Comprehensive API Testing PoC Report

Overview

For our API testing framework in TypeScript using the Playwright Test runner, we evaluated three different HTTP client libraries:

1. **Playwright's Native API Request**
2. **Supertest**
3. **Axios**

Each library was assessed based on several key factors: language compatibility, API protocol support, ease of use, testing capabilities (including retries, timeouts, and assertions), integration with reporting (Allure), advanced features like mocking, scalability, and community support. Our tests are executed via the Playwright Test runner, using its built-in expect assertions to maintain a unified environment.

---

1. Language & Framework Compatibility

Explanation

- **Playwright Native API:** Fully TypeScript-compatible, with built-in type definitions and no additional setup required.
- **Supertest:** Originally designed for JavaScript; requires `@types/supertest` for full TypeScript support.
- **Axios:** Fully TypeScript-friendly with out-of-the-box type definitions.

Concrete Data

Library	GitHub Stars	npm Weekly Downloads
Playwright (@playwright/test)	~70,000	~10 million
Supertest	~14,000	~4.8 million
Axios	~101,000	~48 million

Code Examples

Playwright Native API:

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

test('Playwright API Test - TypeScript Native', async ({ request }) => {
  const response = await request.get('/endpoint', { timeout: 5000 });
  await expect(response.status()).toBe(200);
  const data = await response.json();
  await expect(data).toHaveProperty('key');
});
```

Supertest:

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

const baseURL = 'https://api.example.com';

test('Supertest API Test - TypeScript Compatible', async () => {
  const response = await supertest(baseURL)
    .get('/endpoint')
    .timeout({ deadline: 5000 });
  expect(response.status).toBe(200);
  expect(response.body).toHaveProperty('key');
});
```

Axios:

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

const axiosInstance = axios.create({
  baseURL: 'https://api.example.com',
  timeout: 5000,
});

test('Axios API Test - Fully TypeScript Friendly', async () => {
  const response = await axiosInstance.get('/endpoint');
  expect(response.status).toBe(200);
  expect(response.data).toHaveProperty('key');
});
```

---

2. API Protocol Support

All three libraries support RESTful API testing and JSON payloads. Additional support for GraphQL and SOAP can be integrated.

3. Testing Capabilities

Explanation

- **Playwright Native API:** Integrates seamlessly with Playwright's `expect` assertions.
- **Supertest:** Fluent API for HTTP testing; supports `expect` for assertions.
- **Axios:** Requires manual assertions; best suited for general HTTP requests rather than testing.

Code Examples

Playwright Native API:

```
const response = await request.put('/user/123', { data: { name: 'New Name' } });
await expect(response.status()).toBe(200);
```

Supertest:

```
const response = await supertest(baseURL)
  .put('/user/123')
  .send({ name: 'New Name' })
  .expect(200);
```

Axios:

