SAVONIA
ammattikorkeakoulu

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

WEB-SOVELLUKSEN KOKONAIS-VALTAISEN TESTAUKSEN SUUN-NITTELU SEKÄ TOTEUTUS

TEKIJÄ Juha-Pekka Korhonen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Juha-Pekka Korhonen	
Työn nimi Web-sovelluksen kokonaisvaltaisen testauksen suunnittelu sekä toteutus	
Päiväys 4.4.2023	Sivumäärä/Liitteet 21
Toimeksiantaja/Yhteistyökumppani(t) Spine Networks Oy	

Tiivistelmä

Työn tavoitteena oli suunnitella ja luoda mahdollisimman kattava automatisoitu testaaminen jo olemassa olevalle web-sovellukselle, jonka tarkoitus on estää virhetilanteiden pääseminen sovelluksen tuotantoversioon. Testaaminen suoritettiin päästä päähän -menetelmällä eli testeillä pyrittiin simuloimaan oikean käyttäjän toimintoja sovelluksessa mahdollisimman tarkasti.

Sovelluksessa on yleisesti pyritty käyttämään mahdollisimman nykyaikaisia työkaluja, joten tämä tuli ottaa huomioon jo työn suunnitteluvaiheessa. Työssä päädyttiin käyttämään Cypress-nimistä testaustyökalua, joka täytti valintakriteerit parhaiten testaustyökalujen vertailussa.

Työn lopputulokseksi saatiin lähes koko sovelluksen kattava automatisoitu testaaminen. Automatisoidut testit käyvät kaikki sovelluksen komponentit läpi ja testaa niiden toiminnallisuuden onnistuneesti. Testit myös pystyvät hakemaan, lisäämään, editoimaan sekä poistamaan dataa tietokannasta.

Avainsanat
Testaaminen, web-sovellus, päästä päähän -testaus, Cypress

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Information Technology	
Author(s) Juha-Pekka Korhonen	
Title of Thesis Planning and implementing comprehensive testing of the web application	
Date 4 April 2023	Pages/Appendices 21
Client Organisation /Partners Spine Networks Oy	

Abstract

The aim of this thesis was to plan and create an automated testing procedure which is as comprehensive as possible for an already existing web application. The purpose was to prevent error situations from reaching the production version of the application. The testing procedure was done with an end-to-end method, which means that the test cases try to simulate the actions of a real user in the application as accurately as possible.

In the web application the aim is to use the most modern tools available. This had to be considered already in the planning phase of the work. It was decided to use a testing tool called Cypress, which best fulfilled the selection criteria best in the comparison of testing tools.

As a result, the automated testing procedure covering almost the entire application was created. All the components of the application are tested automatically and their functionality is tested successfully. Data from the database can also be retrieved, added, edited and deleted using these tests.

Keywords
Testing, web application, end-to-end, Cypress

SISÄLTÖ

KUVALUETTELO

1 JOHDANTO

Ohjelmiston testaaminen voi jäädä ohjelmistokehityksessä pienemmälle huomiolle, koska ajatellaan, että se ei juurikaan edistä itse ohjelmistokehitystä. Tämä ajattelutapa on yleensä väärä, koska laaja testaaminen auttaa löytämään virheitä sekä estämään näiden virheiden pääsyn tuotanto versioon. Tämä on tärkeää laadunhallinnan kannalta, koska on helpompaa korjata virheet heti ohjelmistoa kehittäessä kuin huomata ne liian myöhään, jolloin korjaukset voivat viedä todella paljon aikaa.

Opinnäytetyön tilaaja on Spine Networks Oy. Opinnäytetyö on osa jo olemassa olevan projektin kehitystyötä, jossa olen ollut jo mukana noin puoli vuotta. Web-sovellus, jota testataan, on osa projektia, johon kuuluu myös laite, jonka sensoreilla kerätään erilaista dataa rakennustyömaiden sähkökeskuksista. Sovelluksessa pystytään hallitsemaan laitetta käyttävien yrityksien tietoja, niiden käyttäjiä, projekteja sekä tietysti seuraamaan laitteesta tulevaa dataa.

Opinnäytetyössä on tarkoituksena simuloida oikean käyttäjän toimintaa sovelluksessa automatisoitujen testien avulla. Testien tulee pystyä hakemaan, lisäämään, editoimaan sekä poistamaan dataa tietokannasta sekä tarkistamaan, että sovelluksen kaikki komponentit toimivat oikein. Työn edetessä esille nousi myös testien kattavuuden seuranta.

2 KÄSITTEET

NODE.JS

Avoimen lähdekoodin alustariippumaton ajoympäristö JavaScript-koodin suorittamiseen palvelimella, joka pohjautuu Googlen Chrome V8 JavaScript-moottoriin.

REFAKTOROINTI

Prosessi, jossa tietokoneohjelman lähdekoodia muutetaan siten, että sen sisäinen rakenne muuttuu, mutta toiminnallisuus säilyy. Tarkoituksena on parantaa koodin laatua sekä tehdä siitä luettavampaa ja helpommin ylläpidettävää.

CHROMIUM

Googlen avoimen lähdekoodin verkkoselainprojekti. Googlen Chrome-selaimen lisäksi moni muu selain pohjautuu Chromiumiin.

