Haaga-Helia
University of Applied Sciences

Ohjelmistokehityksen teknologioita - Seminaarityö

Seminaarityö

Playwright hyödyntäminen end-to-end testauksessa web-sovelluksissa

Mikko Särkiniemi

14.4.2022

Sisältö

Tiivistelmä

Seminaarityöni keskittyy Playwright-työkalun käyttöön end-to-end (E2E) – testauksessa. Playwright on moderni, avoimen lähdekoodin kirjasto, joka mahdollistaa web-sovellusten automatisoidun testauksen useilla eri selaimilla. Työssäni hyödynnän Playwrightia testatakseni ohjelmistoprojekti 2 -kurssilla kehitetyn web-palvelun käyttöliittymää, joka käyttää tekoälyä mainoskuvien luomiseen huonekaluista.

Työn tavoitteena on osoittaa, kuinka Playwrightia voidaan hyödyntää realististen käyttötilanteiden testaamiseen. Työssä esitellään lyhyesti Playwrightin perusteet, sen tärkeimmät ominaisuudet, kuten useiden selaimien tuki ja tehokas virheenkorjaus, sekä käytetyt testimenetelmät. Tulen suorittamaan testejä, jotka kattavat palvelun kriittiset toiminnallisuudet, kuten mainoskuvien ja tekstien luomisen. Työn tulokset korostavat Playwrightin käytännöllisyyttä ja sen roolia luotettavan ohjelmistokehityksen tukena.

1 Johdanto/Ylätason esittely

Ohjelmistokehityksessä end-to-end-testaus on keskeisessä roolissa varmistettaessa sovelluksen toiminnallisuuden ja käyttökokemuksen laadukkuus. Työni keskittyy Playwrightin käyttöön tällaisen testauksen toteuttamisessa. Playwright on Microsoftin kehittämä työkalu, joka on suunniteltu web-sovellusten automaattiseen testaamiseen. Sen avulla voidaan testata sovelluksen toimintaa eri selaimilla ja käyttöympäristöissä, mikä tekee siitä erinomaisen valinnan nykyaikaisiin monialustaisiin kehitysprojekteihin.

Seminaarityöni tavoitteena on tutkia ja demonstroida Playwrightin käyttöä ohjelmistoprojekti 2 -kurssilla luodun web-palvelun käyttöliittymän testaamiseen. Tämä web-palvelu tarjoaa käyttäjille mahdollisuuden luoda tekoälypohjaisia mainoskuvia myytäville huonekaluille. Työssä käyn läpi Playwrightin perusteet, kuten sen konfiguraation ja tärkeimmät ominaisuudet. Tämän jälkeen keskityn demoon, jossa Playwrightia käytetään palvelun keskeisten toimintojen testaukseen.Playwright

2 Playwrightin perusteet

Playwright on moderni E2E-testausratkaisu, joka tukee useita selaimia, kuten Chrome, Friefox ja Safari. Se on suunniteltu erityisesti nopeaan ja luotettavaan testaukseen. Playwrightin keskeiset ominaisuudet ja hyödyt:

- **Monipuolinen selaintuki:** Playwright tulee Chormia, Webkitiä ja Firefoxia sekä paikallista ja CI/CD-ympäristöissä tapahtuvaa testausta. Myös mobiiliselainten emulointi on mahdollista.
- **Luotettava automaatio:** Automaattiset odotukset ja web-first-väitteet varmistavat testien vakauden ilman manuaalista viiveiden hallintaa.
- **Tehokkaat debug-työkalut:** HTML-raportit, aikamatkailu-debuggaus ja trace viewer tarjoavat yksityiskohtaista tietoa testeistä ja helpottavat virheiden korjausta.
- **Helppokäyttöinen testien luonti:** Työkalut, kuten *codegen,* helpottavat testien kirjoittamista ja lokaattoreiden määrittämistä tarkasti.
- **Integraatio ja ylläpidettävyys:** Tuki useille ohjelmointikielille ja CI-työkaluille sekä kestävä lokaattorimekanismi vähentävät ylläpitotarvetta.

