📌 Setting Up Playwright for API Automation

Install Playwright

Run the following command in your project directory:

```
npm init playwright@latest
```

This sets up the Playwright with necessary dependencies.

🌐 Making API Requests in Playwright

Basic GET Request

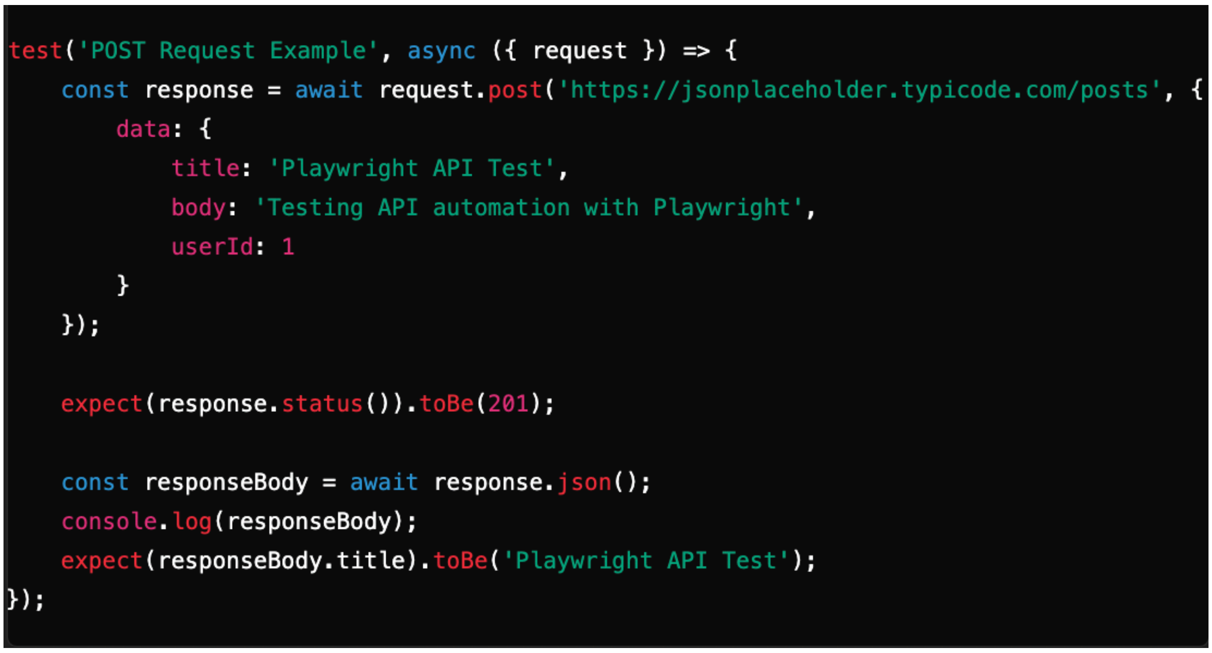
```
import { test, expect } from '@playwright/test';

test('GET Request Example', async ({ request }) => {
    const response = await request.get('https://jsonplaceholder.typicode.com/posts/1');

    // Validate response status
    expect(response.status()).toBe(200);

    // Validate JSON response body
    const responseBody = await response.json();
    console.log(responseBody);
    expect(responseBody.id).toBe(1);
});
```

POST Request with JSON Body



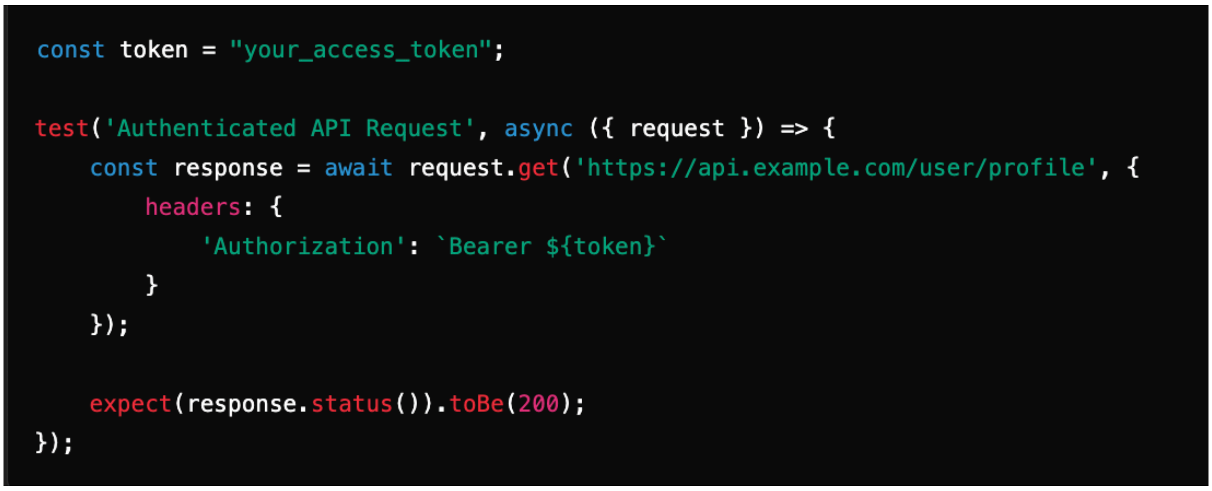
```javascript
test('POST Request Example', async ({ request }) => {
    const response = await request.post('https://jsonplaceholder.typicode.com/posts', {
        data: {
            title: 'Playwright API Test',
            body: 'Testing API automation with Playwright',
            userId: 1
        }
    });

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

    const responseBody = await response.json();
    console.log(responseBody);
    expect(responseBody.title).toBe('Playwright API Test');
});
```

