End-to-End REST API Automation

A practical reference for building maintainable API tests throughout the Playwright TypeScript course.

Trainer:	MD Zaman
Course:	Playwright TypeScript Automation (Testing Track)
Handout Date:	January 08, 2026

How to use this handout

Use the checklist as your definition of “done” for API automation. Then follow the definitions and examples as a reusable pattern for every API module and every project you automate.

1) End-to-end REST API Automation Checklist

1.	API scope and endpoints identified (what to test, what not to test)
2.	Base URL and environment strategy (dev/qa/stage) is config-driven
3.	Authentication method confirmed (token/OAuth/basic/API key) and secrets are not hardcoded
4.	Test data strategy defined (unique data per run, no collisions)
5.	Standard request builder and headers (content-type, auth, correlation id)
6.	Response validation standards (status, key fields, business rules; schema if available)
7.	Negative tests included (401/403/404/400/validation/rate limit where applicable)
8.	Cleanup and idempotency handled (delete created records or isolate data)
9.	API–UI integration flows supported (seed data via API, verify via UI)
10.	Timeouts and resilience set (avoid flaky tests; retries only where justified)
11.	Logging and evidence (safe attachments/snippets without exposing secrets)
12.	Reporting output enabled (HTML and/or JUnit)
13.	CI readiness (headless runs, env vars, secrets handling, artifacts)
14.	Standards and reusability (folders, naming, helpers, documentation)

Recommended project structure (API + UI)

```
tests/
  api/
  ui/
utils/
  auth.ts
  headers.ts
  data.ts
playwright.config.ts
```

Starter commands: npx playwright test • npx playwright test tests/api • npx playwright show-report
Tip: Keep API tests fast and independent. Avoid shared records and always cleanup created data.

2) Definitions and Examples (Playwright + TypeScript)

All examples assume baseURL is set in playwright.config.ts and you run tests with: npx playwright test

Scope and endpoints

Definition: Decide which endpoints are in-scope for automation (critical business flows). Document what is out-of-scope or mocked.

```
// Example in-scope endpoints:
POST /auth/login
POST /users
GET  /users/{id}
DELETE /users/{id}
```

Environment strategy (baseURL)

Definition: Tests must run against different environments without code changes by using environment variables and config.

```
// playwright.config.ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  use: {
    baseURL: process.env.API_BASE_URL || 'https://api.example.com',
  },
});
```

Authentication (token example)

Definition: Confirm the auth method and keep secrets in environment variables (never hardcode). Return a token for reuse.

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

export async function getToken(request: APIRequestContext) {
  const res = await request.post('/auth/login', {
    data: { username: process.env.API_USER, password: process.env.API_PASS },
  });
  expect(res.ok()).toBeTruthy();
  const body = await res.json();
  return body.token as string;
}
```

Standard headers

Definition: Centralize headers such as Authorization and Content-Type so every test stays consistent and easy to update.

```
export function authHeaders(token: string) {
  return {
    Authorization: `Bearer ${token}`,
    'Content-Type': 'application/json',
  };
}
```

Test data (unique per run)

Definition: Create unique test data to avoid collisions and flaky failures due to shared environments.

```
export function uniqueEmail(prefix = 'student') {
  return `${prefix}.${Date.now()}@example.com`;
}
```

Create user: status + field validation

Definition: Validate more than status code: key fields, types, and business rules. Keep assertions meaningful.

```
import { test, expect } from '@playwright/test';
import { getToken } from '../utils/auth';
import { authHeaders } from '../utils/headers';
import { uniqueEmail } from '../utils/data';

test('POST /users creates a user', async ({ request }) => {
  const token = await getToken(request);
  const email = uniqueEmail();

  const res = await request.post('/users', {
    headers: authHeaders(token),
    data: { name: 'Student One', email },
  });

  expect(res.status()).toBe(201);

  const body = await res.json();
  expect(body).toMatchObject({ name: 'Student One', email });
  expect(typeof body.id).toBe('string');
});
```

Negative test: unauthorized

Definition: Confirm the API fails correctly and returns meaningful errors (example: 401 without token).

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

test('GET /users returns 401 without token', async ({ request }) => {
  const res = await request.get('/users');
  expect(res.status()).toBe(401);

  const body = await res.json();
  expect(body).toHaveProperty('message');
});
```

Cleanup: create then delete

Definition: Make tests repeatable by cleaning up created data or isolating test records.

```
import { test, expect } from '@playwright/test';
import { getToken } from '../utils/auth';
import { authHeaders } from '../utils/headers';
import { uniqueEmail } from '../utils/data';

test('Create and cleanup user', async ({ request }) => {
  const token = await getToken(request);

  const createRes = await request.post('/users', {
    headers: authHeaders(token),
    data: { name: 'Temp User', email: uniqueEmail() },
  });
  expect(createRes.status()).toBe(201);

  const created = await createRes.json();
  const userId = created.id;

  const delRes = await request.delete(`/users/${userId}`, {
    headers: authHeaders(token),
  });
  expect([200, 204]).toContain(delRes.status());
});
```

API → UI integration (seed via API, verify in UI)

Definition: Use API calls to create test data quickly, then validate the user experience via UI tests.

```
import { test, expect } from '@playwright/test';
import { getToken } from '../utils/auth';
import { authHeaders } from '../utils/headers';
import { uniqueEmail } from '../utils/data';

test('API seeds user; UI verifies', async ({ request, page }) => {
  const token = await getToken(request);
  const email = uniqueEmail('seeded');

  const createRes = await request.post('/users', {
    headers: authHeaders(token),
    data: { name: 'Seeded User', email },
  });
  expect(createRes.status()).toBe(201);

  await page.goto('/users');
  await expect(page.getByText(email)).toBeVisible();
});
```

Evidence and safe logging

Definition: Attach helpful evidence for debugging without exposing credentials or secrets.

```
// Attach a safe snippet (avoid tokens/passwords)
const bodyText = await res.text();
test.info().attach('api-response', {
  body: bodyText,
  contentType: 'text/plain',
});
```

Reporting (HTML + JUnit)

Definition: Enable stakeholder-friendly reporting and CI integration.

```
// playwright.config.ts
reporter: [
  ['html'],
  ['junit', { outputFile: 'results/junit.xml' }],
],
```

CI readiness (environment variables)

Definition: Use environment variables for baseURL and secrets. Ensure headless runs and artifacts.

```
# Example CI environment variables
API_BASE_URL=https://api.example.com
API_USER=your_user
API_PASS=your_password

# Run command
npx playwright test
```

Quick acceptance criteria for every API test

Each API test should: (1) state intent in the test name, (2) validate status plus key fields, (3) be repeatable (unique data and/or cleanup), (4) produce evidence on failure, and (5) run in CI using env vars.