Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

SAMI RUOHONIEMI

Microsoft Playwright JS

Automatisoitu kuittien tallennus

SÄHKÖ- JA AUTOMAATIOTEKNIIKAN
KOULUTUSOHJELMA
2021

Tekijä Ruohoniemi, Sami	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Huhtikuu,2022
	Sivumäärä 21 + 4	Julkaisun kieli Suomi

Julkaisun nimi
Microsoft Playwright JS Automatisoitu kuittien tallennus

Tutkinto-ohjelma
Sähkö- ja automaatiotekniikka

Tiivistelmä

Tämän opinnäytetyön tarkoituksena oli suunnitella ja luoda automatisoitu kuittitietojen haku Airbnb-sivustolta. Tavoitteena oli toteuttaa automatisoitu ratkaisu niin, että ohjelma hakee käskystä sivulta uusimmat kuittitiedot. Työn toisessa luvussa vertailtiin kahta automatisointi järjestelmää ja käytiin läpi, miksi näistä työkaluista valittiin juuri Playwright. Kolmannessa luvussa lukijalle selvitetään mikä on Airbnb ja alustetaan ohjelman toimivuutta ja käydään läpi mitä tietoja sivulta halutaan saada. Luvussa neljä kerrottiin itse Playwright työkalun asentamisesta, ohjelman suunnittelusta sekä toteutuksesta.

Airbnb-sivujen automatisointiin käytettiin Microsoft Playwright JS ohjelmointikieltä, joka on Javascript-pohjainen automaatiotyökalu, jolla voidaan automatisoida lähes kaikenlaisia sivustoja.

Avainsanat
IT-automaatio, Microsoft Playwright JS, Web-automaatiotyökalu

Author Ruohoniemi, Sami	Type of Publication Bachelor's thesis	Date April 2022
	Number of pages 21 + 4	Language of publication Finnish

Title of publication
Microsoft Playwright JS Automated receipt storage

Degree program
Electrical and Automation Engineering

Abstract

The purpose of this thesis was to design and create an automated search for receipt information on the Airbnb website. The goal was to implement an automated solution so that the program retrieves the latest receipt information from the page. The second chapter of the work compared two automation systems and examined why Playwright was chosen from these tools. The third chapter explains to the reader what Airbnb is and introduces the functionality of the program and goes through what information you want to get from the site. Chapter four covered the installation of the Playwright tool itself, the design and implementation of the program.

The Microsoft Playwright JS programming language, a Javascript-based automation tool that can be used to automate almost any type of website, was used to automate Airbnb pages.

Keywords
IT automation, Microsoft Playwright JS, Web automation tool

SISÄLLYS

SYMBOLI- JA LYHENNELUETTELO

Framework	Runko, jolle voit rakentaa ohjelmistoa.
Funktio	Ohjelman toiminto, joka suorittaa tietynlaisen tehtävän.
JS	JavaScript on selainten ohjelmointi kieli, jolla tehdään sivuista interaktiivisia.
Käyttöliittymäelementti	Selaimessa oleva painike, taulukko, infolaatikko tai hakupalkki.
Node.js	Chromen JavaScript ajoaikaan rakennettu alusta, jolla voidaan rakentaa nopeita ja skaalautuvia verkkosovelluksia.
Visual Studio Code	Avoimen lähdekoodin koodieditori, jossa voidaan rakentaa ja muokata ohjelmaa.

1 JOHDANTO

Tämän opinnäytetyön tavoitteena on suunnitella ja luoda automatisoitu kuittitietojen haku Airbnb-sivustolta. Kuittitietojen haku onnistuu myös manuaalisesti, mutta se on paljon hitaampaa verrattuna automaattiseen ohjelmaan, joka hakee tiedot avaamatta selainta itsessään. Automaatiotyö tehdään sen takia, että kuittien käsittely ja tarkastelu olisi mahdollisimman nopeaa. Työstä on hyötyä Airbnb-sivustolla asuntoa vuokraavalle henkilölle, jonka tarvitsee käsitellä tai tarkastella kuitteja.

