가이드 문서

안녕하세요 OF1-2팀 박지섭입니다.
UI 테스트 작성 방법 및 사용법에 관한 가이드를 전달드립니다.

UI테스트란

- 통합테스트보다 큰 단위: UI 테스트는 개별 모듈 간의 상호작용을 검증하는 통합 테스트보다 더 큰 범위에서 전체 애플리케이션의 흐름을 테스트
- 유닛 테스트, 통합 테스트 보다 `대형 테스트 기법`으로 분류: 애플리케이션의 전체적인 동작을 검증하는 데 중점을 둔다.
- 사용자 흐름 테스트: 실제 애플리케이션을 사용자가 사용하는 것과 같은 방식으로 일련의 동작을 수행해 보는 테스트
- 사용자 시나리오 검증: UI 테스트는 사용자가 애플리케이션을 사용할 때 겪을 수 있는 다양한 시나리오를 검증합니다. 버튼 클릭, 폼 제출, 화면 전환, 텍스트 입력 등의 인터페이스 요소들이 기대한 대로 작동하는지 확인할수 있다.

UI 테스트의 중요성

1. **사용자 경험 향상**: UI 테스트는 애플리케이션의 사용자 인터페이스가 직관적이고 오류 없이 작동하는지 확인합니다. 이는 최종 사용자에게 긍정적인 경험을 제공합니다.
2. **기능 검증**: UI 테스트는 애플리케이션의 다양한 기능들이 예상한 대로 작동하는지 확인합니다. 이는 코드 변경 시 발생할 수 있는 회귀(regression) 버그를 발견하는 데 유용합니다.
3. **품질 보증**: UI 테스트는 애플리케이션의 전반적인 품질을 보장합니다. 이는 사용자 불만을 줄이고, 신뢰성을 높입니다.
4. **시간 절약**: 자동화된 UI 테스트는 수동 테스트보다 빠르고 효율적입니다. 이는 개발 주기를 단축하고, 빠른 배포를 가능하게 합니다.

Puppeteer와 Cucumber 소개

Puppeteer

Puppeteer는 Google에서 개발한 Node.js 라이브러리로, Headless Chrome 또는 Chromium을 제어할 수 있습니다. Puppeteer는 브라우저를 프로그래밍 방식으로 제어할 수 있게 하여, 다음과 같은 다양한 작업을 자동화할 수 있습니다:

- 웹 페이지 탐색 및 상호작용
- 스크린샷 및 PDF 생성
- 웹 페이지 콘텐츠 스크랩

- 퍼포먼스 테스트 및 트러블슈팅

Puppeteer의 주요 기능:

- **Headless 브라우저 제어**: UI 없이 백그라운드에서 브라우저를 실행할 수 있으며, 필요에 따라 UI를 활성화할 수도 있습니다.
- **DOM 조작**: JavaScript를 사용하여 웹 페이지의 DOM 요소를 탐색하고 조작할 수 있습니다.
- **네트워크 요청 조작**: 요청을 가로채고, 수정하고, 응답을 시뮬레이트할 수 있습니다.

Cucumber

Cucumber는 BDD(Behavior Driven Development) 프레임워크로, 비즈니스 이해 관계자와 개발자 간의 소통을 원활하게 하기 위해 자연어 형식의 시나리오를 테스트 코드로 변환합니다. Cucumber는 Gherkin 언어를 사용하여 시나리오를 작성하며, 이를 통해 테스트 케이스를 명확하게 정의할 수 있습니다.

Cucumber의 주요 기능:

- **Gherkin 언어**: 비즈니스 이해 관계자와 개발자가 모두 이해할 수 있는 자연어 형식의 시나리오를 작성할 수 있습니다.
- **스텝 정의**: 각 시나리오의 단계를 코드로 구현하여 실행할 수 있습니다.
- **BDD 지원**: 테스트 주도 개발(TDD)과 비즈니스 주도 개발(BDD)을 통합하여, 요구 사항과 테스트 케이스를 일치시킬 수 있습니다.