RAJAPINTA

Rajapinta ohjelmistokehityksessä on komponenttien ja moduulien välinen raja. Rajapinta on määritelmän, jonka mukaan voidaan tehdä pyyntöjä ohjelmistolle, josta halutaan noutaa tai tuoda tietoja. Ohjelmointirajapinnat voivat toimia ohjelmien välillä tai ohjelmien ja laitteiston välillä.

DEVTOOLS-PROTOKOLLA

Mahdollistaa työkalut Chromiumiin pohjautuvien selainten instrumentointiin, tarkastamiseen, virheenkorjaukseen sekä profilointiin.

BASEURL

Web-sovelluskehityksessä määritelty verkkosivuston etusivun osoiteriviltä löytyvä verkko-osoite.

CONFIG-TIEDOSTO

Sisältää parametreja sekä asetuksia ohjelman rakentamista ja suorittamista varten.

SKRIPTI

Ohjelma tai komentosarja, jonka toinen ohjelma tulkitsee tai suorittaa.

LOCALSTORAGE

Verkkotallennustila, jonka avulla verkkosivustot voivat tallentaa pysyviä tietoja käyttäjien laitteille, kuten evästeitä ilman viimeistä käyttöpäivää. Evästeillä tarkoitetaan pieniä tekstitiedostoja, joita sivusto tallentaa tietokoneellesi tai mobiililaitteellesi.

ACCESSTOKEN

Sisältää kirjautumisistunnon suojaustiedot ja tunnistaa käyttäjän.

DEBUGGAUS

Virheenjäljitys, joka on ohjelmistotuotannon osa, jossa testauksessa löytyneen virheellisen toiminnan aiheuttanut virhe paikallistetaan.

INSTRUMENTOINTI (OHJELMISTO KEHITYKSESSÄ)

Suorituskyvyn mittaamista virheiden diagnosoimiseksi ja jäljitystietojen kirjoittamiseksi.

REACT

Facebookin kehittämä JavaScript-kirjasto, joka on tarkoitettu käyttöliittymien kehittämiseen.

GITLAB

Verkkopalvelu, joka tarjoaa Git-versiohallinnan, wikin, bugienseurannan, kehitystoiveet ja tehtävienhallintatoimintoja.

FRONTEND

Verkkosivuston tai sovelluksen osa, jonka kanssa käyttäjä on suoraan vuorovaikutuksessa.

BACKEND

Sovelluksen tai ohjelman koodin osa, joka vastaa sovelluksen logiikasta sekä datan tallentamisesta ja käsittelystä.

CI/CD

Jatkuva integrointi (CI) ja jatkuva julkaisu (CD) putkisto on sarja vaiheita, jotka on suoritettava uuden ohjelmistoversion toimittamiseksi.

3 TESTAAMINEN OHJELMISTOKEHITYKSESSÄ

3.1 Yleistä

Ohjelmiston testaus on tapa tarkistaa, että vastaako todellinen ohjelmistotuote odotettuja vaatimuksia ja onko ohjelmistotuote virheetön. Testaus yleensä sisältää ohjelmiston komponenttien suorittamista joko manuaalisilla tai automatisoiduilla työkaluilla. Testauksen tarkoituksena on tunnistaa virheet, puutteet tai puuttuvat vaatimukset sekä ratkaista ne mahdollisimman varhaisessa vaiheessa ennen ohjelmistotuotteen toimittamista. Asianmukaisesti testattu ohjelmistotuote takaa luotettavuutta, turvallisuutta sekä lisää kustannustehokkuutta ja asiakastyytyväisyyttä. (Hamilton, 2022)

Ohjelmiston testaus on yleensä erotettu muusta kehityksestä. Testaaminen yleensä tehdään ohjelmistokehityksen kehitysprosessin myöhäisessä vaiheessa joko tuotteen rakennus- tai toteutusvaiheen jälkeen. Testaajalla voi olla todella vähän aikaa testata sovellusta ennen kuin sovellus julkaistaan markkinoille. Tästä syystä vikojen löytymiselle sekä korjaamiselle jää myös todella vähän aikaa. Tämä aiheuttaa yleensä sen, että joko sovellus julkaistaan ajallaan virheiden kanssa tai virheet korjataan, mutta sovelluksen julkaisu viivästyy. Kumpikaan näistä tilanteista ei ole ideaalinen sovellusta tuottavalle yritykselle. Testitoimintojen tekeminen kehitysprosessin aikaisemmassa vaiheessa auttaa ehkäisemään näitä tilanteita. (IBM)

Monet ohjelmistokehitystiimit käyttävät nykyään jatkuvana testauksena tunnettua menetelmää. Jatkuvan testauksen tavoitteena on nopeuttaa ohjelmiston toimitusta ja samalla tasapainottaa kustannukset, laatu ja riskit. Tällä testaustekniikalla tiimien ei tarvitse odottaa ohjelmiston rakentamisvaiheen loppumista ennen testauksen alkamista. Testejä voidaan suorittaa paljon aikaisemmin, jolloin vikoja löydetään nopeammin ja ne on helpompi korjata. (IBM)

Ohjelmiston testaus noudattaa yleistä prosessia. Tehtäviä tai vaiheita ovat testiympäristön määrittely, testitapausten kehittäminen, skriptien kirjoittaminen, testitulosten analysointi ja vikaraporttien lähettäminen. Pienemissä projekteissa manuaalinen testaaminen yleensä riittää, mutta suuremmissa projekteissa tehtävät automatisoidaan. Hyvä testaus kattaa sovellusohjelmointirajapinnan käyttöliittymän sekä järjestelmätasot. (IBM)

