Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики і обчислювальної техніки

Кафедра автоматики та управління в технічних системах

[Компоненти програмної інженерії - 3. Якість програмного забезпечення та тестування](http://wiki.kpi.ua/index.php/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%B8%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D1%97%20%D1%96%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D1%96%D1%97%20-%203.%20%D0%AF%D0%BA%D1%96%D1%81%D1%82%D1%8C%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%20%D1%82%D0%B0%20%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(20202020))

Практична робота №6

Тема: Тестування за допомогою Playwright, автоматизація, Playwright/tests, coverage

**Виконав:**

Студент гр. ІТ-83

Заярнюк М. О.

Київ – 2021

Тема: Тестування за допомогою Playwright, автоматизація, Playwright/tests, coverage

Мета завдання: Навчитися виконувати тестування за допомогою Playwright

Завдання:

1. Створити проект для Playwright на гіт хостингу
2. Подивитись на сайт <https://www.demoblaze.com/> і виділити 2-3 найголовніші сценарії
3. Додати в проект виділені сценарії
4. Створити Page Objects для кожного сценарію
5. Додати можливість збирати trace та video для впавших тестів (щоб перевірити це треба, щоб хоча б один тест впав:-)
6. Завдання із \*: спробувати включити коверейдж репорт і вивести його результати після тесту (і css і js )
7. Завдання із \*: додати в тест перевірку через скріншот (спочатку треба зберегти зображення як темплейт)

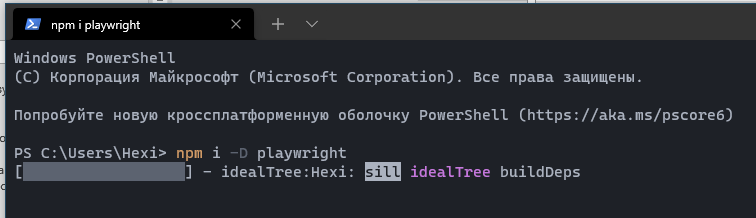
Хід роботи:

Наступним кроком буде встановлення npm пакету Playwright та @Playwright/Test для node.js,

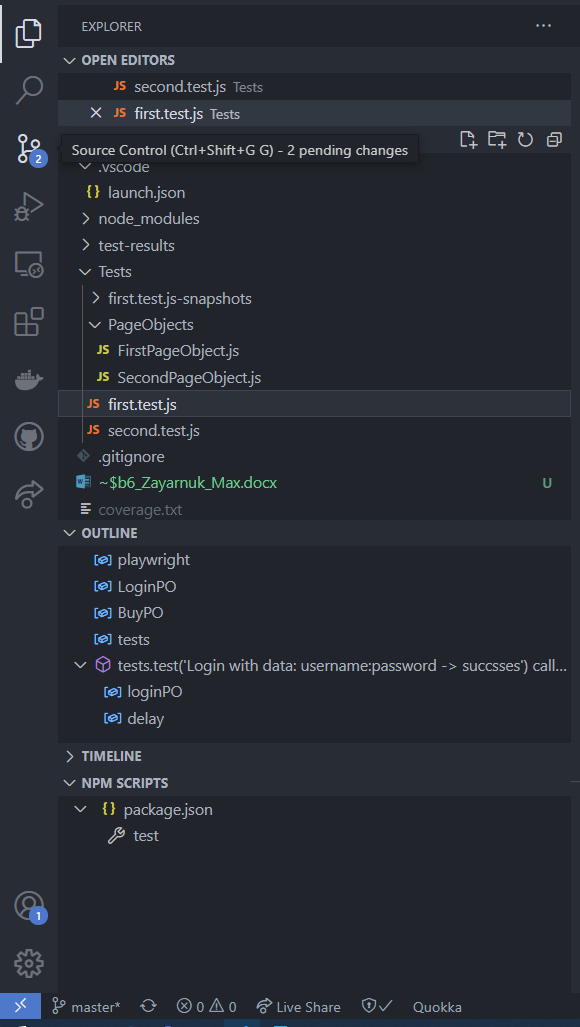
Зробити це можна виконавши наступну команду у powershell.