폴더 구조

features

- **설명:** 해당 폴더는 UI 테스트를 동작하게 하는 시나리오가 적혀있으며, 시나리오는 Gherkin 언어로 작성되어 있습니다.
    - **Gherkin 언어:** 읽기 쉽고, 자연어 스타일의 문법을 사용하는 언어로, 테스트 시나리오를 기술하기 위해 사용합니다.
- **예시:**

```
@login
Feature: Login

  @login
  Scenario: Successful login
    Given I am on the login page
    When I enter valid credentials
    And I click the login button
```

```
    And make new Document
    Then I should be redirected to the dashboard
```

- **설명:**
  - 각 시나리오마다 어노테이션을 사용해서, 시나리오를 분리해야 합니다.
    - 어노테이션을 사용하지 않고, 시나리오만 작성하게 되면 원하는 시나리오만 실행시키기가 어렵습니다.
  - 위 예시에서 각 라인은 특정 동작을 설명합니다. 해당 라인들은 Given, When, Then 구문을 사용하도록 되어있습니다.
    - **Given:** 초기 상태를 설명하는 라인입니다.
    - **When, And:** 사용자가 취하는 행동을 의미하는 라인입니다.
    - **Then:** 예상되는 결과를 설명합니다.

step-definitions

- **설명:** 해당 폴더에는 `features` 폴더에 있는 시나리오를 구현하는 단계 정의가 들어 있습니다. 각 단계는 실제로 테스트 코드들을 포함하고 있으며, 위에서 Gherkin 언어로 작성된 시나리오와 매핑됩니다.
- **예시:** login 시나리오 구현 부분입니다.

```
import { expect } from "chai";
import { Given, When, Then, Before, After } from "@cucumber/cucumber";
import { LOCALHOST_URL, PASSWORD_INPUT, TYPING_DELAY, URL, USERID_INPUT }
from "constants/common";
import puppeteer, { Browser, Page } from 'puppeteer';

let browser: Browser;
let page: Page;

Before({ tags: '@login' }, async function () {
  browser = await puppeteer.launch({ headless: true });
  page = await browser.newPage();
});

After({ tags: '@login' }, async function () {
  await browser.close();
});

Given('I am on the login page', async function () {
  await page.goto(LOCALHOST_URL);
});
```

```
When('I enter valid credentials', async function () {
  await page.waitForNavigation();
  const userIdInput = await page.waitForSelector(USERID_INPUT);
  if (!userIdInput) return;
  await userIdInput.type('seunghyun_baek', { delay: TYPING_DELAY });
  const passwordInput = await page.waitForSelector(PASSWORD_INPUT);
  if (!passwordInput) return;
  await passwordInput.type('qwer1234!@', { delay: TYPING_DELAY });
});

When('I click the login button', async function () {
  const loginButton = await page.waitForSelector('#root > div > button');
  if (!loginButton) return;
  await loginButton.click();
});

When('make new Document', async function () {
  const clickDocxRadioButton = await page.waitForSelector('#root > div > div
> span:nth-child(1)');
  if (!clickDocxRadioButton) return;
  await clickDocxRadioButton.click();
  const createNewDocumentInput = await page.waitForSelector('#root > div >
div > div > input[type=text]');
  if (!createNewDocumentInput) return;
  await createNewDocumentInput.type('공동편집 UI 테스트', { delay: TYPING_DELAY
});
  const createButton = await page.waitForSelector('#root > div > div > div >
button');
  if (!createButton) return;
  await createButton.click();
});

Then('I should be redirected to the dashboard', async function () {
  const expectedURL = URL;
  await page.waitForSelector('#root', { timeout: 10 * 1000 });
  const currentURL = page.url();
  expect(currentURL).to.equal(expectedURL);
});
```

- **주의점:** 올바른 매칭을 위해 시나리오의 각 스텝을 유니크하게 작성해야 합니다. 특정 문구가 여러 시나리오에 포함되면 독립적으로 작동하지 않습니다.
  - 예시: 2명의 유저가 문서를 여는 작업에 대해 `"User A" and "User B" have opened the same document`와 같이 작성하면 모든 시나리오에 해당 문장이 포함된 경우 모두 성공 케이스가 되어 독립적으로 작동하지 않게 됩니다.