3.2 End To End -testaus

Opinnäytetyössä käytettiin End To End eli suomeksi päästä päähän -testausmenetelmää. Päästä päähän -testaaminen on ohjelmistojen testausmenetelmä, jolla varmistetaan, että sovellukset toimivat odotetulla tavalla ja että tietovirtaa ylläpidetään käyttäjän tehtäviä sekä prosesseja varten. Tämän tyyppisen testauksen tarkoituksena on siis simuloida sovelluksen käyttöä oikean käyttäjän näkökulmasta. Päästä päähän -testaus on hyvä tapa testata ohjelmistoja, mutta siihen liittyy myös omat haasteensa. Haasteena voi tulla vastaan esimerkiksi se, että testaaminen on aikaa vievää. Tämä tietysti riippuu testattavan sovelluksen koosta. Testien tekeminen edellyttää myös hyvää ymmärrystä käyttäjien tavoitteista ja ne on suunniteltava vastaamaan todellisia skenaarioita. (Schmitt, 2022)

Web-sovellusten päästä päähän -testaaminen tapahtuu selainta käyttäen jonkin työkalun avulla. Esimerkkejä työkaluista ovat Selenium, Cypress sekä Puppeteer. Päästä päähän -testauksia on kahta eri muotoa. On manuaalinen testaus sekä automaattinen testaus.

Manuaalisen testauksen suorittaa ihmistesteri, joka on suoraan vuorovaikutuksessa testausohjelmiston kanssa. Testaajat voivat nopeasti oppia, mikä toimii ja mikä ei testaussuunnitelmaa kirjoittaessa. Manuaalinen testaus auttaa tunnistamaan testitapaukset ja auttaa testaajia paljastamaan piilotetut käyttäjävuorovaikutus polut järjestelmässä. Tämä antaa testaajille tarvittavat tiedot, jotta he voivat aloittaa näiden testitapausten automatisoinnin tulevaisuudessa. (Schmitt, 2022)

Kun projektin koko kasvaa, kaikkien päästä päähän -testien suorittaminen manuaalisesti tulee olemaan vähemmän hallittavissa. Tämä pätee erityisesti käyttöliittymien testaamiseen, koska käyttöliittymän toiminto voi johtaa moniin muihin toimintoihin. Tämä monimutkaisuus tekee testausautomatisaatiosta välttämätöntä. Päästä päähän -testit voivat auttaa automatisoimaan käyttäjävuorovaikutteista testausta. Kun testitapauksista on päätetty, ne voidaan kirjoittaa koodiksi ja integroida automaattiseen testaustyökaluun. (Schmitt, 2022)

Koska ohjelmistot hankkivat uusia ominaisuuksia todella nopeasti, on testien automatisointi järkevä vaihtoehto. Automaatio mahdollistaa virheiden havaitsemisen nopeammin, koska koko koodikanta tarkistetaan automaattisesti uutta koodia lisättäessä. Testien automatisoinnilla on siis mahdollista säästää todella paljon aikaa. (Schmitt, 2022)

4 END TO END -TESTAUSTYÖKALUJA

4.1 Selenium

Selenium on avoimen lähdekoodin työkalu, joka tukee selainautomaatiota. Selenium tarjoaa käyttöliittymän, jolla pystytään kirjoittamaan testejä useilla eri ohjelmointikielillä esimerkiksi Javalla, Rubyllä, PHP:lla, Perlillä, Pythonilla, C# tai Node.JS:llä. Selenium jakaantuu kolmeen osaan: WebDriver, IDE sekä Grid. Selenium WebDriver käyttää selainten valmistajien selainautomaatiorajapintoja selaimen hallitsemiseen sekä testien suorittamiseen. Tämä toimii niin kuin todellinen käyttäjä käyttäisi selainta. Selenium IDE on työkalu, jota käytetään Selenium testien kehittämiseen. Se on helppokäyttöinen Chrome -ja Firefox-laajennus, joka on myös tehokkain tapa kehittää testitapauksia. Selenium Grid puolestaan mahdollistaa sen, että testejä voidaan suorittaa eri koneissa eri alustoilla. WebDriver testien kehittämisen jälkeen saatetaan joutua suorittamaan testejä useilla selaimilla ja eri käyttöjärjestelmäyhdistelmillä. Tämä on mahdollista Gridin avulla. (BrowserStack; Selenium, 2022)

Etuna Seleniumissa on se, että se toimii useissa eri verkkoselaimissa kuten Google Chromessa, Mozilla Firefoxissa, Safarissa ja Internet Explorerissa. Se myös tarjoaa käyttäjäystävällisen käyttöliittymän, jonka avulla testiskriptien kirjoittaminen ja suorittaminen on helppoa sekä erinomaisen näkymän päästä päähän -testaamiseen. Yksi Seleniumin suurimmista eduista on sen joustavuus, jota tehostaa testitapausten uudelleenryhmittely sekä refaktorointi. (Bhamra, 2023)