$ pm i -D playwright

$ pm i -D @Playwright/Test



Далі, переходимо до VS Code та створюємо і хостимо проект на гітхабі.



Я обрав два сценарії:

* Логін
* Покупка товару без логіну

Для кожного зі сценаріїв, створюємо Page Objects, лістнинг приведений нижче:

const jestpackage = require('jest');

class BuyPO {

*/\*\**

*\* @param {import('playwright').Page} page*

*\*/*

    constructor(*page*) {

      this.page = *page*;

    }

    async Navigate() {

        await this.page.goto('https://www.demoblaze.com/prod.html?idp\_=1');

        const delay = *ms* => new Promise(*resolve* => setTimeout(resolve, *ms*));

        await delay(5000);

    }

    async Buy()

    {

        await this.page.click('a[onclick^="addToCart"]');

    }

  }

  module.exports = { BuyPO };

class LoginPO {

*/\*\**

*\* @param {import('playwright').Page} page*

*\*/*

    constructor(*page*) {

      this.page = *page*;

    }

    async Navigate() {

        await this.page.goto('https://www.demoblaze.com/index.html');

    }

    async FillAndLogin(*usernmame*, *password*) {

      await this.page.click('id=login2');

      await this.page.fill('#loginusername', *usernmame*);

      await this.page.fill('#loginpassword', *password*);

      await this.page.keyboard.press("Tab");

      await this.page.keyboard.press("Tab");

      await this.page.keyboard.press("Enter");

      const delay = *ms* => new Promise(*resolve* => setTimeout(resolve, *ms*));

      await delay(2000);

    }

    async Validate()

    {

      const text = await this.page.innerText('id=nameofuser');

      return text;

    }

  }

  module.exports = { LoginPO };

Та додамо розділ scripts у package.json:

{

  "dependencies": {

    "expect": "^27.3.1",

    "playwright": "^1.16.3"

  },

  "scripts": {

    "test": "npx playwright test"

  },

  "devDependencies": {

    "@playwright/test": "^1.16.3",

    "jest": "^27.3.1"

  }

}

Це потрібно для того, щоб дати зрозуміти Node.js що означає команда test.

Нижче і вижче може бачити залежності проекту.

Після чого, додаємо конфіг файл для playwright:



*/\*\* @type {import('@playwright/test').PlaywrightTestConfig} \*/*

const config = {

    use: {

      headless: false,

      viewport: { width: 1280, height: 720 },

      channel: "chrome",

      video: 'retain-on-failure',

      trace: 'retain-on-failure'

    },

  };

  module.exports = config;

Дані параметри означають наступне:

headless: false, - параметр, що відповідає за показ браузеру під час тестування

viewport: { width: 1280, height: 720 }, - розмір браузеру

channel: "chrome", - тип браузеру (Google Chrome)

video: 'retain-on-failure', (запис відео, якщо тест провалився)

trace: 'retain-on-failure' (запис трейсу, якщо тест провалився)

Наступним кроком буде створення самих тестів, створюємо два файли, лістнинг яких приведений нижче:

*//Перевірка логіну*

const playwright = require('playwright');

const { LoginPO } = require('./PageObjects/LoginPageObject');

const { BuyPO } = require('./PageObjects/BuyPageObject');

const tests = require("@playwright/test");

tests.test('Login with data: username:password -> succsses', async({*page*}) =>

   {

      const loginPO = new LoginPO(*page*);

      await loginPO.Navigate();

      await loginPO.FillAndLogin("username","password");

      const delay = *ms* => new Promise(*resolve* => setTimeout(resolve, *ms*));

      await delay(1000);

      tests.expect(await loginPO.Validate()).toBe("Welcome username");

      tests.expect(await *page*.screenshot()).toMatchSnapshot('mainPage.png');

  });

