **Preconditions**:
  - Valid token (auth_token) in the environment (from POST /auth).
  - Existing booking (booking_id) in the environment (from POST /booking).
- **Steps**:

1. In the “Booking Room” folder of your “RB” collection, create a new request named “DELETE TC24: Verify Successful Cancellation of Booking”.
2. **Request Setup**:
   - Method: DELETE
   - URL: {{base_url}}/booking/{{booking_id}}
   - Headers:
     - Cookie: token={{auth_token}}
3. **Tests**:

```
pm.test("Status code is 201 or 204", () => {
  pm.expect(pm.response.code).to.be.oneOf([201, 204]);
});
pm.test("Response body matches expected", () => {
  if (pm.response.code === 201) {
    pm.expect(pm.response.text()).to.equal("Created");
  } else {
    pm.expect(pm.response.text()).to.equal("");
  }
});
```

4. Save and send.

- **Expected Result**:
  - Status: 201 Created (per Restful Booker API docs) or 204 No Content
  - Response: "Created" (for 201) or empty body (for 204)
  - Tests pass.

TC21: Verify Booking is Removed After Cancellation

- **Description**: Ensure the booking no longer appears in GET /booking after deletion, simulating “removed from admin dashboard.”
- **Preconditions**:
    - Booking deleted via TC24.
    - booking_id of the deleted booking is stored in the environment.
- **Steps**:

1. Create a new request named “GET TC25: Verify Booking Removed After Cancellation”.
2. **Request Setup**:
    - Method: GET
    - URL: {{base_url}}/booking/{{booking_id}}
    - Headers: None
3. **Tests**:

```
pm.test("Status code is 404", () => {
  pm.response.to.have.status(404);
});
pm.test("Response indicates booking not found", () => {
  pm.expect(pm.response.text()).to.equal("Not Found");
});
```

4. Save and send.

- **Expected Result**:
    - Status: 404 Not Found
    - Response: "Not Found"
    - Tests pass.

TC22: Verify Cancellation with Invalid Booking ID

- **Description**: Ensure DELETE /booking/:id with an invalid booking ID returns an error.
- **Preconditions**:
    - Valid token (auth_token) in the environment.
- **Steps**:

1. Create a new request named “DELETE TC26: Verify Cancellation with Invalid Booking ID”.
2. **Request Setup**:
    - Method: DELETE
    - URL: {{base_url}}/booking/999999
    - Headers:
        - Cookie: token={{auth_token}}
3. **Tests**:

```
pm.test("Status code is 404 or 405", () => {
  pm.expect(pm.response.code).to.be.oneOf([404, 405]);
});
pm.test("Response indicates error", () => {
  pm.expect(pm.response.text()).to.be.oneOf(["Not Found", "Method Not Allowed"]);
});
```

4. Save and send.

- **Expected Result**:
    - Status: 404 Not Found or 405 Method Not Allowed (Restful Booker API behavior varies)
    - Response: "Not Found" or "Method Not Allowed"
    - Tests pass.

TC23: Verify Cancellation of Already Deleted Booking

- **Description**: Ensure DELETE /booking/:id for a booking that was already deleted returns an error.
- **Preconditions**:
  - Valid token (auth_token) in the environment.
  - Booking (booking_id) already deleted (from TC24).
- **Steps**:

1. Create a new request named "DELETE TC27: Verify Cancellation of Already Deleted Booking".
2. **Request Setup**:
   - Method: DELETE
   - URL: {{base_url}}/booking/{{booking_id}}
   - Headers:
     - Cookie: token={{auth_token}}
3. **Tests**:

```
pm.test("Status code is 404 or 405", () => {
  pm.expect(pm.response.code).to.be.oneOf([404, 405]);
});
pm.test("Response indicates error", () => {
  pm.expect(pm.response.text()).to.be.oneOf(["Not Found", "Method Not Allowed"]);
});
```

4. Save and send.

- **Expected Result**:
  - Status: 404 Not Found or 405 Method Not Allowed
  - Response: "Not Found" or "Method Not Allowed"
  - Tests pass.