4.2 Cypress

Cypress on täysin JavaScript-pohjainen käyttöliittymätestaustyökalu, joka tukee Chrome-, Firefox- sekä Edge-selaimia. Cypress on kehittäjäystävällinen työkalu, joka käyttää uniikkia DOM-manipulaatiotekniikkaa ja toimii suoraan selaimessa. Se myös tarjoaa ainutlaatuisen interaktiivisen testisuorittimen missä se suorittaa kaikki sille annetut komennot. Cypressilla voidaan suorittaa yksikkötestejä, integraatiotestejä sekä päästä päähän -testejä. Testiskripteihin ei tarvitse lisätä odotuskomentoja, koska Cypress osaa odottaa komentoja automaattisesti. (Unadkat, 2022)

Cypress eroaa muista testaustyökaluista kuten Seleniumista siten, että sitä ajetaan samassa ajosilmukassa kuin sovellustakin. Cypressin takana on Node-palvelinprosessi. Cypress ja Node-prosessi kommunikoivat jatkuvasti, synkronoivat sekä suorittavat tehtäviä toistensa puolesta. Molempien osien käyttö antaa mahdollisuuden vastata sovelluksen tapahtumiin reaaliajassa ja samalla työskennellä selaimen ulkopuolella tehtävissä, jotka vaativat suurempia etuoikeuksia. Cypress myös sisältää monia ominaisuuksia, joita ei muista testaustyökaluista löydy. Esimerkiksi, koska Cypress on asennettu paikallisesti koneellesi, voi Cypress tallentaa testien suorittamisen aikana tilannekuvia, jonka avulla kehittäjä näkee tarkalleen mitä kyseisessä vaiheessa on tehty. Cypress ei myöskään käytä Seleniumia tai WebDriveria, joka takaa nopean, johdonmukaisen ja luotettavan testauksen. Cypressista löytyy myös verkkoliikenteen hallinta, joka hallitsee, käsittelee ja testaa tapauksia ilman palvelinta. (Cypress, 2023)

4.3 Puppeteer

Puppeteer on Googlen kehittämä Node-kirjasto, joka tarjoaa korkean tason rajapinnan päättömän Chromen tai Chromiumin ohjaamiseen DevTools-protokollan kautta. Päättömällä selaimella tarkoitetaan tavallista selainta, mutta näytöllä ei näy mitään. Ohjelma toimii taustajärjestelmässä eli se ei ole näkyvissä, mutta siinä on silti toimintoja. Kun käytämme normaalia selainta, näemme ohjelman jokaisen vaiheen graafisessa käyttöliittymässä, kun taas päättömässä selaimessa ohjelman vaiheita voidaan seurata joko konsolin tai komentorivin kautta. (Chrome Developers 2018; Lambdatest, 2023)

Puppeteer voi myös käyttää täyttä Chromea tai Chromiumia. Puppeteer eroaa Seleniumista siten, että se keskittyy pelkästään Chromeen sekä Chromiumin testaukseen sekä Javascriptiin kun taas Selenium tukee eri selaimia sekä useita eri ohjelmointi kieliä. Puppeteeria ei myöskään pidetä selaimen automaatiotyökaluna toisinkuin Seleniumia sekä Cypressia. Se on enemmänkin Chromium-selaimen sisäisten ominaisuuksien hallintatyökalu, jonka avulla voidaan käyttää päätöntä selainta useampien tehtävien suorittamiseksi kuten pyyntöjen ja vastausten käsittelyyn sekä elementtien paikantamiseen. (Lambdatest, 2023; Shashko, 2022)

5 TOTEUTUS

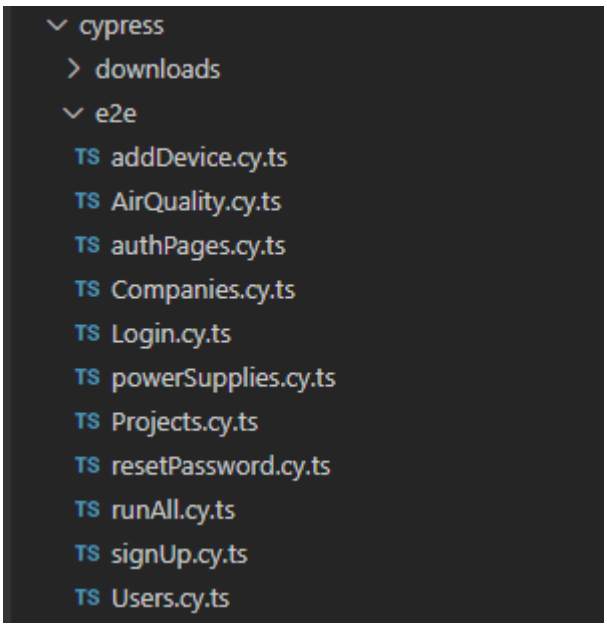
5.1 Suunnittelu

Opinnäytetyön suunnittelu alkoi eri testaustyökalujen etsimisellä sekä niihin tutustumisella. Työkaluihin tutustuessa kirjattiin ylös hyviä sekä huonoja puoli kyseisistä työkaluista ja niitä vertailtiin toisiinsa. Vertailun kohteena oli jo aikaisemminkin mainitut työkalut eli Selenium, Cypress sekä Puppeteer. Kun koettiin, että eri työkaluihin oli tutustuttu tarpeeksi sekä käytiin vielä työn tilaajan kanssa muistiinpanot läpi sekä keskusteltiin siitä, että mitä testaustyökalua käytämme projektissa. Päädyimme käyttämään Cypress nimistä testaustyökalua. Päädyimme tähän tulokseen siitä syystä, että olemme pyrkineet käyttämään sovelluksessa mahdollisimman uusia sekä nykyaikaisia työkaluja sitä mukaa kun olemme sitä kehittäneet. Cypress tuntuu olevan vaihtoehdoista uudenaikaisin sekä myös helppokäyttöinen runsaiden dokumentaatioiden vuoksi. Cypressilla tehdyt testit myös osoittautuivat nopeammiksi kuin esimerkiksi Seleniumilla tehdyt testit. Tämä katsottiin myös suureksi eduksi jo suunnitteluvaiheessa.