2.1 Playwrightin käyttöönotto ja käyttö

Tässä on tiivistetty ohje Playwrightin käyttöönottoon ja käyttöön end-to-end testauksessa:

2.1.1 Playwrightin asennus

1. *Luo erillinen npm-projekti end-to-end-testejä varten.*
2. *Asenna Playwright komennolla:*
    - *npm init playwright@latest*
3. *Tämä luo Playwrightille tarvittavat tiedostot ja konfiguraatiot*

2.1.2 Testattavan sovelluksen käynnistäminen

Playwhright ei itsessään käynnistä testattavaa sovellusta automaattisesti, joten varmista, että sovellus on käynnissä testauksen aikana.

2.1.3 npm-skriptien lisääminen

Lisää package.json-tiedostoon seuraavat skriptit:

```
  "scripts": {
    "test": "playwright test",
    "test:report": "playwright show-report"
  },
```

2.2 Playwright esimerkkitestit

Playwright luo automaattisesti esimerkkitestejä projektin luomisen yhteydessä:

tests > example.spec.js > ...

```
// @ts-check
const { test, expect } = require('@playwright/test');

test('has title', async ({ page }) => {
  await page.goto('https://playwright.dev/');

  // Expect a title "to contain" a substring.
  await expect(page).toHaveTitle(/Playwright/);
});

test('get started link', async ({ page }) => {
  await page.goto('https://playwright.dev/');

  // Click the get started link.
  await page.getByRole('link', { name: 'Get started' }).click();

  // Expects page to have a heading with the name of Installation.
  await expect(page.getByRole('heading', { name: 'Installation' })).toBeVisible();
});
```

Testit voi suorittaa komennolla:

npm test

Tulokset ilmoitetaan näin:

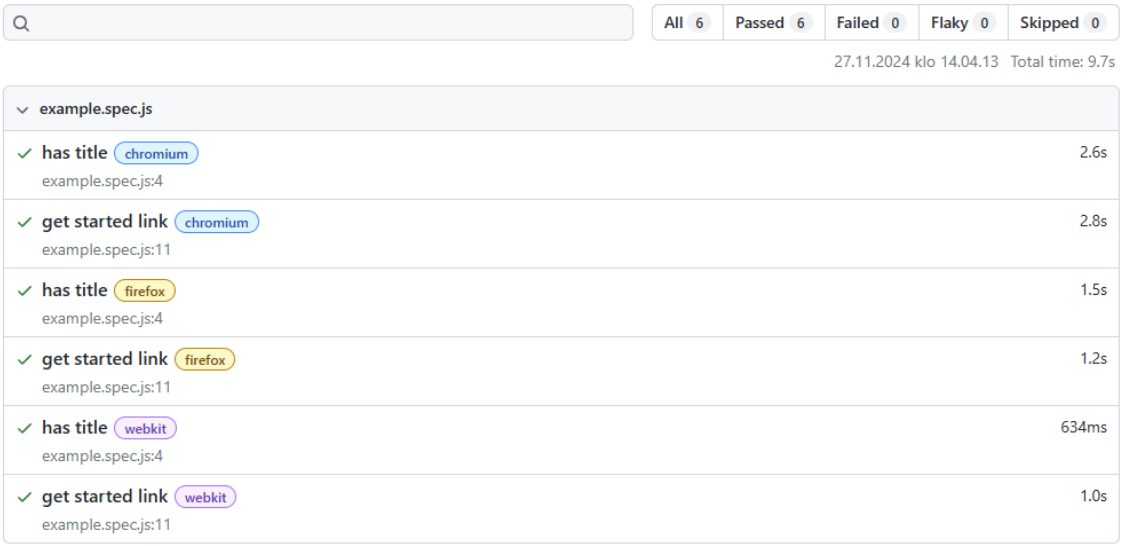
```
> playwrighttests@1.0.0 test
> playwright test


Running 6 tests using 6 workers
  6 passed (9.7s)

To open last HTML report run:

  npx playwright show-report
```

2.3 Tarkemman testiraportin tarkastelu

Generoi yksityiskohtainen raportti komennolla:

npm run test:report

Tämä avaa selaimessa näkymän, jossa testien yksityiskohtia voi tarkastella tarkemmin:



2.4 Testien Suorittamisesta ja tarkastelusta lisää

2.4.1 Testiympäristön valinta

Playwright suorittaa testit oletuksena kolmessa eri selaimessa (Chromium, Firefox ja WebKit), kuten ylemmästä kuvasta näkyy. Tämä on hyödyllistä pienellä testimäärällä, mutta suuremmalla testimäärällä suoritusaika voi kasvaa merkittävästi, koska testit ajetaan jokaisessa selaimessa erikseen.