Airbnb on vuonna 2008 perustettu yhdysvaltalainen yritys, joka tarjoaa sovelluksen ja nettisivuston, jonka avulla voi vuokrata huoneiston tai asunnon ympäri maailmaa. Vuokrauksen jälkeen vuokralainen ja vuokranantaja voivat arvioida toisensa. Arvioiden perusteella henkilöt voivat valita oman majoituspaikkansa tai vuokralaisensa. (Rawes & Lacoma, 2021.)

Opinnäytetyössä on tarkoitus edetä seuraavanlaisella tavalla. Toisessa luvussa puhutaan hieman siitä, mikä tämä Microsoft Playwright JS oikein on. Miksi valitsin tehdä automatisoinnin juuri tällä työkalulla, miten se toimii ja mihin sovellusta voisi käyttää. Opinnäytetyön kolmannessa luvussa käydään läpi itse Airbnb-sivustoa. Tutkitaan miten ohjelman kuuluisi sivulla toimia sekä mitä tietoja ohjelmoijan kuuluu tietää itse sivusta ennen kuin ohjelma pystyy navigoida sivustolla. Työn neljännessä luvussa tarkastellaan, miten automaattinen tiedostojenhaku suunniteltiin. Minkälaisia ongelmia ohjelmoinnin aikana kohdattiin. Pystytiinkö kaikkia ongelmia ratkaisemaan vai jouduttiinko ongelma kiertää käyttäen jotakin toista lähestymistapaa.

Aloitin opinnäytetyöni kirjoittamisen joulukuussa 2021, vaikka itse automatisoidun ohjelman kirjoitin jo syksyn aikana ollessani työharjoittelussa. Teen tämän opinnäytetyön itselleni, eikä tässä ole toimeksiantajaa.

2 MICROSOFT PLAYWRIGHT JS

2.1 Mikä on playwright

Playwright on Microsoftin kehittämä ja ylläpitämä melko uusi avointa lähdekoodia käyttävä selainten automatisointijärjestelmä. Sillä voi suorittaa niin sanotun end-to-end testauksen, mikä tarkoittaa sitä, että sillä voi suorittaa kaikki testauksen vaiheet alusta loppuun. Playwrightilla voidaan automatisoida kaikkia nykyaikaisia selaimia ja se on suunniteltu frameworkiksi, joka ratkaisee nykypäivän verkkosovellusten testaustarpeet. (Shain, 2021.)

Playwright JS on nimensä mukaisesti Javascript pohjainen työkalu, josta JS lyhenne tulee. Playwright kuitenkin tukee myös muitakin kieliä kuten Typescriptiä, Pythonia, Javaa sekä .NET:tiä (Playwright, n.d.). Kuvassa 1 vielä esimerkki syntaksista (Kuva 1).

```
const playwright = require('playwright');


(async () => {

    var url = "http://localhost:8080/#/"
    var username = "sami.ruohoniemi"
    var password = "1234"
    var delay = 150

    const browser = await playwright["chromium"].launch({ headless:false });
    const page = await browser.newPage({
        viewport: {
            width: 1920,
            height: 1080,
        },
    });
    // Login
    await page.goto(url);
    await page.type( '[ name = username ]', username );
    await page.type( '[ name = password ]', password );
    await page.click('button:has-text("Kirjaudu")');

    // Logout
    await page.click( '[ name=logout ]', { delay:delay });
    await page.close();
})();
```

Kuva 1. Playwrightin syntaksi

2.2 Miksi playwright valittiin

Projektin alussa tehtävään vertailtiin kahta automatisointityökalua ja molemmilla tehtiin lyhyt automatisointitesti. Toinen näistä työkaluista oli Seleniumlibrary. Selenium on samalla tapaa avoimen lähdekoodin automaatiotestauspaketti, jota käytetään laajasti verkkosovellusten ja selainten automaatiotestaukseen. Sillä voidaan automatisoida selaimet ja se toimii vuorovaikutuksessa käyttöliittymäelementtien kanssa kopioidessa käyttäjän toimia testatakseen, toimiiko verkkosovellus odotetulla tavalla.