Suunnitteluun kuului myös testauksen arkkitehtuurin hahmottelu. Päädyttiin siihen tulokseen, että testit luodaan ensin käyttäjälle, kenellä on pääsy kaikkiin toimintoihin sovelluksessa, jotta pystymme testaamaan mahdollisimman kattavasti sovellusta. Osa sovelluksen toiminnoista on piilotettu käyttäjän roolin taakse, joten jos halutaan testata sovellusta mahdollisimman kattavasti, on järkevää testata sovellusta alkuun roolilla, miltä löytyy kaikki valtuudet sovellukseen. Muiden roolien testaaminen voidaan tehdä tämän jälkeen aikataulun puitteissa. Testaus etenee yksinkertaisesti samassa järjestyksessä kuin sivuvalikossa olevat sivut. Testaus loppuu kun, kaikki sivut ja niiden toiminnot on käyty läpi. Viimeisenä vaiheena suunnittelussa oli tarkempi perehtyminen valitun työkalun dokumentaatioon ja yleiseen toimivuuteen.

5.2 Testien työstäminen

Työskentely aloitettiin asentamalla Cypress osaksi projektia. Projektissa on käytössä node.js, joten Cypress saatiin asennettua helposti ”nmp install cypress --save-dev” -komennolla. Testejä varten tietokantaan luotiin uusi käyttäjä, mille on annettu valtuudet tehdä mitä vain web-sovelluksessa sekä määriteltiin baseUrl Cypressin config-tiedostoon testien navigoinnin helpottamiseksi. Projektin skripteihin myös lisättiin komento cy:open, jolla Cypressin oma käyttöliittymä voidaan avata kätevästi ”npm run cy:open” -komennolla. Cypress luo projektiin kansion nimeltään cypress jossa tulee mukana esimerkki testejä, jotka voidaan poistaa. Cypress kansiosta löytyy myös kansio nimeltään e2e, joka toimii tässä työssä tehtyjen testien tallennuspaikkana.



Kuva 1. Cypressin luoma kansio sekä testitiedostot

Web-sovelluksessa on kaikki sisältö kirjautumisen takana, joten loogisesti myös jokainen testi alkaa sovellukseen kirjautumisella. Cypress hakee kirjautumissivun tekstikentät niille määritellyn nimen perusteella ja syöttää niihin testikäyttäjän sähköpostin sekä salasanan. Kirjautumisen jälkeen sovellus ohjaa käyttäjän etusivulle, jossa esitetään tervehdys käyttäjän etunimellä. Testissä tarkistetaan, että tämä tervehdys löytyy ja myös, että localStorageen on luotu kryptattu accessToken, joka sisältää käyttäjän tietoja.

```
cy.get('input[name=email]').type(email)
cy.get('input[name=password]').type(`${password}{enter}`)
cy.contains('Tervetuloa takaisin, Tester')
cy.wrap(localStorage).invoke('getItem', 'accessToken').should('exist')
```

Kuva 2. Kirjautuminen testissä

Testejä luodessa ongelmaksi muodostui, että Cypress ei löydä joitakin komponentteja, jotka ovat testaamisen kannalta tärkeitä. Nämä komponentit olivat kuvalla varustettuja painikkeita, joilla ei ollut erikseen määriteltyä nimeä ja ne eivät sisältäneet mitään tekstiä, jonka avulla Cypress voisi niitä hakea. Näille komponenteille lisättiin data-cy attribuutti, joka ratkaisi kyseisen ongelman. Tätä attribuuttia voidaan hyödyntää vain testauksessa ja se ei vaikuta komponenttien toiminnallisuuteen tai ulkonäköön millään lailla.

```
<Delete
  data-cy="delete"
  style={{ cursor: 'pointer' }}
  onClick={() => handleDelete(row.id)}
/>
```

Kuva 3. Esimerkki komponentista, jossa on käytetty data-cy attribuuttia

Kirjautumisen jälkeen Cypress navigoi itsensä testattavalle sivulle, joka sille on määritelty koodissa. Lähes kaikki sivut testattavassa web-sovelluksessa sisältää lomakkeen. Lomakkeessa on näkyvillä yritykseen liittyviä tietoja, joita käyttäjä voi editoida sekä poistaa, jos käyttäjä on näihin toimintoihin