🔐 Handling Authentication in API Tests

Using Bearer Token Authentication



```javascript
const token = "your_access_token";

test('Authenticated API Request', async ({ request }) => {
    const response = await request.get('https://api.example.com/user/profile', {
        headers: {
            'Authorization': `Bearer ${token}`
        }
    });

    expect(response.status()).toBe(200);
});
```

Storing Authentication State for Reuse

```
test('Login and Store Token', async ({ request }) => {
    const response = await request.post('https://api.example.com/login', {
        data: { username: 'testuser', password: 'password123' }
    });

    expect(response.status()).toBe(200);
    const responseBody = await response.json();

    // Save token for further API calls
    const authToken = responseBody.token;
});
```

🔄 PUT & DELETE Requests in Playwright

PUT Request (Updating Data)

```
test('PUT Request Example', async ({ request }) => {
    const response = await request.put('https://jsonplaceholder.typicode.com/posts/1',
        data: {
            title: 'Updated Title',
            body: 'Updated Content'
        }
    });

    expect(response.status()).toBe(200);
});
```

DELETE Request (Deleting Data)

```
test('DELETE Request Example', async ({ request }) => {
    const response = await request.delete('https://jsonplaceholder.typicode.com/posts/1

    expect(response.status()).toBe(200);
});
```

🛠️ Validating API Response with Playwright Assertions

```
test('Validate Response JSON Schema', async ({ request }) => {
    const response = await request.get('https://jsonplaceholder.typicode.com/posts/1');

    expect(response.status()).toBe(200);

    const responseBody = await response.json();

    expect(responseBody).toHaveProperty('id');
    expect(responseBody).toHaveProperty('title');
    expect(responseBody).toHaveProperty('body');
});
```

📈 Running API Tests in Playwright

Run all Playwright tests using:

```
npx playwright test
```

Run only API tests:

```
npx playwright test tests/api
```

Run tests in headed mode (useful for debugging):

```
npx playwright test --headed
```

Run a single test file:

```
npx playwright test tests/api/auth.spec.ts
```

🛠️ Playwright API Testing with CI/CD Integration

Integrate Playwright API tests with **GitHub Actions, Jenkins, or Azure DevOps** by adding a test script to `package.json`:

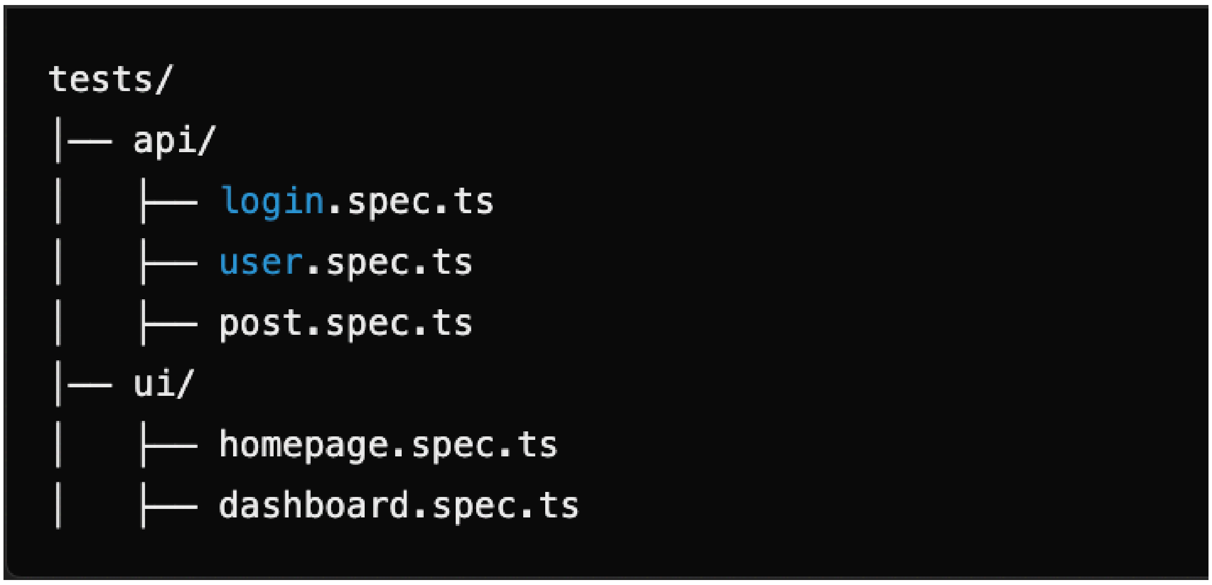
```
"scripts": {
  "test:api": "npx playwright test tests/api"
}
```

Run in CI/CD:

```
npm run test:api
```

📌 Structuring API Tests in Playwright

Folder Structure:



```
tests/
|— api/
|   ├── login.spec.ts
|   ├── user.spec.ts
|   ├── post.spec.ts
|— ui/
|   ├── homepage.spec.ts
|   ├── dashboard.spec.ts
```

Organising tests in separate folders helps manage UI and API automation efficiently.

🚀 Playwright API Test Best Practices

✅ **Use Environment Variables:** Store API keys, tokens, and URLs securely.
✅ **Avoid Hardcoded Data:** Use dynamic test data.
✅ **Validate Response Time:** Set limits for API response time.
✅ **Use Request Hooks:** Mock API responses for testing.
✅ **Run Tests in Parallel:** Use Playwright's parallel execution for faster test runs.

🎯 Conclusion

Playwright is not just for UI testing—it's a powerful tool for **API automation** as well! It provides built-in API support, making validating **REST APIs, authentication mechanisms, and performance testing easier.**

Start **implementing these tests in your projects** today!