utils

- **설명:** 해당 폴더에는 테스트에 자주 사용되는 유틸성 함수들이 포함되어 있습니다. 예를 들어 페이지 탐색, 글씨 적기, 단어 탐색 등이 있습니다.
- **폴더 구성:**
  - **clearup:**
    - 테스트가 끝난 뒤 문서를 비워주는 역할을 하는 함수 정보들이 담겨 있습니다.
  - **customConsole:**
    - 콘솔을 커스텀하게 사용할 수 있게끔 만들어놓은 유틸성 함수입니다. 이를 import해서 사용하면 아래와 같은 형식의 콘솔 정보를 확인할 수 있습니다.
    - `[2024-05-27 13:11:04 \n Error location: step-definitions/loginsteps.ts:24 \n Error정보 ]`
  - **elementActions:**
    - Element에 관련된 기능들을 수행하는 유틸성 함수입니다. element를 클릭하거나, 단어를 변경하거나, 글씨를 작성하는 등의 동작을 수행합니다.
  - **keyboardActions:**
    - 키보드에 관련된 기능들을 수행하는 유틸성 함수입니다.
    - `page.keyboard` (puppeteer 기능)에 관련된 정보는 이 모듈을 사용하면 됩니다.
  - **mouseActions:**
    - 마우스에 관련된 기능들을 수행하는 유틸성 함수입니다.
    - `page.mouse` (puppeteer 기능)에 관련된 정보는 이 모듈을 사용하면 됩니다.
  - **navigation:**
    - 페이지를 이동할 때 사용하는 함수입니다.
  - **shapeActions:**
    - 도형에 관련된 동작을 수행하는 함수입니다.
  - **tableActions:**
    - 표에 관련된 동작을 수행하는 함수입니다.
  - **textEvaluation:**
    - 텍스트에 대해서 값을 평가하는 기능들이 담겨 있습니다. 예를 들어, `checkSingleMessageSynchronized`는 단일 문장에 대해 실제 값과 기대 값을 비교하고, 문장이 여러 개인 경우 `checkMessageSynchronized`를 사용해 각 문장을 순회하며 값이 일치하는지 확인합니다.

- **util:**
  - 전체적으로 사용할 수 있는 유틸 함수들에 대한 정보입니다.

constants

- **설명:** 현재는 common 파일과 office_docx 파일로 나뉘어져 있습니다.
  - **common:** 공통적으로 사용되는 상수들의 집합으로, LOCALHOST_URL이나 URL같은 공통적으로 사용되는 부분 및 login 버튼이나 첫 번째 문서에 관련된 selector 정보를 담고 있습니다.
  - **office_docx:** docx, pptx, xlsx 중 docx에 해당하는 selector 정보들을 모아놓았습니다.
    - 예시:

```
export const PARAGRAPH = '.pk-office-d-text.pk-office-Normal';
export const DISCONNECT_BUTTON = '#pk-office-a-root > div.src-style-appbar-AppBar-module__container--e1mgO > div.src-style-appbar-AppBar-module__right--GX4AR > button';
```

report

- **설명:**
  - report의 경우에는 모든 시나리오의 테스트가 끝난 뒤 생성되는 리포트 파일을 포함합니다. generate-report.ts 스크립트가 리포트 생성을 담당합니다.
  - report의 경우에는 `generate-report.ts` 측에서 리포트 생성을 담당합니다.

```
import reporter from 'cucumber-html-reporter';

const options = {
  theme: 'bootstrap',
  jsonFile: 'reports/cucumber-report.json',
  output: 'reports/cucumber-report.html',
  reportSuiteAsScenarios: true,
  scenarioTimestamp: true,
  launchReport: true,
  metadata: {
    'App Version': '1.0.0',
    'Test Environment': 'STAGING',
    Browser: 'Chrome 93.0',
    Platform: 'Windows 10'
  }
};
```

```
try {
  reporter.generate(options);
} catch (err) {
  console.error('Failed to generate HTML report:', err);
}
```

- **theme:** 리포트의 테마를 설정합니다. 여기서는 'bootstrap' 테마를 사용합니다.
- **jsonFile:** Cucumber 테스트 결과가 저장된 JSON 파일 경로를 지정합니다.
- **output:** 생성될 HTML 리포트의 파일 경로를 지정합니다.
- **reportSuiteAsScenarios:** 리포트를 시나리오 단위로 보여줍니다.
- **scenarioTimestamp:** 각 시나리오의 타임스탬프를 포함합니다.
- **launchReport:** 리포트 생성 후 자동으로 브라우저에서 열도록 설정합니다.
- **metadata:** 리포트에 포함할 메타데이터를 정의합니다. 예를 들어, 앱 버전, 테스트 환경, 브라우저, 플랫폼 등을 설정할 수 있습니다.