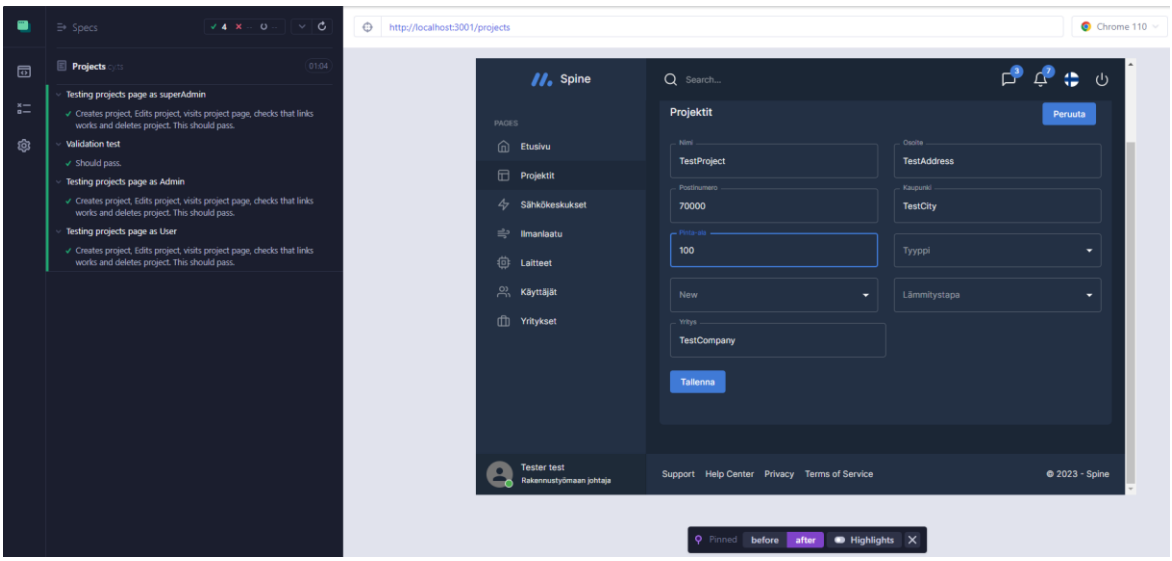
valtuutettu. Käyttäjällä on myös mahdollista lisätä uutta tietoa lomakkeen avulla. Kun testi on suorittanut jonkin edellä mainituista toiminnoista, se myös tarkistaa, että kyseinen toiminto on suoritettu onnistuneesti.

```
//ADD Project
cy.contains('Projektit').click()
cy.contains('Lisää').click()
cy.get('input[name=name]').type('TestProject')
cy.get('input[name=address]').type('TestAddress')
cy.get('input[name=zip]').type('70000')
cy.get('input[name=city]').type('TestCity')
cy.get('input[name=area]').type('100')
cy.get('[name=type]').parent().click()
cy.contains('Kerrostalo').click()
cy.get('[name=new]').parent().click()
cy.contains('Saneerauskohde').click()
cy.get('[name=warmingMethod]').parent().click()
cy.contains('Kaukolämpö').click()
cy.contains('Tallenna').click()
cy.contains('TestProject').should('exist')
//EDIT Project
cy.contains('TestProject').parents('tr').find('[data-cy=edit]').click()
cy.get('input[name=name]').clear()
cy.get('input[name=name]').type('Edited')
cy.contains('Tallenna').click()
cy.contains('Edited').should('exist')
//DELETE Project
cy.contains('Edited').parents('tr').find('[data-cy=delete]').click()
cy.contains('Edited').should('not.exist')
```

Kuva 4. Projektit sivun lomakkeen testaaminen

Kun Cypressin käyttöliittymä avataan testien ajoa varten, kysyy se ensimmäisenä, suoritetaanko E2E-testejä vai komponenttitestejä ja millä selaimella testit halutaan ajaa. Testaustyypin sekä selaimen valinnan jälkeen aukeaa näkymä, jossa on kaikki luodut testitiedostot. Käyttöliittymässä myös näkyy, milloin viimeksi testitiedosto on saanut muutoksia versionhallintaan. Testitiedostoa klikkaamalla aloittaa Cypress testin suorittamisen.

Testien suorittamista voidaan Cypressin käyttöliittymän avulla seurata reaaliajassa. Cypress ottaa testin jokaisesta vaiheesta tilannekuvia. joita voidaan myöhemmin käyttää apuna esimerkiksi debuggaamisessa.



Kuva 5. Projektit sivun testien suorittaminen Cypress käyttöliittymän näkökulmasta

Kuvan 5 vasemmassa reunassa näkyy, että Projects.cy.ts testitiedosto sisältää neljä eri testiä. Kolme näistä testeistä on lähes identtisiä siitä syystä, että sivua vain testataan eri rooleilla. Sovellus sisältää erilaisia toimintoja, joita ohjataan käyttäjän roolin perusteella, joten sovelluksen toimivuuden kannalta on tärkeää testata kaikki sovelluksen sivut kaikilla mahdollisilla rooleilla. Neljännessä testissä tarkistetaan, että lomake ei hyväksy tyhjiä kenttiä ja näin tallenna virheellistä tietoa.

5.3 Kattavuus

Työn edetessä nousi esille testien kattavuuden seuranta. Testien kattavuuden seuraaminen on tärkeä osa testausprosessia. Testejä luodessa herää usein kysymys, että onko jo testattu kaikkea vai tarvitseeko testejä tehdä lisää. Nopean etsinnän jälkeen Cypressin omista dokumentaatioista löytyi ohjeet kattavuuden seurantaan.

