## *DAY 10 – MOCKAPI & RESTFULBOOKER*

### 1. Token Generation with Invalid Credentials

- **Purpose:** Verifies that incorrect username/password combinations are **rejected**.
- Used the **Restful Booker API**
- **How:** Posts multiple invalid credential sets to the /auth endpoint and checks for a reason field in the response, implying failure.
- **Status Check:** Expects HTTP 200 (API returns success status even for failed auth attempts but includes failure details in body).

```
import { test, expect } from '@playwright/test';

test.describe('Token generation with invalid credentials', () => {
    const invalidCreds = [
      { username: 'admin', password: 'invalid' },
      { username: 'Admin', password: 'password123' },
      { username: 'admin', password: 'PASSWORD123' },
      { username: 'invalid', password: 'invalid123' }
    ];

    invalidCreds.forEach((creds) => {
        test(`should not authenticate user: ${creds.username} /
${creds.password}`, async ({ request }) => {
          const res = await request.post('/auth', {
            data: { username: creds.username, password: creds.password },
          });
          expect(res.status()).toBe(200);
          const body = await res.json();
          expect(body).toHaveProperty('reason');
          console.log('Response:', body);
        });
      });

  });
```

### 2. Booking Flow with Token

- **Token Creation:** Reads valid credentials from a JSON file (create_token.json) and retrieves an **auth token**.
- Used the **Restful Booker API**
- **Create Booking:** Posts a new booking with createBookingData, asserts creation, and checks for bookingid.

- **Retrieve Booking:** Retrieves the just-created booking by ID to verify data persistence.
- **Update Booking:** Uses the auth token to update a booking with updateBookingData.
- **Delete Booking:** Authenticates and deletes a booking. Checks for successful deletion (201 status).
- **Edge Case:** Tries to delete the same booking **twice**, expecting the second attempt to return 405 Method Not Allowed.

```
import { test, expect } from '@playwright/test';
import * as fs from 'fs';

const createData = JSON.parse(fs.readFileSync('./test-
data/createData.json', 'utf-8'));
const updateData = JSON.parse(fs.readFileSync('./test-
data/updateData.json', 'utf-8'));

let createdId: string;

test('POST /test/users - Create User', async ({ request }) => {
  const res = await request.post('/test/users', {
    data: createData
  });
  expect(res.status()).toBe(201);
  const body = await res.json();
  expect(body).toHaveProperty('id');
  createdId = body.id;
  console.log('Response:', body);
});

test('GET /test/users/:id - Retrieve Created User', async ({ request }) =>
{
  const res = await request.get(`/test/users/${createdId}`);
  expect(res.status()).toBe(200);
  console.log('Response:', await res.json());
});

test('PUT /test/users/:id - Update Created User', async ({ request }) => {
  const res = await request.put(`/test/users/${createdId}`, {
    data: updateData
  });
  expect(res.status()).toBe(200);
  console.log('Response:', await res.json());
});
```

```
test('DELETE /test/users/:id - Delete Created User', async ({ request }) =>
{
  const res = await request.delete(`/test/users/${createdId}`);
  expect(res.status()).toBe(200);
  console.log('Response:', await res.json());
});

test('DELETE /test/users/:id again - Ensure User is Gone', async ({ request
}) => {
  const res = await request.delete(`/test/users/${createdId}`);
  expect(res.status()).toBe(404);
  console.log('Response:', await res.json());
});
```

### 3. User CRUD Operations

- Used the **Restful Booker API & MOCKAPI**
- **Create User:** Posts user data (createData) to /test/users and expects a 201 Created with an id.
- **Read User:** Uses the id to fetch the user and expects a 200 OK.
- **Update User:** Sends an update to the same user ID with updateData, expects success.
- **Delete User:** Deletes the user by ID, expects 200 OK.
- **Edge Case:** Deletes the same user **again** and expects a 404 Not Found.