리포트 방식에 대해서 사용 및 설명

UI Test 방법

```
"scripts": {
"build": "tsc",
"test:login": "tsc && cucumber-js --format progress --tags @login --import
dist/step-definitions/**/*.js features/",
"test:others": "cucumber-js --format progress --tags 'not @login' --import
dist/step-definitions/**/*.js features/",
"report": "node generate-report.js",
"test": "build test:login test:others report"
}
```

- 현재 스크립트는 위와 같이 작성되어 있으며, 기본적으로 npm run test를 실행하게 될 시에, 모든 테스트 들이 동작하도록 되어있습니다.(추가 확인 필요)

에러 핸들링 및 커스텀 콘솔 사용 방법

1. **에러 핸들링:** `withErrorHandling` 함수를 사용하여 에러를 처리
2. **커스텀 콘솔:** `import '/utils/customConsole.js'` 수행

주의 사항

1. **puppeteer는 크롬에서 동작하는 기능입니다.** 문서에 편집을 가할 때 해당 탭에 있어야 동작이 가능합니다. 따라서 동시편집 케이스와 다중 유저 동작 케이스가 나뉘게 됩니다.
    - **동시편집 케이스:** 유저 수만큼 browser를 생성하고, 각 유저들에게 해당 browser를 부여하여 동작하도록 합니다.
        - 예시

```
let browsers: { [key: string]: Browser } = {};
let pages: { [key: string]: Page } = {};

Before(async function () {
  const userBrowsers = await Promise.all([
    puppeteer.launch({ headless: ISHEADLESS }),
    puppeteer.launch({ headless: ISHEADLESS })
  ]);

  browsers['UserA'] = userBrowsers[0];
  browsers['UserB'] = userBrowsers[1];

  pages['UserA'] = await browsers['UserA'].newPage();
  pages['UserB'] = await browsers['UserB'].newPage();
});
```

- **다중 유저 동작 케이스:** 하나의 browser를 생성하고, 유저들에게 해당 browser를 부여하여 사용합니다. 이때 각 유저들은 하나의 탭을 가지고 있고, 각 유저가 어떤 동작을 하게 될 때는 탭이 활성화가 되어야 하기 때문에, page.bringToFront()로 탭을 활성화 시켜주셔야 합니다.

```
// pages를 array 형태로 사용하는 방법
let browser: Browser;
let pages: { [key: string]: Page } = {};

Before(async function () {
  browser = await puppeteer.launch({ headless: ISHEADLESS });
  pages['UserA'] = await browser.newPage();
  pages['UserB'] = await browser.newPage();
});

// page들을 분리하여 사용하는 방식
```

```
let browser: Browser;
let pageA Page;
let pageB Page;

Before(async function () {
  browser = await puppeteer.launch({ headless: ISHEADLESS });
  pageA = await browser.newPage();
  pageB = await browser.newPage();
});
```

추가적으로 필요한 정보

Node 버전

- 해당 테스트는 Node 18.10 버전 이상에서 사용할 수 있습니다. 오피스에서는 Node 16 버전을 사용하고 있으므로, 18.10 버전으로 업그레이드한 뒤 사용하도록 해야 합니다.
- **nvm 설치 방법 가이드:** nvm 설치 가이드
- **맥 기준 nvm 설치 방법:**
  1. `$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh | bash` 또는 `brew install nvm` 명령어 입력
  2. 설치 후 nvm을 통해 Node 버전 업그레이드

Cucumber와 Puppeteer 사용 시 유용한 팁

시나리오에서 파라미터 사용하는 방법

- 일반적인 파라미터 정의 및 사용법:
- **Example 사용한 파라미터 정의:** 시나리오 파일에서 `Scenario Outline` 키워드를 사용하여 파라미터를 정의합니다. 파라미터는 `<>` 로 표현합니다.