Kattavuuden seurantaa varten asennettiin @cypress/code-coverage paketti. Tämän jälkeen se lisättiin cypress/support/index.js tiedostoon tuontilauseella sekä rekisteröitiin tehtävä Cypressin config-tiedostoon. Kattavuuden seurantaa varten, lähdekoodi tuli myös instrumentoida. Tapa, miten Instrumentointi toteutetaan, vaihtelee riippuen siitä, että millä testattava sovellus on tehty. Tässä projektissa käytössä on React, joten instrumentointia varten täytyi asentaa @cypress/instrument-cra moduuli. Tämän lisäksi luotiin uusi npm-skripti ”start-dev”, joka vaatii kyseisen moduulin ennen palvelimen käynnistämistä.

```
import { defineConfig } from 'cypress'

export default defineConfig({
  e2e: {
    setupNodeEvents(on, config) {
      // implement node event listeners here

      require('@cypress/code-coverage/task')(on, config)
      return config
    },
    baseUrl: 'http://localhost:3001'
  }
})
```

Kuva 6. Cypressin config-tiedosto

Instrumentoidun palvelimen käynnistämisen jälkeen ajettiin yksi testitiedosto läpi. Tässä kohtaa voitiin todeta, että kattavuuden seuranta toimii ja projektiin ilmestyi ”coverage” -niminen kansio, jonka sisälle oli ilmestynyt HTML-tiedosto jokaisesta projektiin kuuluvasta tiedostosta. Tiedostoja läpi käydessä kuitenkin huomattiin, että niissä ei näkynyt kattavuus kuin sen testitiedoston osalta, joka oli juuri ajettu. Kattavuutta kuitenkin haluttiin katselmoida koko projektin näkökulmasta eikä vain yhden sivun ja tätä varten luotiin ”runAll” -niminen testitiedosto, joka ottaa vastaan kaikki muut testitiedostot muutamaa poikkeusta lukuun ottamatta.

```
> e2e > TS runAll.cy.ts
import './Companies.cy'
import './addDevice.cy'
import './Login.cy'
import './Users.cy'
import './Projects.cy'
import './powerSupplies.cy'
import './authPages.cy'
import './resetPassword.cy'
```

Kuva 7. "runAll" testitiedoston tuontilauseet

Kaikkien testien ajamiseen ”runAll”-tiedoston kautta menee hieman vajaa kuusi minuuttia, mutta kattavuutta voidaan nyt tarkastella koko projektin laajuisesti. Kattavuuden seuranta osoittautui hyödylliseksi apuvälineeksi testien kehitysvaiheessa.

Kuvassa 8 on kuvankaappaus "cypress/code-coverage" luomasta src/pages kansiosta. Jokaisella kansiossa sijaitsevalla tiedostolla on oma rivinsä, mistä voidaan tarkastella mitä kaikkea testi on käynyt läpi. Rivillä on esitettynä edistymispalkki, joka kertoo, kuinka kattavasti tiedosto on kokonaisuudessaan testattu. Riveiltä löytyy myös eriteltynä esimerkiksi funktiot osio, mikä kertoo, kuinka kattavasti tiedoston funktiot on testattu. Tiedoston nimeä klikkaamalla päästään tarkastelemaan kattavuutta koodirivi kerrallaan. Tällä voidaan varmistua siitä, että testit todella käyvät koko sovelluksen kaikki toiminnot läpi.

File ▲		Statements		Branches		Functions		Lines	
Companies.tsx		93.33%	56/60	97.36%	37/38	100%	19/19	93.22%	55/59
Company.tsx		100%	16/16	100%	4/4	100%	8/8	100%	16/16
CurrencyTable.tsx		100%	28/28	100%	8/8	100%	9/9	100%	28/28
Device.tsx		81.63%	40/49	65.62%	21/32	71.42%	10/14	81.63%	40/49
DeviceDetailsTable.tsx		77.77%	7/9	0%	0/2	60%	3/5	75%	6/8
DeviceLocationTable.tsx		58.33%	7/12	0%	0/2	37.5%	3/8	54.54%	6/11
DeviceSensordata.tsx		100%	6/6	100%	2/2	100%	3/3	100%	6/6
Devices.tsx		81.35%	48/59	66.66%	20/30	94.73%	18/19	81.03%	47/58
PowerSupplies.tsx		89.39%	59/66	91.66%	55/60	100%	22/22	89.23%	58/65
PowerSupply.tsx		66.66%	12/18	100%	6/6	40%	4/10	66.66%	12/18
Profile.tsx		100%	36/36	100%	4/4	100%	17/17	100%	33/33
Project.tsx		100%	15/15	100%	8/8	100%	7/7	100%	14/14
Projects.tsx		93.44%	57/61	98.14%	53/54	100%	19/19	93.33%	56/60
Users.tsx		94.25%	82/87	95.45%	63/66	100%	31/31	94.11%	80/85
addDevice.tsx		91.3%	42/46	92.3%	24/26	100%	13/13	91.11%	41/45

Kuva 8. Kuvankaappaus "cypress/code-coverage" luomasta src/pages kansiosta

6 JATKOKEHITYS

Tiesin jo opinnäytetyötä aloittaessa, että minun tulee miettiä työhön liittyvää jatkokehitystä. Testejä tulee ylläpitää sitä mukaa, kuin itse testattavaa sovellusta päivitetään. Jos testejä ei päivitetä sovelluksen mukana, testit menevät rikki ja niistä ei ole enää mitään hyötyä kenellekään. Testit tulisi siis päivittää ajan tasalle heti, kun huomaa, että testi menee rikki sovelluksen päivittämisen yhteydessä. Tämä on nopein ja helpoin ratkaisu, koska tekijällä on tuoreessa muistissa mitä päivityksiä hän on sovellukseen tehnyt ja näin hän tietää heti mitä päivityksiä testit vaativat. Näin rikkoutuneita testejä ei myöskään pakkaudu päällekkäin.