Ratkaisu tähän on valita haluttu selain testiä suoritettaessa. Esimerkiksi, jos testit halutaan suorittaa vain Chromella, käytetään seuraavaa komentoa:

```
npm test -- --project=chromium
```

2.4.2 Playwrightin käyttöliittymätila (UI-mode)

Playwright tarjoaa oman käyttöliittymän (UI-mode), jossa testejä voidaan suorittaa ryhmissä (esim. describe-lohkon sisällä olevat testit) tai yksitellen. Lisäksi suoritettujen testien tuloksia ja yksityiskohtia voi tarkastella selkeämmin.

Käyttöliittymän avaaminen tapahtuu seuraavalla komennolla:

a. *npm test -- --ui*

Tämä toiminto mahdollistaa tehokkaamman testitulosten analysoinnin ja nopeuttaa kehitystyötä. Tulemme demonstroimaan sen toimintaa tarkemmin luvussa 2.3 Testien tulokset

3 Testien luominen

Seuraavaksi luomme testejä, jotka kattavat sovelluksen käyttöliittymän toiminnallisuuden sekä backend-yhteyden tarkistamisen. Testien avulla varmistamme seuraavat asiat:

1. Etusivun elementit ja toiminnot näkyvät odotetusti.
2. Etusivulta voidaan navigoida mainoskuvan luomiseen tarkoitetulle lomakesivulle.
3. Lomakkeen täyttäminen ja lähettäminen onnistuu oikein.
4. Backend vastaa odotetusti lomakkeen lähetyksen jälkeen.

Luomme uuden tiedoston, appTests.spec.js, jossa nämä testit määritellään. Tiedosto lisätään siis aiemmin luotuun Playwright -projektiin.

3.1 Luodut testit

Jokainen testikokonaisuus voidaan sijoittaa omaan describe-lohkoonsa, mikä helpottaa testien suorittamista ja tulosten tarkistamista. Tällä tavalla testit pysyvät loogisesti ryhmiteltyinä ja ovat helpommin ylläpidettäviä.

```
8   // Määrittele testisarja nimeltä 'UI tests'
9   describe('UI tests', () => {
10
11      //Testit tulevat tähän:
12
```

Lisätään beforeEach-lohko, joka varmistaa, että jokaisen testin alussa siirrytään oikealle sivulle. Tämä vähentää koodin toistoa ja tekee testeistä selkeämpiä ja helpommin ylläpidettäviä.

```
3   // Koukku, joka suoritetaan ennen jokaista testiä
4   beforeEach(async ({ page, request }) => {
5       // Siirry määriteltyyn URL-osoitteeseen ennen jokaista testiä
6       await page.goto('http://localhost:5173');
7   });
```

3.1.1 Etusivun otsikon renderöitymisen testi

```
13  // Määrittele testi nimeltä 'is there h1 header'
14  test('is there h1 header', async ({ page }) => {
15      // Varmista, että elementti, joka sisältää tekstin 'Luo upeita mainoskuvia hetkessä', on näkyvissä
16      await expect(await page.getByText('Luo upeita mainoskuvia hetkessä')).toBeVisible();
17  });
18
```

Etusivulta löytyy seuraava otsikko.

Seuraava testi tarkistaa, että tämä otsikko renderöityy oikein. Kommentit koo-

```
<h1>Luo upeita mainoskuvia hetkessä</h1>
```

dissa selittävät testin toimintaa.

