Modularization and reusable functions

Navigate to https://playwright.dev

Click the Docs link

Verify the URL

Use variables and functions to organize the code

Project Structure

```
playwright-project/
 ├── tests/
 │    └── playwrightdev.spec.ts
 ├── utils/
 │    └── helpers.ts
 ├── playwright.config.ts
```

utils/helpers.ts — Variables & Functions

```
export const baseUrl = 'https://playwright.dev';

export function getDocsUrl(): string {
  return `${baseUrl}/docs/intro`;
}
```

tests/playwrightdev.spec.ts

```
import { test, expect } from '@playwright/test';
import { baseUrl, getDocsUrl } from '../utils/helpers';

test('Playwright.dev Docs navigation', async ({ page }) => {
  // Use variable
  await page.goto(baseUrl);

  // Click on Docs link using role
  await page.getByRole('link', { name: 'Docs' }).click();

  // Use function to get expected URL
  await expect(page).toHaveURL(getDocsUrl());
});

npx playwright test tests/playwrightdev.spec.ts
```

Get Request

==========

```
import {request,test,expect} from '@playwright/test'
test('API get Method',async () => {

   const apicontext =await request.newContext();
   //api uri
```

```
  const response =await apicontext.get('https://restful-booker.herokuapp.com/booking');

  console.log(await response.json());

  //console.log('response status',await response.status());
   expect(await response.status()).toBe(200);




})
```

==========

Post request:

```
import {request,test,expect} from '@playwright/test'
test('API get Method',async () => {

  const apicontext =await request.newContext();
        const payload = {
   firstname: "Ambika",
   lastname: "KK",
   totalprice: 1000,
   depositpaid: false,
   bookingdates: {
```

```
      checkin: "2025-06-01",
          checkout: "2025-06-16"
          },
          food: "Breakfast",
    additionalneeds: "Non-Smoking Room"
  };

   //api uri

   const response =await apicontext.post('https://restful-booker.herokuapp.com/booking',
          {data:payload,headers:{ 'Content-Type': 'application/json'}});
   console.log(await response.json())
});
```

==========

Shared authentication between API calls-- shared-auth.spec.ts

==========

```
import { test, request, expect } from '@playwright/test';
import { bookingPayload } from '../utility/payload';

test('Shared authentication between API calls', async () => {
  // Step 1: Generate token using /auth endpoint
  const apiContext = await request.newContext();
  const authResponse = await
apiContext.post('https://restful-booker.herokuapp.com/auth', {
    data: {
          username: 'admin',
```

```
        password: 'password123'
        },
        headers: { 'Content-Type': 'application/json' }
  });

  expect(authResponse.ok()).toBeTruthy();
  const authData = await authResponse.json();
  const token = authData.token;
  console.log('Auth Token:', token);

  // Step 2: Use token for booking creation
  const bookingResponse = await
apiContext.post('https://restful-booker.herokuapp.com/booking', {
        data: bookingPayload,
    headers: {
        'Content-Type': 'application/json',
        'Cookie': `token=${token}`
        }
  });

  expect(bookingResponse.ok()).toBeTruthy();
  const bookingData = await bookingResponse.json();
  console.log('Booking Created:', bookingData);

  // Step 3: Validate response fields
  expect(bookingData.booking.firstname).toBe(bookingPayload.firstname);
  expect(bookingData.booking.lastname).toBe(bookingPayload.lastname);
```

```
  expect(bookingData.booking.totalprice).toBe(bookingPayload.totalprice);
});
=============================================================================
=======
```

Datadriven+ API Testing

```
playwright-api-project/
│
├── tests/
│   └── api-tests.spec.ts          # Single spec file combining all scenarios
│
├── data/
│   └── bookings.json              # JSON file for multiple booking payloads
│
├── playwright.config.ts           # Playwright configuration
├── package.json                   # Dependencies and scripts
└── README.md                              # Documentation
```

1.data/bookings.json

```
[
  {
        "firstname": "Ambika",
        "lastname": "KK",
        "totalprice": 1000,
        "depositpaid": true,
        "bookingdates": {
```

```
        "checkin": "2025-06-01",
        "checkout": "2025-06-16"
        },
        "additionalneeds": "Breakfast"
  },
  {
        "firstname": "John",
        "lastname": "Doe",
        "totalprice": 1500,
        "depositpaid": false,
        "bookingdates": {
        "checkin": "2025-07-01",
        "checkout": "2025-07-10"
        },
        "additionalneeds": "Lunch"
  }
]
```

2.tests/api-tests.spec.ts

```
import { test, request, expect } from '@playwright/test';
import readJsonFile  from '../test-data/bookings.json';

test.describe('Combined API Tests', () => {

        const bookings = readJsonFile;
  //  Token-based Authentication (Data-driven)
  for (const bookingPayload of bookings) {
```

```
  test(`Create booking for ${bookingPayload.firstname} with Token Auth`, async () => {
     console.log(`my payload :::${bookingPayload.bookingdates.checkin}`);
       const apiContext = await request.newContext();

       // Step 1: Generate token
       const authResponse = await
apiContext.post('https://restful-booker.herokuapp.com/auth', {
       data: { username: 'admin', password: 'password123' },
       headers: { 'Content-Type': 'application/json' }
       });
    expect(authResponse.ok()).toBeTruthy();
       const token = (await authResponse.json()).token;

       // Step 2: Create booking using token
       const bookingResponse = await
apiContext.post('https://restful-booker.herokuapp.com/booking', {
       data: bookingPayload,
      headers: {
       'Content-Type': 'application/json',
       'Cookie': `token=${token}`
       }
       });

    expect(bookingResponse.ok()).toBeTruthy();
       const bookingData = await bookingResponse.json();
    console.log(`Booking Created for ${bookingPayload.firstname}:`, bookingData);

       // Validate response
```

```
    expect(bookingData.booking.firstname).toBe(bookingPayload.firstname);
    expect(bookingData.booking.lastname).toBe(bookingPayload.lastname);
        });
  }


});
```

Benefits

=======

Cleaner tests: No repeated logic for token generation or booking creation.