파라미터 예시

```
Scenario Outline: Successful login with multiple users
  Given I am on the login page
  When I enter username "<username>" and password "<password>"
  And I click the login button
  Then I should be redirected to the dashboard
```

```
  Examples:
    | username | password    |
    | user1    | password123 |
    | user2    | password456 |
    | user3    | password789 |
```

사용부 예시

```
Given('Login with {string} and {string}', async function (username,
password) {
  await page.type('#username', username);
  await page.type('#password', password);
});

When('Navigate to {string} page', async function (url) {
  await page.goto(url);
});

Then('Verify {string}', async function (param) {
  const pageContent = await page.content();
  expect(pageContent).to.include(param);
});
```

Before와 After

- **@cucumber/cucumber**에서 제공하는 Before와 After를 사용하여 각 시나리오 전후에 실행되는 코드를 정의할 수 있습니다.
    1. **BeforeAll:** 모든 시나리오 전에 한 번 실행되는 코드
    2. **Before:** 각 시나리오 전에 실행되는 코드 (예: 테스트 데이터 초기화, 사용자 로그인)
    3. **BeforeStep:** 각 단계 전에 실행되는 코드 (예: 각 단계 전 스크린샷 촬영)
- After도 이와 동일합니다. Before대신 After를 넣으면, 시나리오 후에 실행되는 코드가 됩니다.

이슈 및 해결 방법 공유

1. **공동 편집 케이스에서 문서 다수를 열어놓고 시작할 때 timeout 에러 발생:**
    - Cucumber에서의 timeout 시간을 5초로 설정하고 있으므로, timeoutError가 발생하는 경우
        1. timeout 시간을 늘려서 해결되는지 확인

    2. timeout 문제 해결되지 않을 시, evaluate나, waitForSelector등 조건에 만족하지 않는 케이스가 있는지 확인

2. **글자가 사라지는 경우:**
    - "Hello world,"를 쓰기 바랬는데, 어쩌다가 "Hell world,"와 같이 글자가 유실되는 경우가 발생할 수 있습니다.
    - 현재 해당 이슈는 로직적으로 수정되었지만, 해당 부분에서 문제 발생시, TYPING_DELAY을 조절하여 사용하면 됩니다.

Puppeteer 기능

- puppteer의 기능들은 아래 사이트에서 확인 가능합니다.
    - https://pptr.dev/api
- **page 기능:**
    - 주로 사용 하는 page 기능들입니다.
        - **evaluate:** 컨텍스트에서 함수를 평가하고 결과를 반환합니다. Promise가 해결될 때까지 기다렸다가 해당 값을 반환합니다.

```
const result = await page.evaluate(() => {
  return document.querySelector('h1').innerText;
});
console.log(result); // 페이지에서 h1 태그의 텍스트를 가져옴
```

- page.$
    - CSS 셀렉터를 사용하여 페이지에서 첫 번째 일치하는 요소를 선택합니다. 요소가 없으면 `null`을 반환합니다.

```
const element = await page.$('div.container');
if (element) {
  console.log('Element found');
} else {
  console.log('Element not found');
}
```

- page.bringToFront
    - 해당되는 페이지를 포그라운드로 가져옵니다. 여러 페이지 제어할 때 유용합니다.

```
await page.bringToFront();
console.log('Page is now in the foreground');
```

- waitForSelector
  - 주어진 CSS 셀렉터가 나타날 때까지 기다립니다. 타임아웃이 발생하면 에러를 던집니다.
  - 주의사항:
    - 해당 기능 관련되서 일반적으로는 정상적으로 동작하지만, checkMessageSynchronized 함수의 경우, 순회하면서 모든 조건이 맞는지에 대해서 확인합니다. 이런 경우에는 정상적으로 동작 안하는 경우가 있으므로, `waitForElements()` 함수처럼 작성해주시면 됩니다.

```
await page.waitForSelector('input#search');
console.log('Search input appeared');
```