Authentication

Introduction

Playwright executes tests in isolated environments called browser contexts. This isolation model improves reproducibility and prevents cascading test failures. Tests can load existing authenticated state. This eliminates the need to authenticate in every test and speeds up test execution.

Core concepts

Regardless of the authentication strategy you choose, you are likely to store authenticated browser state on the file system.

We recommend to create `playwright/.auth` directory and add it to your `.gitignore`. Your authentication routine will produce authenticated browser state and save it to a file in this `playwright/.auth` directory. Later on, tests will reuse this state and start already authenticated.

- Bash
- PowerShell
- Batch

```
mkdir -p playwright/.auth
echo $'\nplaywright/.auth' >> .gitignore
New-Item -ItemType Directory -Force -Path playwright\.auth
Add-Content -path .gitignore "`r`nplaywright/.auth"
md playwright\.auth
echo. >> .gitignore
echo "playwright/.auth" >> .gitignore
```

Basic: shared account in all tests

This is the **recommended** approach for tests **without server-side state**. Authenticate once in the **setup project**, save the authentication state, and then reuse it to bootstrap each test already authenticated.

When to use

- When you can imagine all your tests running at the same time with the same account, without affecting each other.

When not to use

- Your tests modify server-side state. For example, one test checks the rendering of the settings page, while the other test is changing the setting, and you run tests in parallel. In this case, tests must use different accounts.
- Your authentication is browser-specific.

Details

Create `tests/auth.setup.ts` that will prepare authenticated browser state for all other tests.

tests/auth.setup.ts

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

const authFile = path.join(__dirname, '../playwright/.auth/user.json');

setup('authenticate', async ({ page }) => {
  // Perform authentication steps. Replace these actions with your own.
  await page.goto('https://github.com/login');
  await page.getByLabel('Username or email address').fill('username');
  await page.getByLabel('Password').fill('password');
  await page.getByRole('button', { name: 'Sign in' }).click();
  // Wait until the page receives the cookies.
  //
  // Sometimes login flow sets cookies in the process of several redirects.
  // Wait for the final URL to ensure that the cookies are actually set.
  await page.waitForURL('https://github.com/');
  // Alternatively, you can wait until the page reaches a state where all
cookies are set.
  await expect(page.getByRole('button', { name: 'View profile and more'
})).toBeVisible();

  // End of authentication steps.

  await page.context().storageState({ path: authFile });
});
```

Create a new `setup` project in the config and declare it as a dependency for all your testing projects. This project will always run and authenticate before all the tests. All testing projects should use the authenticated state as `storageState`.

playwright.config.ts

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

export default defineConfig({
  projects: [
    // Setup project
    { name: 'setup', testMatch: /.*\.setup\.ts/ },

    {
      name: 'chromium',
      use: {
```

```
        ...devices['Desktop Chrome'],
        // Use prepared auth state.
        storageState: 'playwright/.auth/user.json',
      },
      dependencies: ['setup'],
    },

    {
      name: 'firefox',
      use: {
        ...devices['Desktop Firefox'],
        // Use prepared auth state.
        storageState: 'playwright/.auth/user.json',
      },
      dependencies: ['setup'],
    },
  ],
});
```

Tests start already authenticated because we specified `storageState` in the config.

tests/example.spec.ts

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

test('test', async ({ page }) => {
  // page is authenticated
});
```

Note that you need to delete the stored state when it expires. If you don't need to keep the state between test runs, write the browser state under testProject.outputDir, which is automatically cleaned up before every test run.

Authenticating in UI mode

UI mode will not run the `setup` project by default to improve testing speed. We recommend to authenticate by manually running the `auth.setup.ts` from time to time, whenever existing authentication expires.

First enable the `setup` project in the filters, then click the triangle button next to `auth.setup.ts` file, and then disable the `setup` project in the filters again.

Moderate: one account per parallel worker

This is the **recommended** approach for tests that **modify server-side state**. In Playwright, worker processes run in parallel. In this approach, each parallel worker is authenticated once. All tests ran by worker are reusing the same authentication state. We will need multiple testing accounts, one per each parallel worker.

When to use

- Your tests modify shared server-side state. For example, one test checks the rendering of the settings page, while the other test is changing the setting.

When not to use

- Your tests do not modify any shared server-side state. In this case, all tests can use a single shared account.

Details

We will authenticate once per worker process, each with a unique account.

Create `playwright/fixtures.ts` file that will override `storageState` fixture to authenticate once per worker. Use testInfo.parallelIndex to differentiate between workers.

playwright/fixtures.ts

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

export * from '@playwright/test';
export const test = baseTest.extend<{}, { workerStorageState: string }>({
  // Use the same storage state for all tests in this worker.
  storageState: ({ workerStorageState }, use) => use(workerStorageState),

  // Authenticate once per worker with a worker-scoped fixture.
  workerStorageState: [async ({ browser }, use) => {
    // Use parallelIndex as a unique identifier for each worker.
    const id = test.info().parallelIndex;
    const fileName = path.resolve(test.info().project.outputDir,
`.auth/${id}.json`);

    if (fs.existsSync(fileName)) {
      // Reuse existing authentication state if any.
      await use(fileName);
      return;
    }

    // Important: make sure we authenticate in a clean environment by
unsetting storage state.
    const page = await browser.newPage({ storageState: undefined });

    // Acquire a unique account, for example create a new one.
    // Alternatively, you can have a list of precreated accounts for
testing.
    // Make sure that accounts are unique, so that multiple team members
    // can run tests at the same time without interference.
    const account = await acquireAccount(id);

    // Perform authentication steps. Replace these actions with your own.
    await page.goto('https://github.com/login');
    await page.getByLabel('Username or email
address').fill(account.username);
    await page.getByLabel('Password').fill(account.password);
    await page.getByRole('button', { name: 'Sign in' }).click();
    // Wait until the page receives the cookies.
    //
    // Sometimes login flow sets cookies in the process of several
redirects.
    // Wait for the final URL to ensure that the cookies are actually set.
    await page.waitForURL('https://github.com/');
```

```
    // Alternatively, you can wait until the page reaches a state where all
cookies are set.
    await expect(page.getByRole('button', { name: 'View profile and more'
})).toBeVisible();

    // End of authentication steps.

    await page.context().storageState({ path: fileName });
    await page.close();
    await use(fileName);
  }, { scope: 'worker' }],
});
```