Playwrightilla sekä Seleniumilla on omat etunsa ja rajoituksensa. Mikä tarkoittaa, että valinta näiden välillä tapahtuu usein subjektiivisesti skenaarion mukaan. Playwright tarjoaa nopean testauksen monimutkaisissa verkkosovelluksissa päättömällä arkkitehtuurilla ja se vaatii edellytyksenä Node.js:n. Playwright on melko uusi ja siitä puuttuu tuki eri tasoilla, kuten yhteisössä, selaimissa, tietyissä laitteissa, kielivaihtoehdoissa ja integraatiossa. Selenium tarjoaa kaiken tämän. Playwright on ylivoimaisesti parempi monimutkaisissa verkkosovelluksissa, mutta kattavuus on rajallinen. Päinvastoin Selenium tarjoaa laajan kattavuuden, skaalautuvuuden ja joustavuuden sekä vahvan yhteisön tuen. (Tiwari, 2021.)

Kummatkin automatisointityökalut olivat yksinkertaisia asentaa ja niillä oli vaivatonta simuloida pientä testiympäristöä. Vastakkainasettelun ja muutamien pienien ongelmien jälkeen huomasin, miten Playwright soveltui paremmin tehtävään kuin Selenium. Kokemukseni ohjelmoinnista perustuu suurimmaksi osin JavaScriptistä eikä Seleniumin dokumentaatio siitä ollut kovin laaja. Seleniumilla ohjelmoija voi kirjoittaa ohjelmia eri kielillä, kuten Java, Ruby, Perl, C#, Python ja PHP. Näissä kielissä syntaksi eli kirjoitusasu on hieman erilainen. Seleniumin testiympäristössä on tällainen syntaksi (Kuva 2).

```
*** Settings ***
Library         SeleniumLibrary

*** Variables ***
${SERVER}       localhost:8080
${LOGIN URL}    http://${SERVER}
${BROWSER}      Chrome
${USER}         sami.ruohoniemi
${PASSWORD}     1234

*** Test Cases ***
Valid Login
    Open Browser To Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser

*** Keywords ***
Open Browser To Login Page
    Open Browser    ${LOGIN URL}    ${BROWSER}
    Title Should Be    Login Page

Input Username
    [Arguments]    ${USER}
    Input Text    username_field    ${USER}

Input Password
    [Arguments]    ${PASSWORD}
    Input Text    password_field    ${PASSWORD}

Submit Credentials
    Click Button    login_button

Welcome Page Should Be Open
    Title Should Be    Welcome Page
```

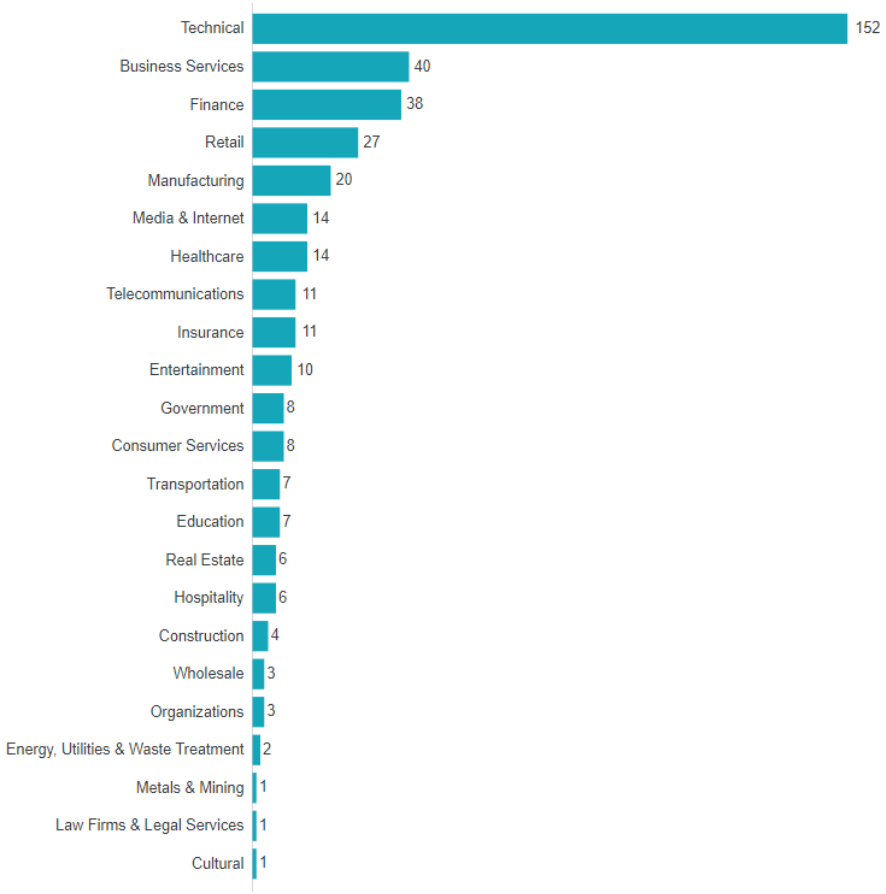
Kuva 2. Seleniumlibraryn syntaksi.