Easy maintenance: Update endpoints or headers in one place.

Scalable: Can add more helpers (e.g., updateBooking, deleteBooking)

Global Setup and Teardown:

Global Setup: Prepare environment before any tests run (e.g., login, create session, load config).

Global Teardown: Clean up after all tests finish (e.g., close browser, clear session, log results).

Feature	---export (Named)---	export default
Export type	---Multiple exports allowed---	Only one default export
Import syntax---	import { name }---	import name
Use case---	Utilities, constants---	Main function/class/module
Flexibility--	High---	Simpler for single export

```
├── tests/
│   └── login.spec.ts
├── pages/
│   └── LoginPage.ts
├── utils/
│   └── env.ts
├── global-setup/
│   └── setup.ts
│   └── teardown.ts
├── test-data/
│   └── loginData.xlsx
├── playwright.config.ts
├── .env
├── package.json
```

Step 1: Create Global Setup File

global-setup/setup.ts

```
import { chromium } from '@playwright/test';
import * as dotenv from 'dotenv';

dotenv.config();

export default async () => {
  const browser = await chromium.launch();
```

```
  const page = await browser.newPage();

  await page.goto(`${process.env.BASE_URL}/web/index.php/auth/login`);
  await page.fill('#username', process.env.USERNAME!);
  await page.fill('#password', process.env.PASSWORD!);
  await page.click('button[type="submit"]');

  // Save storage state for reuse
  await page.context().storageState({ path: 'storageState.json' });

  await browser.close();
};
```

Step 2: Create Global Teardown File (Optional)

global-setup/teardown.ts

```
export default async () => {
  console.log('Global teardown completed.');
  // You can add cleanup logic here if needed
};
```

Step 3: Update playwright.config.ts

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

```
export default defineConfig({
  use: {
    baseURL: process.env.BASE_URL,
    storageState: 'storageState.json',
  },
  globalSetup: require.resolve('./global-setup/setup'),
  globalTeardown: require.resolve('./global-setup/teardown'),
});
```

Benefits of Global Setup/Teardown

Feature	Benefit
Session reuse	Avoids logging in before every test
Faster tests	Reduces redundant setup steps
Clean structure	Centralized setup and cleanup logic

==========

Why Use Fixtures?

Benefit	Description
Reusability	Share setup logic across tests
Modularity	Keep tests clean and focused
Scalability	Easily add more shared resources
Isolation	Each test gets its own instance of the fixture

Common Use Cases

==========

Reusing login logic

Creating page object instances

Injecting test data

Managing browser contexts

```
import { test as base, expect,Page ,chromium} from '@playwright/test';

type Fixtures = {
  loggedInPage: Page;
};

const test = base.extend<Fixtures>({
  loggedInPage: async ( {},use) => {
        const browser= await chromium.launch({headless:false})
        const context = await browser.newContext();
        const page = await context.newPage();

        await page.goto('https://the-internet.herokuapp.com/login');
        await page.fill('#username', 'tomsmith');
        await page.fill('#password','SuperSecretPassword!');
        await page.click('button[type="submit"]');
        await use(page);
  },
});

test('dashboard loads after login', async ({ loggedInPage }) => {
```

```
  await loggedInPage.goto('https://the-internet.herokuapp.com/secure');
  expect(await loggedInPage.isVisible('text= Secure Area')).toBeTruthy();
});
```

==========

Global setup and teardown

Fixtures and parallel execution

Configuration via playwright.config.ts

Environment variables

Project Structure

```
├── tests/
│   └── login.spec.ts
├── pages/
│   └── LoginPage.ts
├── utils/
│   └── env.ts
├── global-setup.ts
├── global-teardown.ts
├── playwright.config.ts
├── .env
```

Note:

dotenv is a Node.js package that reads a .env file.

dotenv.config() loads the variables defined in that file into process.env.

.env – Environment Variables

==========

BASE_URL=https://www.saucedemo.com

USERNAME=standard_user

PASSWORD=secret_sauce

utils/env.ts – Load Environment Variables

to Use it in Playwright TypeScript

==========

Environment variable:

```
import * as dotenv from 'dotenv';
dotenv.config();

const baseUrl = process.env.BASE_URL;
const username = process.env.USERNAME;
const password = process.env.PASSWORD;

test('Login using environment variables', async ({ page }) => {
  await page.goto(baseUrl!);
```

```
  await page.getByPlaceholder('Username').fill(username!);
  await page.getByPlaceholder('Password').fill(password!);
  await page.getByRole('button', { name: 'Login' }).click();
});
```

==========