TC24: Verify Cancellation Without Auth Token

- **Description**: Ensure DELETE /booking/:id without a token returns 403 Forbidden.
- **Preconditions**:
  - Existing booking (booking_id) in the environment.
- **Steps**:

1. Create a new request named “DELETE TC28: Verify Cancellation Without Auth Token”.
2. **Request Setup**:
   - Method: DELETE
   - URL: {{base_url}}/booking/{{booking_id}}
   - Headers: None (omit Cookie header)
3. **Tests**:

```
pm.test("Status code is 403", () => {
  pm.response.to.have.status(403);
});
```

4. Save and send.

- **Expected Result**:
  - Status: 403 Forbidden
  - Tests pass.

TC25: Verify Cancellation with Invalid Auth Token

- **Description**: Ensure DELETE /booking/:id with an invalid token returns 403 Forbidden.
- **Preconditions**:
  - Existing booking (booking_id) in the environment.
- **Steps**:

1. Create a new request named “DELETE TC29: Verify Cancellation with Invalid Auth Token”.
2. **Request Setup**:
   - Method: DELETE
   - URL: {{base_url}}/booking/{{booking_id}}
   - Headers:
     - Cookie: token=invalid_token
3. **Tests**:

```
pm.test("Status code is 403", () => {
  pm.response.to.have.status(403);
});
```

4. Save and send.

- **Expected Result**:
  - Status: 403 Forbidden
  - Tests pass.

Check Room Availability

TC26: Verify Check Room Availability with Valid Date Range

- **Description**: Ensure GET /booking with valid checkin and checkout parameters returns a list of bookings (or available rooms indirectly).
- **Preconditions**: None.
- **Steps**:
  1. In the "Booking Room" folder of your "RB" collection, create a new request named "GET TC30: Verify Check Room Availability with Valid Dates".
  2. **Request Setup**:
     - Method: GET
     - URL: {{base_url}}/booking?checkin=2023-01-01&checkout=2023-01-05
     - Headers: None
  3. **Tests**:

```
pm.test("Status code is 200", () => {
   pm.response.to.have.status(200);
});
pm.test("Response is an array", () => {
   const jsonData = pm.response.json();
   pm.expect(jsonData).to.be.an("array");
});
pm.test("Response time is less than 2000ms", () => {
   pm.expect(pm.response.responseTime).to.be.below(2000);
});
```

  4. Save and send.
- **Expected Result** (if API supports date filtering):
  - Status: 200 OK
  - Response: Array of bookings (e.g., [{ "bookingid": 1 }, ...]), ideally filtered by date range.
  - Response time < 2000ms.
  - Tests pass.
- **Actual Result** (if API does not support date filtering):
  - The API will likely ignore the query parameters and return all bookings, which we'll note as a limitation.

TC27: Verify JSON Response Includes Room Type, Availability Status, and Price

- **Description**: Ensure the response includes expected fields (room type, availability status, price) by fetching a booking's details after filtering.
- **Preconditions**:
  - A booking ID is available from TC30 or a previous POST /booking.
- **Steps**:

1. Create a new request named "GET TC31: Verify Room Details in Availability Response".
2. **Request Setup**:
   - Method: GET
   - URL: {{base_url}}/booking/{{booking_id}}
   - Headers: None
3. **Tests**:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
pm.test("Response includes booking details", () => {
  const jsonData = pm.response.json();
  pm.expect(jsonData).to.have.property("firstname");
  pm.expect(jsonData).to.have.property("totalprice"); // Price field
  pm.expect(jsonData).to.have.property("bookingdates");
});
pm.test("Response time is less than 2000ms", () => {
  pm.expect(pm.response.responseTime).to.be.below(2000);
});
```

4. Save and send.

- **Expected Result**:
  - Status: 200 OK
  - Response: { "firstname": "John", "lastname": "Doe", "totalprice": 123, "bookingdates": { "checkin": "2023-01-01", "checkout": "2023-01-05" }, ... }
  - Response time < 2000ms.
  - Tests pass.
- **Note**: The Restful Booker API does not have explicit "room type" or "availability status" fields. I'm using totalprice for price and inferring availability via bookingdates. If the API had a proper availability endpoint, it might return fields like room_type, is_available, and price_per_night.

TC28: Verify Error for Missing Checkin Date Parameter

- **Description**: Ensure GET /booking with missing checkin parameter returns an error or unfiltered results.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named "GET TC32: Verify Missing Checkin Date Parameter".
  2. **Request Setup**:
     - Method: GET
     - URL: {{base_url}}/booking?checkout=2023-01-05
     - Headers: None
  3. **Tests**:

```
pm.test("Status code is 400 or 200", () => {
  pm.expect(pm.response.code).to.be.oneOf([400, 200]);
});
pm.test("Response indicates error or unfiltered results", () => {
  if (pm.response.code === 400) {
    pm.expect(pm.response.text()).to.include("Bad Request");
  } else {
    const jsonData = pm.response.json();
    pm.expect(jsonData).to.be.an("array"); // Unfiltered results if API ignores missing param
  }
});
```

  4. Save and send.
- **Expected Result** (if API enforces parameter validation):
  - Status: 400 Bad Request
  - Response: "Bad Request"
  - Tests pass.
- **Actual Result** (if API does not validate):
  - Status: 200 OK
  - Response: Unfiltered array of bookings.
  - Tests pass.

TC29: Verify Error for Missing Checkout Date Parameter

- **Description**: Ensure GET /booking with missing checkout parameter returns an error or unfiltered results.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named “GET TC33: Verify Missing Checkout Date Parameter”.
  2. **Request Setup**:
     - Method: GET
     - URL: {{base_url}}/booking?checkin=2023-01-01
     - Headers: None
  3. **Tests**:

```
pm.test("Status code is 400 or 200", () => {
  pm.expect(pm.response.code).to.be.oneOf([400, 200]);
});
pm.test("Response indicates error or unfiltered results", () => {
  if (pm.response.code === 400) {
    pm.expect(pm.response.text()).to.include("Bad Request");
  } else {
    const jsonData = pm.response.json();
    pm.expect(jsonData).to.be.an("array"); // Unfiltered results
  }
});
```

  4. Save and send.
- **Expected Result** (if API enforces validation):
  - Status: 400 Bad Request
  - Response: "Bad Request"
  - Tests pass.
- **Actual Result** (if API does not validate):
  - Status: 200 OK
  - Response: Unfiltered array of bookings.
  - Tests pass.

TC30: Verify Error for Invalid Date Format

- **Description**: Ensure GET /booking with invalid date format returns an error or unfiltered results.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named "GET TC34: Verify Invalid Date Format".
  2. **Request Setup**:
     - Method: GET
     - URL: {{base_url}}/booking?checkin=invalid-date&checkout=2023-01-05
     - Headers: None
  3. **Tests**:

```
pm.test("Status code is 400 or 200", () => {
  pm.expect(pm.response.code).to.be.oneOf([400, 200]);
});
pm.test("Response indicates error or unfiltered results", () => {
  if (pm.response.code === 400) {
    pm.expect(pm.response.text()).to.include("Bad Request");
  } else {
    const jsonData = pm.response.json();
    pm.expect(jsonData).to.be.an("array"); // Unfiltered results
  }
});
```

  4. Save and send.
- **Expected Result** (if API enforces validation):
  - Status: 400 Bad Request
  - Response: "Bad Request"
  - Tests pass.
- **Actual Result** (if API does not validate):
  - Status: 200 OK
  - Response: Unfiltered array of bookings.
  - Tests pass.

TC31: Verify Response Time for Availability Check

- **Description**: Ensure the availability check response is returned in less than 2 seconds.
- **Preconditions**: None.
- **Steps**:
  1. Create a new request named “GET TC35: Verify Response Time for Availability Check”.
  2. **Request Setup**:
     - Method: GET
     - URL: {{base_url}}/booking?checkin=2023-01-01&checkout=2023-01-05
     - Headers: None
  3. **Tests**:

     ```
     pm.test("Response time is less than 2000ms", () => {
       pm.expect(pm.response.responseTime).to.be.below(2000);
     });
     ```
  4. Save and send.
- **Expected Result**:
  - Response time < 2000ms
  - Test passes.