Loppujen lopuksi tein päätöksen käyttää Playwrightia, koska sen dokumentointi oli mielestäni parempi. JavaScript-pohjainen syntaksi oli helpompi omaksua, joten automaatiota oli yksinkertaisempi jatkaa Playwrightin testiautomaatiojärjestelmällä. Tässä tapauksessa dokumentaatio on kuin sanakirja, josta nähdään miten Playwrightia kirjoitetaan.

2.3 Playwrightin käytön jakautuminen eri aloille

Playwright JS on monipuolinen testiautomaatiojärjestelmä. Sitä pystytään käyttämään kaikenlaisiin selaimen automaatioprojekteihin, niin suurempiin kuin pienempiinkin ja alasta riippumatta. Ei ole siis väliä onko kyseessä Kelan sivuilla oleva dokumentin täydennys tai vaikka Zalandossa ostoksien lisääminen ostoskoriin. Playwrightin testiautomaatiojärjestelmää hyödyntävät tuhannet yritykset ympäri maailmaa. Yritykset jakautuvat monille aloille, kuitenkin eniten järjestelmää käyttäviä yrityksiä on teknisellä

alalla (Kuva 3). Suurin tunnettu yritys, joka käyttää Playwright JS on monelle nuorelle tuttu Snap inc. eli yritys, joka omistaa Snapchatin (HG Insights, n.d.-a).



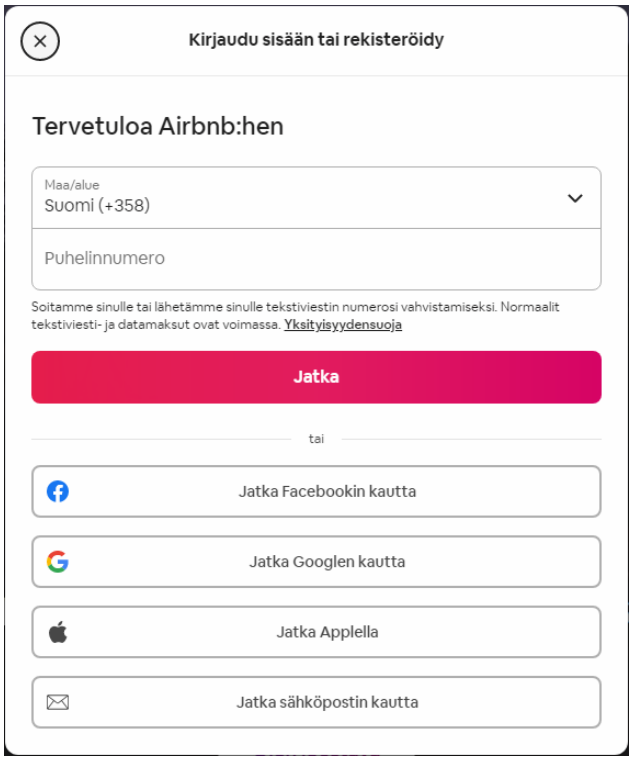
Kuva 3. Playwright JS käytön jakautuminen eri aloille. (HG Insights, n.d.-b)