*//Перевірка покупи*

const v8toIstanbul = require('v8-to-istanbul');

const playwright = require('playwright');

const { BuyPO } = require('./PageObjects/BuyPageObject');

const tests = require("@playwright/test");

  tests.test('No redirect to main page after buy -> failed', async({*page*}) =>

   {

    await *page*.coverage.startJSCoverage();

    const buyPO = new BuyPO(*page*);

    await buyPO.Navigate();

    await buyPO.Buy();

    const coverage = await *page*.coverage.stopJSCoverage();

*// Краще коментувати, адже вивід даних займає багато часу*

    for (const entry of coverage) {

      const converter = new v8toIstanbul('', 0, { source: entry.source });

      await converter.load();

      converter.applyCoverage(entry.functions);

      console.log(JSON.stringify(converter.toIstanbul()));

    }

    await tests.expect(*page*).toHaveURL('https://www.demoblaze.com/');

  });

Для того, щоб тест запустився, потрібно прописати його як метод test(), async дозволяє нам зробити його асинхронним.

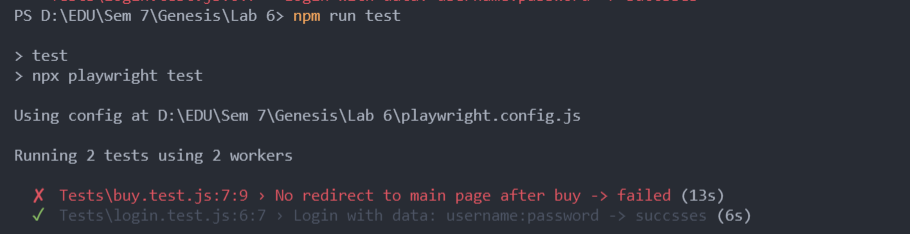
***ВАЖЛИВО:***

***\*файли потрібно називати name.test.js\*, щоб пакет побачив їх у папці.***

Далі ми просто викликаємо раніше створені Page Object, і порівнюємо отримане і бажане значення у expect() методі.  
Запустимо ці тести командою:

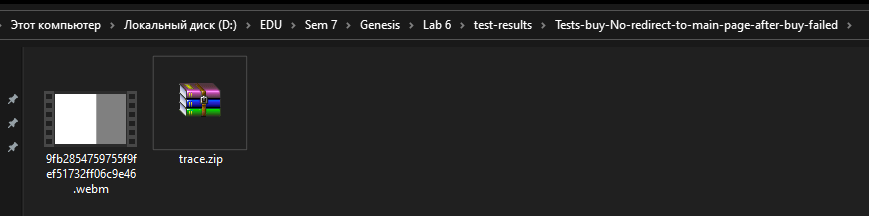
$ npm test

Ось такий вивід ми отримуємо:



Перший тест не проходить – і це очікуваний результат, другий проходить і це також очікуваний результат.

Після проходу тестування, у тих тестів що, не дали негативний результат, зберігається відео і трейс:



Також, в проект доданий coverage, у тесті покупки без логіну, ось код виконання:

*// Краще коментувати, адже вивід даних займає багато часу*

    for (const entry of coverage) {

      const converter = new v8toIstanbul('', 0, { source: entry.source });