3.1.2 Stabilityform 1 -lomakkeen saatavuuden testi

Etusivulla on nappi, joka navigoi lomakesivulle (Stabilityform 1). Testissä varmistetaan, että nappi löytyy, sitä voidaan painaa, ja navigoinnin jälkeen lomakesivun otsikko renderöityy.

```
19      // Määrittele testi nimeltä 'is there button to get to stability 1 form'
20      test('is there button to get to stability 1 form', async ({ page }) => {
21          // Etsi painike, jonka nimi on 'Luo mainoskuvasi stability1', ja klikkaa sitä
22          await page.getByRole('button', { name: 'Luo mainoskuvasi stability1' }).click();
23          // Varmista, että elementti, joka sisältää tekstin 'Stability AI 1 Form', on näkyvissä
24          await expect(await page.getByText('Stability AI 1 Form')).toBeVisible();
25      });
26  });
```

3.1.3 Lomakkeen täyttö ja backend-vastauksen tarkistus

Testataan lomakkeen toiminnallisuus siten, että:

1. Lomake täytetään Playwrightin avulla.
2. Lomake lähetetään backendille.
3. Tarkistetaan, että vastaus saadaan, ja kuva renderöityy näytölle.

Testissämme lisäämme odotusajan, koska backendin vastaus kestää noin 25 sekuntia.

```
// Määrittele testisarja nimeltä 'image generation tests'
describe('image generation tests', () => {
    // Määrittele testi nimeltä 'can I generate image'
    test('can I generate image', async ({ page }) => {
        // Etsi painike, jonka nimi on 'Luo mainoskuvasi stability1', ja klikkaa sitä
        await page.getByRole('button', { name: 'Luo mainoskuvasi stability1' }).click();
        // Täytä tekstikenttä, jonka testitunnus on 'testPrompt', tekstillä 'Tuoli Testi'
        await page.getByTestId('testPrompt').fill('Tuoli Testi');
        // Aseta tiedosto 'chair.png' tiedostokenttään, jonka testitunnus on 'testImg'
        await page.getByTestId('testImg').setInputFiles('chair.png');
        // Etsi painike, jonka nimi on 'Generoi mainos', ja klikkaa sitä
        await page.getByRole('button', { name: 'Generoi mainos' }).click();
        // Odota, että elementti, jonka testitunnus on 'generatedImage', tulee näkyviin (aikakatkaisu 35 sekuntia)
        const image = await page.waitForSelector('img[data-testid="generatedImage"]', { timeout: 35000 });
        // Hae 'src'-attribuutin arvo kuvasta
        const src = await image.getAttribute('src');

        // Tarkista, että kuvan src-attribuutti ei ole tyhjä
        expect(src).not.toBe('');
        expect(src).not.toBeNull();
    });
});
```

3.2 Testien ajo ja tulokset

3.2.1 Testien ajo

Kun haluamme suorittaa testit vain tietyllä selaimella, kuten Chromella, se onnistuu seuraavalla komennolla:

npm test -- --project chromium

Tämän jälkeen komentorivi näyttää testien tulokset:

```
> playwrighttests@1.0.0 test
> playwright test --project chromium


Running 3 tests using 3 workers
  3 passed (24.1s)

To open last HTML report run:

  npx playwright show-report
```