3 AIRBNB

Airbnb toiminta alkoi vuonna 2007 kahden muotoilijan ylimääräisestä huoneesta, jota he vuokrasivat kolmelle matkustajalle, kun he etsivät paikkaa yöpyä. Vuosi tämän jälkeen kaverukset perustivat yrityksen nimeltään Air Bed & Breakfast, joka sittemmin lyhennettiin muotoon Airbnb vuonna 2009. Nykyään miljoonat ihmiset, jotka ovat luoneet käyttäjän Airbnb-sivulle voivat vuokrata yöpymissijaa matkustajille. (Airbnb, n.d.-a)

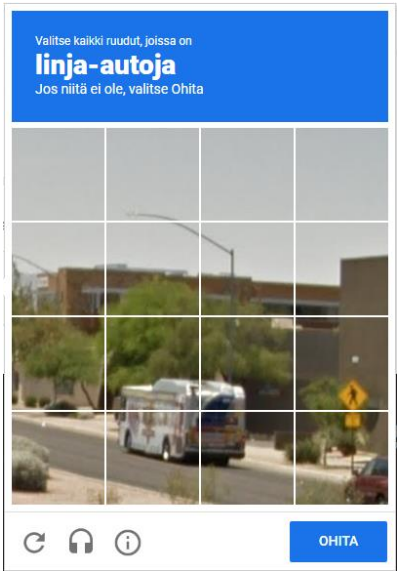
3.1 Sivun tarkastelu

Tarkasteltaessa Airbnb-sivustoa, voidaan huomata, että kirjautumisvaihtoehtoja on useita. Tällaisessa tilanteessa täytyy miettiä mikä näistä vaihtoehdoista olisi automaatiojärjestelmälle suotuisin (Kuva 4).



Kuva 4. Sivun kirjautumisvaihtoedot. (Airbnb, n.d.-b)

Tapauksessa ei ole esimerkiksi järkevää käsitellä tätä puhelimesta saatua kirjautumiskoodia, koska ohjelma halutaan pitää yksinkertaisena. Tässä tapauksessa päätin, että käytetään sähköposti kirjautumista, jolloin ohjelma pystyy täyttää kirjautumiskohdat automaattisesti. Ohjelmassa oli kuitenkin yksi ongelma, jota ei pysty mitenkään suorittamaan ja se oli reCAPTCHA sovellus (Kuva 5).

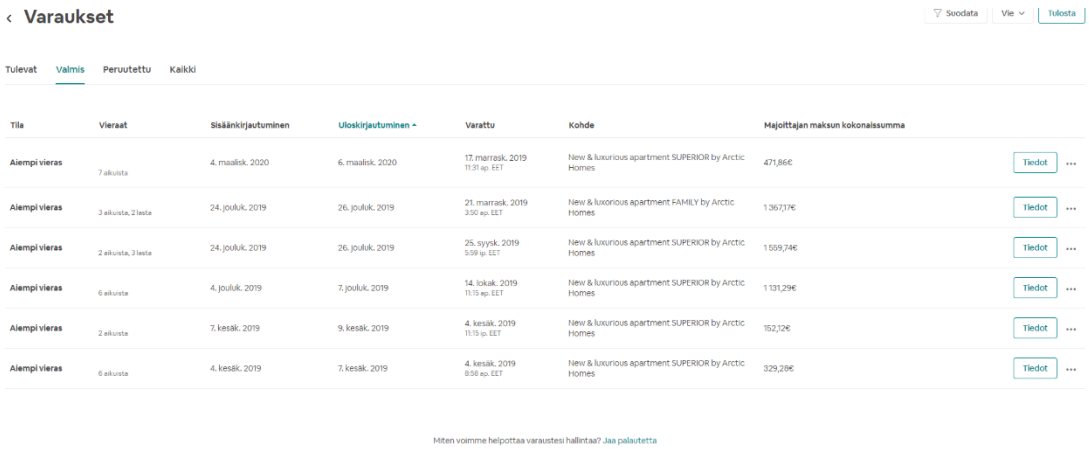


Kuva 5. Esimerkki reCAPTCHA testistä. (Airbnb, n.d.-c)

Tämä on Googlen ilmainen apuohjelma, joka auttaa suojaamaan verkkosivustoja roskapostilta ja väärinkäytöltä. ” CAPTCHA” on testi, joka erottaa ihmiset ja robotit toisistaan. Ongelman ratkaisuun palataan neljännessä luvussa. (Google support, n.d.)