```
const response = await axiosInstance.put('/user/123', { name: 'New Name' });
expect(response.status).toBe(200);
```

---

4. Integration & Reporting

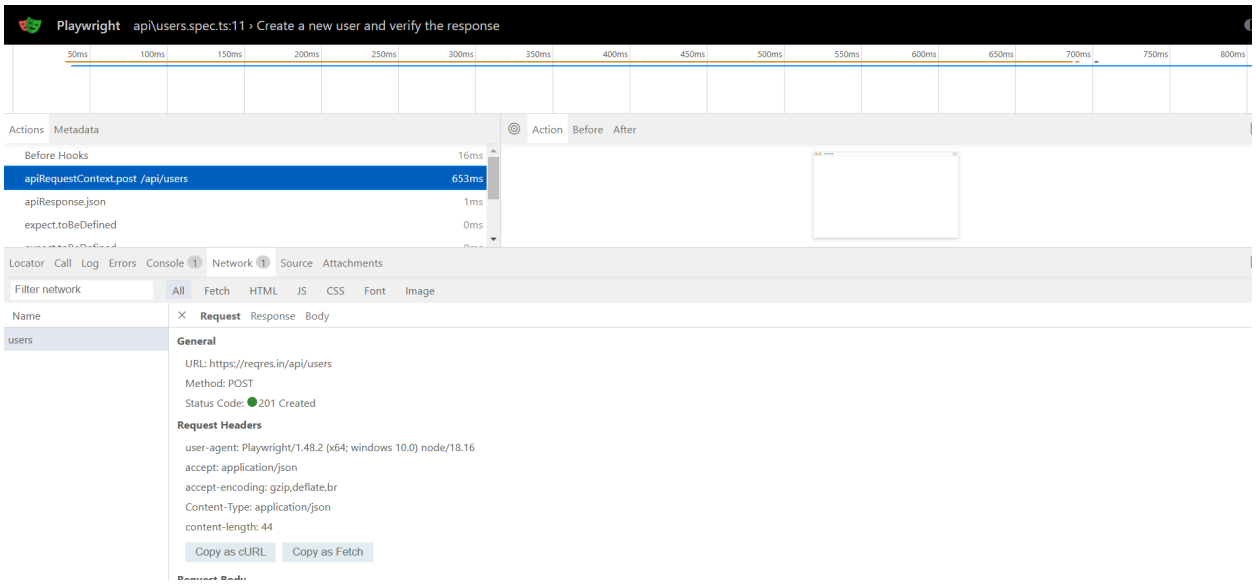
Explanation

- **Playwright Native API:** Built-in library and reporting capabilities for network traces.
- **Supertest & Axios:** Need to install libraries and lack of network work traces capabilities.

Playwright Native API

This has built in support for network traces in report as Playwright runner knows the api calls.

a) Playwright report

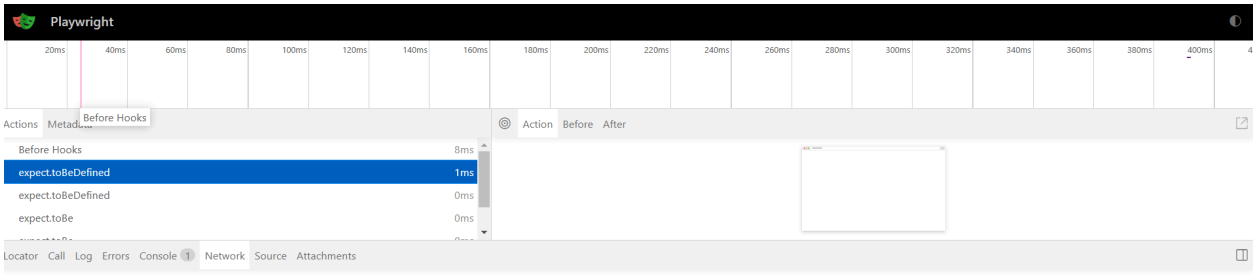


b) Allure report

Supertest & Axios

No support to network traces as Playwright runner doesn't know api calls.

a) Playwright report



b) Allure report

---

5. Mocking Purpose

a) Playwright Request Mocking

API-based mocking (when using Playwright's `request` object) typically requires an external intercept library (e.g., Nock) because the `request` object is effectively Node.js's HTTP client under the hood.

```
test('Mock external HTTP call using Nock', async () => {
  // 1. Create a Playwright request context (fixture).
  const context = await request.newContext();

  // 2. Use Nock to intercept the request to the given domain and path.
  nock('https://jsonplaceholder.typicode.com')
    .get('/todos/1')
    .reply(200, {
      userId: 1,
      id: 1,
      title: 'Mocked Title',
      completed: false
    });

  // 3. Now make the request with Playwright; it will be intercepted by Nock.
  const response = await context.get('https://jsonplaceholder.typicode.com/todos/1');
  expect(response.status()).toBe(200);

  const data = await response.json();
  expect(data.title).toBe('Mocked Title');
});
```

b) Axios Mocking

Axios is a popular HTTP client library, but it does not include built-in mocking. Instead, we typically use libraries like `axios-mock-adapter`

```
import axios from 'axios';
import MockAdapter from 'axios-mock-adapter';

const mock = new MockAdapter(axios);

// Mock a GET request to /users
mock.onGet('/users').reply(200, {
  users: [{ id: 1, name: 'Bob' }],
});