Now, each test file should import `test` from our fixtures file instead of `@playwright/test`. No changes are needed in the config.

tests/example.spec.ts

```
// Important: import our fixtures.
import { test, expect } from '../playwright/fixtures';

test('test', async ({ page }) => {
  // page is authenticated
});
```

Advanced scenarios

Authenticate with API request

When to use

- Your web application supports authenticating via API that is easier/faster than interacting with the app UI.

Details

We will send the API request with APIRequestContext and then save authenticated state as usual.

In the setup project:

tests/auth.setup.ts

```
import { test as setup } from '@playwright/test';

const authFile = 'playwright/.auth/user.json';

setup('authenticate', async ({ request }) => {
  // Send authentication request. Replace with your own.
  await request.post('https://github.com/login', {
    form: {
      'user': 'user',
      'password': 'password'
    }
  });
  await request.storageState({ path: authFile });
});
```

Alternatively, in a worker fixture:

playwright/fixtures.ts

```
import { test as baseTest, request } from '@playwright/test';
import fs from 'fs';
import path from 'path';

export * from '@playwright/test';
export const test = baseTest.extend<{}, { workerStorageState: string }>({
  // Use the same storage state for all tests in this worker.
  storageState: ({ workerStorageState }, use) => use(workerStorageState),

  // Authenticate once per worker with a worker-scoped fixture.
  workerStorageState: [async ({}, use) => {
    // Use parallelIndex as a unique identifier for each worker.
    const id = test.info().parallelIndex;
    const fileName = path.resolve(test.info().project.outputDir,
`.auth/${id}.json`);

    if (fs.existsSync(fileName)) {
      // Reuse existing authentication state if any.
      await use(fileName);
      return;
    }

    // Important: make sure we authenticate in a clean environment by
unsetting storage state.
    const context = await request.newContext({ storageState: undefined });

    // Acquire a unique account, for example create a new one.
    // Alternatively, you can have a list of precreated accounts for
testing.
    // Make sure that accounts are unique, so that multiple team members
    // can run tests at the same time without interference.
    const account = await acquireAccount(id);

    // Send authentication request. Replace with your own.
    await context.post('https://github.com/login', {
      form: {
        'user': 'user',
        'password': 'password'
      }
    });

    await context.storageState({ path: fileName });
    await context.dispose();
    await use(fileName);
  }, { scope: 'worker' }],
});
```

Multiple signed in roles

When to use

- You have more than one role in your end to end tests, but you can reuse accounts across all tests.

Details

We will authenticate multiple times in the setup project.

tests/auth.setup.ts

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

const adminFile = 'playwright/.auth/admin.json';

setup('authenticate as admin', async ({ page }) => {
  // Perform authentication steps. Replace these actions with your own.
  await page.goto('https://github.com/login');
  await page.getByLabel('Username or email address').fill('admin');
  await page.getByLabel('Password').fill('password');
  await page.getByRole('button', { name: 'Sign in' }).click();
  // Wait until the page receives the cookies.
  //
  // Sometimes login flow sets cookies in the process of several redirects.
  // Wait for the final URL to ensure that the cookies are actually set.
  await page.waitForURL('https://github.com/');
  // Alternatively, you can wait until the page reaches a state where all
cookies are set.
  await expect(page.getByRole('button', { name: 'View profile and more'
})).toBeVisible();

  // End of authentication steps.

  await page.context().storageState({ path: adminFile });
});

const userFile = 'playwright/.auth/user.json';

setup('authenticate as user', async ({ page }) => {
  // Perform authentication steps. Replace these actions with your own.
  await page.goto('https://github.com/login');
  await page.getByLabel('Username or email address').fill('user');
  await page.getByLabel('Password').fill('password');
  await page.getByRole('button', { name: 'Sign in' }).click();
  // Wait until the page receives the cookies.
  //
  // Sometimes login flow sets cookies in the process of several redirects.
  // Wait for the final URL to ensure that the cookies are actually set.
  await page.waitForURL('https://github.com/');
  // Alternatively, you can wait until the page reaches a state where all
cookies are set.
  await expect(page.getByRole('button', { name: 'View profile and more'
})).toBeVisible();