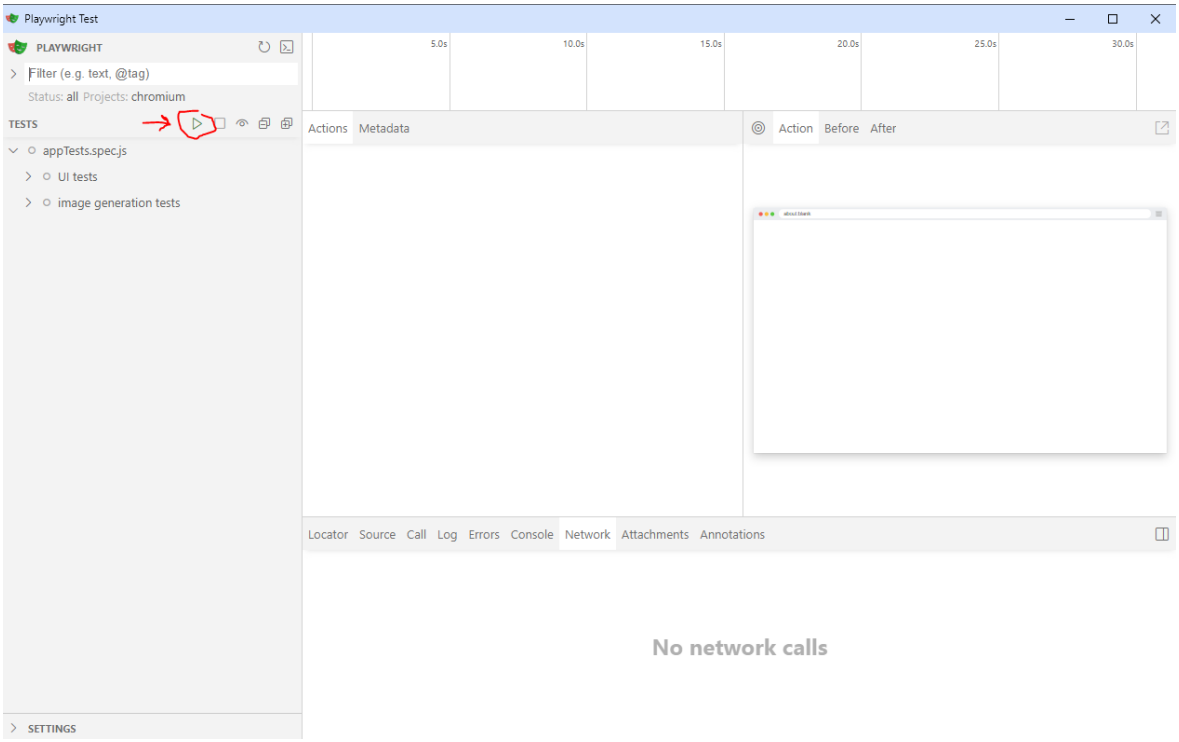
Esimerkiksi, onnistuneet testit merkitään vihreällä, ja mahdolliset epäonnistuneet testit saavat selkeän virheilmoituksen. Tämä auttaa tunnistamaan ongelmat nopeasti.

3.2.2 Playwright käyttöliittymätila (UI-mode)

Playwright tarjoaa myös oman käyttöliittymätilan, jonka avulla testien suorittaminen ja analysointi on intuitiivisempaa. UI-tilan voi avata siis komennolla:

npm test -- --ui

Tämä komento avaa graafisen käyttöliittymän, jossa näet listan kaikista testeistä. Testit voi suorittaa yksitellen tai ryhmissä painamalla ”Play”-painiketta.