Tällä hetkellä testit joudutaan käynnistämään käsin koodieditorin kautta, mutta suunnitelmissa on saada testit osaksi GitLab-versionhallinnan CI/CD-putkia, jolloin testit ajettaisiin automaattisesti joka kerta kun versionhallintaan syötetään uutta koodia tai vanhaan koodiin tehdään muutoksia. Tämä vaatii kuitenkin tarkkaa perehtymistä aiheeseen sekä dokumentaatioon, jonka takia tämä jäi jatkokehityksen puolelle.

Tarkoituksena olisi myös saada backend mukaan kattavuuden seurantaa. Tällä hetkellä pystymme tiedostamaan, että backend toimii niin kauan kuin data liikkuu testejä ajaessa, mutta jos saisimme backendin koodit mukaan kattavuuden seurantaan auttaisi se meitä hahmottamaan, mitä kaikkea sieltä kutsutaan ja jääkö jotain testaamatta. Tästä olisi varmasti hyötyä tulevaisuudessa.

7 YHTEENVETO

Opinnäytetyön tavoitteena oli suunnitella sekä luoda mahdollisimman kattava automatisoitu web-sovelluksen testaaminen päästä päähän -menetelmällä, josta on mahdollisimman paljon hyötyä ohjelmiston laadunhallinnan kannalta. Opinnäytetyön aihe oli tarpeellinen, koska olen työskennellyt projektin parissa entuudestaan ja huomannut, että välillä näitä virheitä pääsee tuotantoversioon pakostakin ja se ei koskaan ole hyvä asia kenenkään kannalta. Opinnäytetyö oli myös aiheena itselleni mielekäs, koska olen aikaisemmin saanut vain pienen pintaraapaisun automatisoidusta testaamisesta, joten tämä tuli itselleni lähes uutena asiana ja koen uusien asioiden oppimisen tärkeänä itseni kehittämisen kannalta. Opinnäytetyöstä saamallani osaamisella ohjelmistonkehittäjänä on minulle hyötyä tulevaisuudessa.

Vaikka automatisoitu testaaminen sekä siinä käytettävät työkalut tulivat minulle uutena asiana, pystyin silti toteuttamaan opinnäytetyön aikataulussa. Tämän mahdollisti se, että panostin opinnäytetyön alkuvaiheessa suunnitteluun hieman enemmän kuin yleensä. Opin siis opinnäytetyöstä suunnittelun tärkeyden ohjelmistokehityksessä ja pyrin kiinnittämään huomiota tähän jatkossa. Minulla oli myös automatisoitua testausta kehittäessä etuna se, että olen työskennellyt testattavan sovelluksen parissa jo pidemmän aikaa. Perehdyin myös testauksessa käytetyn työkalun omaan dokumentaatioon ennen työn aloittamista enemmän kuin yleensä, mikä teki itse työskentelystä sujuvampaa.

LÄHTEET

BrowserStack. What is Selenium? Haettu 9.2.2023 osoitteesta https://www.browserstack.com/selenium

Selenium 2022. Selenium Overview. Haettu 9.2.2023 osoitteesta https://www.selenium.dev/documentation/overview/

Prabhpreet Kaur Bhamra 2023. What is Selenium? Introduction to Selenium Automation Testing. Haettu 9.2.2023 osoitteesta https://intellipaat.com/blog/tutorial/selenium-tutorial/introduction/

Chrome Developers 2018. Overview of Puppeteer. Haettu 9.2.2023 osoitteesta https://developer.chrome.com/docs/puppeteer/overview/

Lambdatest 2023. Puppeteer Tutorial: Complete Guide to Puppeteer Testing. Haettu 10.2.2023 osoitteesta https://www.lambdatest.com/puppeteer

Daniel Shashko 2022. Puppeteer vs Selenium: Main Differences. Haettu 10.2.2023 osoitteesta https://brightdata.com/blog/proxy-101/puppeteer-vs-selenium

Jash Unadkat 2022. Cypress vs Selenium: Key Differences. Haettu 10.2.2023 osoitteesta https://www.browserstack.com/guide/cypress-vs-selenium

Cypress 2023. Why Cypress? Haettu 10.2.2023 osoitteesta https://docs.cypress.io/guides/overview/why-cypress#Our-mission

Cypress 2023. Key Differences. Haettu 10.2.2023 osoitteesta https://docs.cypress.io/guides/overview/key-differences#What-you-ll-learn

Thomas Hamilton 2022. What is Software Testing? Haettu 13.2.2023 osoitteesta https://www.guru99.com/software-testing-introduction-importance.html

IBM. What is software testing? Haettu 13.2.2023 osoitteesta https://www.ibm.com/topics/software-testing

Jacob Schmitt 2022 What is end-to-end testing? Haettu 14.2.2023 osoitteesta https://circleci.com/blog/what-is-end-to-end-testing/