// Then your code that calls axios will receive the mock response
const response = await axios.get('/users');
console.log(response.data); // { users: [ { id: 1, name: 'Bob' } ] }
```

c) Supertest Mocking

Supertest doesn't have a built-in feature to mock external HTTP calls. But we can use server instances like Express

```
const express = require('express');
const app = express();

app.get('/hello', (req, res) => {
  res.send({ msg: 'Hello World!' });
});

module.exports = app;

// app.test.js (using Jest)
const request = require('supertest');
const app = require('./app');

test('GET /hello returns a greeting', async () => {
  const response = await request(app).get('/hello');
  expect(response.status).toBe(200);
  expect(response.body).toEqual({ msg: 'Hello World!' });
});
```

6. Session Management

a) Playwright Request

Using this Playwright Request, sharing browser session with api is seamless

```
test('Fetch user details after UI login', async ({ page }) => {
  await page.goto('https://example.com/login');
  await page.fill('#username', 'user');
  await page.fill('#password', 'password');
  await page.click('#submit');

  // Reuse session to fetch user details via API
  const apiResponse = await page.context().request.get('/user-details');
  expect(apiResponse.status()).toBe(200);
  console.log(await apiResponse.json());
});
```

b) Axios

Axios doesn't automatically use the browser session. We need to **extract cookies** from Playwright and set them in Axios manually.

```
test('Fetch user details after UI login using Axios', async ({ page, context }) => {
  // Step 1: Login via UI
  await page.goto('https://example.com/login');
  await page.fill('#username', 'user');
  await page.fill('#password', 'password');
  await page.click('#submit');

  // Step 2: Extract cookies from Playwright
  const cookies = await context.cookies();
  const cookieString = cookies.map(cookie => `${cookie.name}=${cookie.value}`).join('; ');

  // Step 3: Make API request using Axios with browser cookies
  const response = await axios.get('https://example.com/user-details', {
    headers: {
      'Cookie': cookieString
    }
  });

  // Step 4: Validate response
  expect(response.status).toBe(200);
  console.log(response.data);
});
```

c) **Supertest**

Supertest also doesn't automatically use Playwright's session, so we **need to extract cookies manually**.

```
test('Fetch user details after UI login using Supertest', async ({ page, context }) => {
  // Step 1: Login via UI
  await page.goto('https://example.com/login');
  await page.fill('#username', 'user');
  await page.fill('#password', 'password');
  await page.click('#submit');

  // Step 2: Extract cookies from Playwright
  const cookies = await context.cookies();
  const cookieString = cookies.map(cookie => `${cookie.name}=${cookie.value}`).join('; ');

  // Step 3: Make API request using Supertest with browser cookies
  const response = await request('https://example.com')
    .get('/user-details')
    .set('Cookie', cookieString);

  // Step 4: Validate response
  expect(response.status).toBe(200);
  console.log(response.body);
});
```

7. Final Comparison Table

Aspect	Playwright Native API	Supertest	Axios
Dependancy	Built in	External Library	External Library
TypeScript Support	Fully supported	Needs `@types/supertest` For some types	Fully supported
Assertions	Supported with Playwright assertions	Built for testing with own assertions mechanism with fluent api but also supports Playwright assertions	Built only to behave as a httpclient and not for test purposes. But it will support Playwright assertions
Testing Capabilities	Can be used to test Rest, GraphQL, Soap etc	Can be used to test Rest, GraphQL, Soap etc	Can be used to test Rest, GraphQL, Soap etc
Report Integration	Default support to Playwright runner with network trace support	Not integrated with Playwright runner for network trace	Not integrated with Playwright runner for network trace
Parallel testing support	supports parallel testing with Playwright runner	supports parallel testing with Playwright runner	supports parallel testing with Playwright runner
Dependency Injection in tests	Supported by default fixtures	Supported by custom fixtures	Supported by custom fixtures
Mocking Support	No Built in but can use mocking libraries like Nock	No Built in but seamless integration with mock Express server	No Built but can use mocking libraries like `axios-mock-adapter`
Reporting	Built-in Allure support	Needs manual integration	Needs manual integration

Browser to api Session transfer	Straightforward and Seamless	Extract cookies from browser session and pass to library	Extract cookies from browser session and pass to library