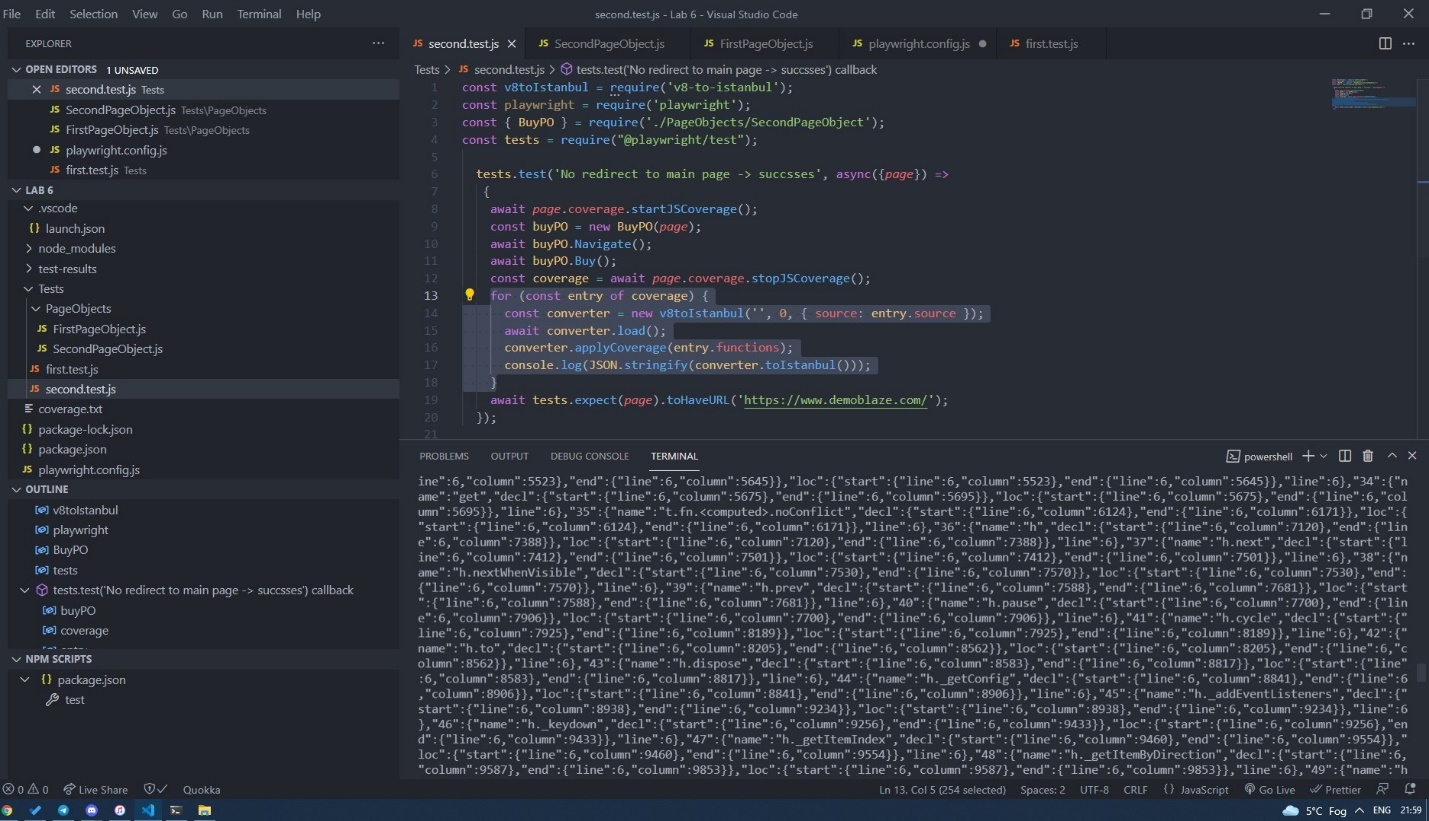
      await converter.load();

      converter.applyCoverage(entry.functions);

      console.log(JSON.stringify(converter.toIstanbul()));

    }

І його вивід:



А також, додатковий асерт результатів по скріншотам:

tests.expect(await *page*.screenshot()).toMatchSnapshot('mainPage.png');

***ВАЖЛИВО:***

***Потрібно розуміти, що снапшот mainPage.png створиться не спершого разу, тому на першій спробі такий тест провалиться, проте запрацює на другий раз.***



Висновки:

Сьогодні я навичився тестувати додатки за допомогою PlayWright, дізнався про і навчився використовувати css-селектори та відточив навички роботи з гітхабом. Також сторив асерти для тестування, самі тести та зрівнянв вивід тестів за скрінами, та додав coverage.