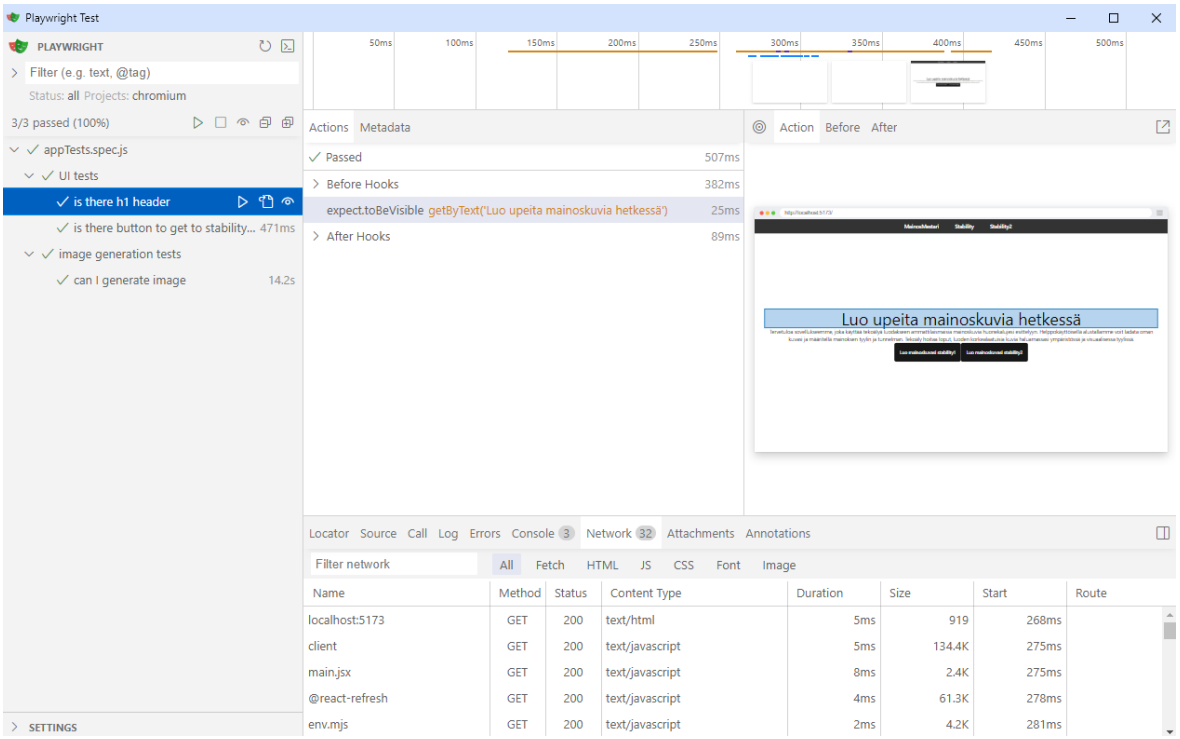


Käyttöliittymä tarjoaa myös seuraavat ominaisuudet:

- **Testien tulosten tarkastelu**
  - Jokaisen suoritetun testin viereen ilmestyy merkintä, joka kertoo onnistuiko testi vai epäonnistuiko se.

- **Testivaiheiden analysointi**
  - Suoritetuista testeistä voi tarkastella yksityiskohtaisesti jokaista vaihetta, kuten elementtien löytämistä ja käyttäjän toimintoja.

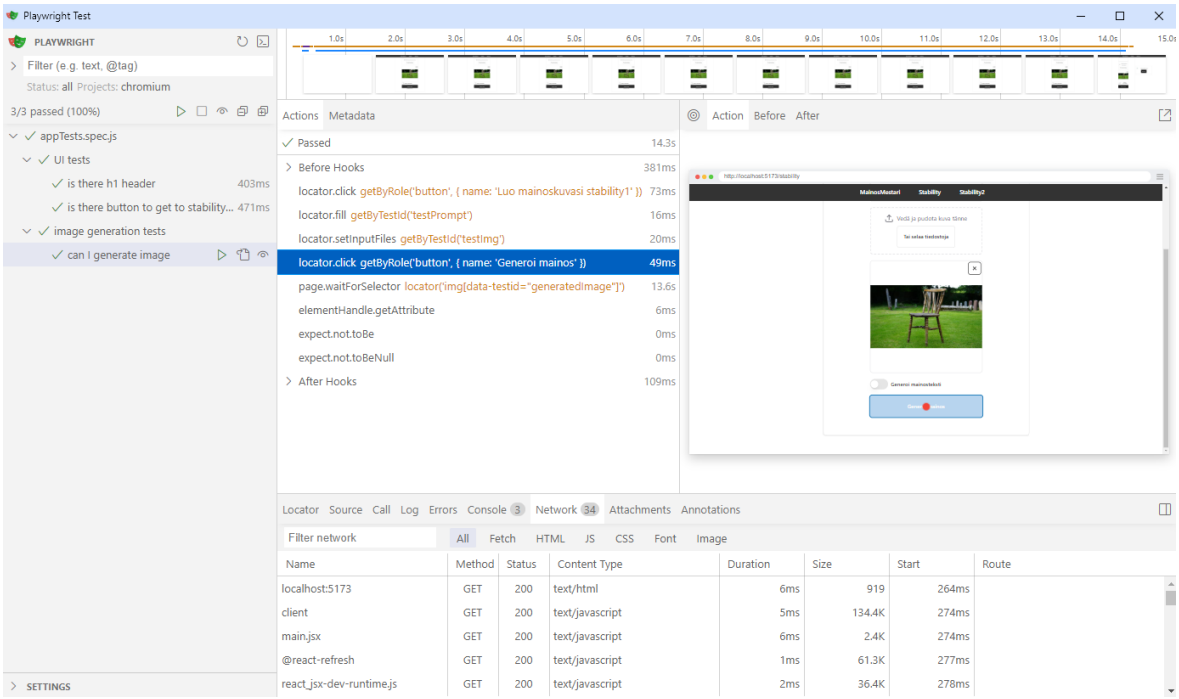
- **Käyttöliittymän tallennettu tila**
  - Playwright näyttää testauksen aikana sovelluksen käyttöliittymän tilan, mikä helpottaa ongelmien jäljittämistä.