  // End of authentication steps.

  await page.context().storageState({ path: userFile });
});
```

After that, specify `storageState` for each test file or test group, **instead of** setting it in the config.

tests/example.spec.ts

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

test.use({ storageState: 'playwright/.auth/admin.json' });
```

```
test('admin test', async ({ page }) => {
  // page is authenticated as admin
});

test.describe(() => {
  test.use({ storageState: 'playwright/.auth/user.json' });

  test('user test', async ({ page }) => {
    // page is authenticated as a user
  });
});
```

See also about authenticating in the UI mode.

Testing multiple roles together

When to use

- You need to test how multiple authenticated roles interact together, in a single test.

Details

Use multiple BrowserContexts and Pages with different storage states in the same test.

tests/example.spec.ts

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

test('admin and user', async ({ browser }) => {
  // adminContext and all pages inside, including adminPage, are signed in
as "admin".
  const adminContext = await browser.newContext({ storageState:
'playwright/.auth/admin.json' });
  const adminPage = await adminContext.newPage();

  // userContext and all pages inside, including userPage, are signed in as
"user".
  const userContext = await browser.newContext({ storageState:
'playwright/.auth/user.json' });
  const userPage = await userContext.newPage();

  // ... interact with both adminPage and userPage ...

  await adminContext.close();
  await userContext.close();
});
```

Testing multiple roles with POM fixtures

When to use

- You need to test how multiple authenticated roles interact together, in a single test.

Details

You can introduce fixtures that will provide a page authenticated as each role.

Below is an example that creates fixtures for two Page Object Models - admin POM and user POM. It assumes `adminStorageState.json` and `userStorageState.json` files were created in the global setup.

playwright/fixtures.ts

```
import { test as base, type Page, type Locator } from '@playwright/test';

// Page Object Model for the "admin" page.
// Here you can add locators and helper methods specific to the admin page.
class AdminPage {
  // Page signed in as "admin".
  page: Page;

  // Example locator pointing to "Welcome, Admin" greeting.
  greeting: Locator;

  constructor(page: Page) {
    this.page = page;
    this.greeting = page.locator('#greeting');
  }
}

// Page Object Model for the "user" page.
// Here you can add locators and helper methods specific to the user page.
class UserPage {
  // Page signed in as "user".
  page: Page;

  // Example locator pointing to "Welcome, User" greeting.
  greeting: Locator;

  constructor(page: Page) {
    this.page = page;
    this.greeting = page.locator('#greeting');
  }
}

// Declare the types of your fixtures.
type MyFixtures = {
  adminPage: AdminPage;
  userPage: UserPage;
};

export * from '@playwright/test';
export const test = base.extend<MyFixtures>({
  adminPage: async ({ browser }, use) => {
    const context = await browser.newContext({ storageState:
'playwright/.auth/admin.json' });
    const adminPage = new AdminPage(await context.newPage());
    await use(adminPage);
    await context.close();
  },
  userPage: async ({ browser }, use) => {
    const context = await browser.newContext({ storageState:
'playwright/.auth/user.json' });
    const userPage = new UserPage(await context.newPage());
    await use(userPage);
    await context.close();
  },
```

```
});
```

tests/example.spec.ts

```
// Import test with our new fixtures.
import { test, expect } from '../playwright/fixtures';

// Use adminPage and userPage fixtures in the test.
test('admin and user', async ({ adminPage, userPage }) => {
  // ... interact with both adminPage and userPage ...
  await expect(adminPage.greeting).toHaveText('Welcome, Admin');
  await expect(userPage.greeting).toHaveText('Welcome, User');
});
```

Session storage

Reusing authenticated state covers cookies and local storage based authentication. Rarely, session storage is used for storing information associated with the signed-in state. Session storage is specific to a particular domain and is not persisted across page loads. Playwright does not provide API to persist session storage, but the following snippet can be used to save/load session storage.

```
// Get session storage and store as env variable
const sessionStorage = await page.evaluate(() =>
JSON.stringify(sessionStorage));
fs.writeFileSync('playwright/.auth/session.json', sessionStorage, 'utf-8');

// Set session storage in a new context
const sessionStorage =
JSON.parse(fs.readFileSync('playwright/.auth/session.json', 'utf-8'));
await context.addInitScript(storage => {
  if (window.location.hostname === 'example.com') {
    for (const [key, value] of Object.entries(storage))
      window.sessionStorage.setItem(key, value);
  }
}, sessionStorage);
```

Avoid authentication in some tests

You can reset storage state in a test file to avoid authentication that was set up for the whole project.

not-signed-in.spec.ts

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

// Reset storage state for this file to avoid being authenticated
test.use({ storageState: { cookies: [], origins: [] } });

test('not signed in test', async ({ page }) => {
  // ...
});
```

Previous
API testing

Next
Auto-waiting

Learn

Community

More