3.2.3 Esimerkki: Lomakkeen täytön ja lähetyksen analysointi

Jos suoritat ”image generation test”-testin, voit UI-tilassa tarkastella esimerkiksi lomakkeen täyttövaheita. Käyttöliittymä näyttää reaaliajassa seuraavat tiedot:

1. Mitä testi etsii
    a. Esimerkiksi syötettäviä kenttiä tai nappeja.
2. Löytyvätkö elementit
    a. Näet visuaalisen korostuksen, joka osoittaa, mitkä elementit Playwright löytää ja käyttää.
3. Lomakkeen lähetys
    a. Voit tarkistaa, miten data välitettiin backendille ja mitä vastaus sisältää.



Ui-tilan avulla testauksen tarkastelu on huomattavasti yksinkertaisempaa ja tehokkaampaa kuin pelkän komentorivin kautta. Tämä tekee siitä erityisen hyödyllisen monimutkaisissa testeissä tai virheiden debuggaamisessa.

4 Yhteenveto ja johtopäätökset

Tässä seminaarityössä tutkittiin Playwrightin soveltuvuutta end-to-end-testaamiseen ohjelmistoprojekti 2 -kurssilla kehitetyn tekoälypohjaisen web-palvelun käyttöliittymän ja backend-yhteyden testaukseen. Työssä saavutettiin useita merkittäviä löydöksiä, jotka osoittavat Playwrightin hyödyllisyyden ja tehokkuuden modernin ohjelmistokehityksen tukena.

4.1 Löydökset

- *Etusivun ja navigoinnin toimivuus*
  - *Testit varmistivat käyttöliittymän keskeiset elementit ja toimintojen odotetun toiminnan.*
- *Lomakkeen ja backend-yhteyden testaus*
  - *Lomake täytettiin ja lähetettiin onnistuneesti ja testit vahvistivat backendin palauttaman vastauksen.*
- *UI-tilan hyödyt*
  - *Playwrightin käyttöliittymätila helpotti testitulosten analysointia ja virheiden debuggaamista.*

4.2 Johtopäätökset

Playwright parantaa testien tehokkuutta ja ylläpidettävyyttä tarjoamalla laajan selaintuen, helppokäyttöiset debug-työkalut ja yksityiskohtaiset raportit. Se sopii erityisesti monialustaisten web-sovellusten laadunvarmistukseen.

4.3 Oma oppiminen

Työn aikana opin tehokkaasti hyödyntämään Playwrightin keskeisiä ominaisuuksia, kuten automatisoitujen testien luontia. Opin myös, kuinka tärkeää on luoda selkeitä ja ylläpidettäviä testikokonaisuuksia, jotka vastaavat sovelluksen todellisia käyttötapauksia. Playwrightin UI-mode tarjosi hyvää kokemusta testitulosten tarkemmasta analysoinnista ja debuggaamisesta, mikä oli erityisen hyödyllistä monimutkaisempien testien aikana.

4.4 Kehitysideat

Tulevaisuudessa Playwrightia voisi hyödyntää laajempien testikokonaisuuksien, CI/CD-integraatioiden ja mobiilialustojen testauksessa.

Linkki seminaari videoon: SeminaariVideo.webm

Lähdeluettelo

Playwright. s.a. Playwright Documentation. Luettavissa: https://playwright.dev/. Luettu: 28.11.2024.

Fullstack Open. s.a. End-to-end-testaus Playwrightilla. Luettavissa: https://fullstackopen.com/osa5/end_to_end_testaus_playwright#playwright. Luettu: 28.11.2024.