3.2 Halutut kuittitiedot

Ohjelmaa tehtäessä voidaan antaa suora osoite kyseiselle sivulle, jossa ohjelman on tarkoitus operoida ja tässä tapauksessa osoite on varaukset välilehti. Varauksista nähdään kaikki viimeaikaiset varaukset ja kuitit kustakin varauksesta (Kuva 6). Kuvasta on poistettu vuokraajien nimet yksityisyyden suojaamiseksi. Tälle sivulle työkalu ohjataan suoraan operoimaan ohjelmaa.



Kuva 6. Varaukset, joista kuitit halutaan tallentaa. (Airbnb, Airbnb, n.d.-d)

Ohjelman tulee navigoida varaukset sivulle ja avata kyseiseltä sivulta aina viimeisimmän varauksen kuittitiedot, eli painamalla uusimman varauksen ”Tiedot”-nappia. Tämän jälkeen sovellus tallentaa kaikki mahdolliset kuitit haluttuun kansioon kuitin omalla tunnuksella, josta kuittia on mahdollista tarkastella.

Kuittien tallentamisen yhteydessä kohdattiin ongelma, jota Playwrightilla ei pystytty käsittelemään. Tämä ongelma johtui kuittien tallentamisesta. Kuvassa 6 oikeasta yläreunasta löytyy ”tulosta”-nappi, josta Airbnb-sivusto avaa erillisen pop-up ikkunan PDF-tiedoston tallentamista varten. Tätä ikkunaa ei ollut mahdollista käsitellä, joten ongelma kierrettiin ottamalla kuvakaappaus kuittisivusta.

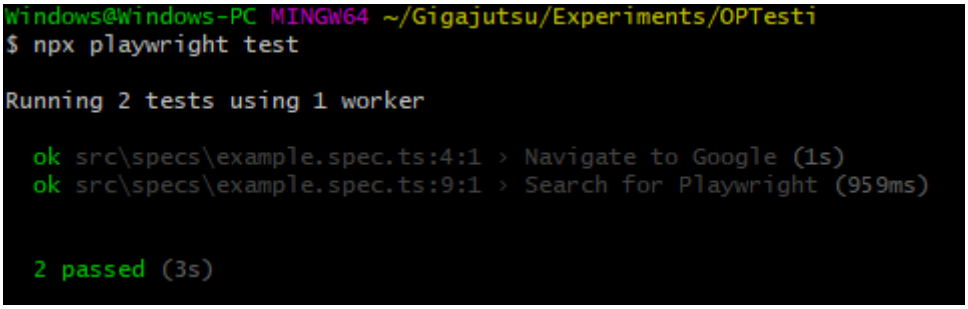
4 SUUNNITTELU JA TOTEUTUS

4.1 Asennus

Asennuksen alussa on hyvä luoda uusi kansio mihin ohjelma luodaan. Tässä työssä käytetään Visual Studio Code:a koodin kirjoittamiseen, jossa on jo valmiiksi ladattu node.js. Sitä tarvitaan itse Playwright työkalun asennukseen. Asennuksen apuna käytetään Playwrightin ”Getting-started”-sivuja. Tämän avulla luotiin testiympäristö missä pystyttiin ensin kokeilemaan itse työkalun mahdollisia toimintoja. Kuvasta voidaan tarkastella testiympäristön asennuksen vaiheita (Kuva 7).

Kuva 7. Playwrightin ja testiympäristön asennuskomennot ja versiot.

Testiympäristön asennuksen jälkeen voidaan suorittaa pienimuotoinen harjoitus. Testistä huomataan, että ympäristö toimii halutulla tavalla. Playwrightin hyvän dokumentoinnin avulla testiympäristö oli yksinkertainen suorittaa. ”Getting-started”-sivulla oli testi, jonka tarkoituksena on avata Googlen oletusselain ja hakea hakumoottorin avulla sanaa ”Playwright”. Testi suoritettiin komennolla ”npx playwright test”, josta huomatiin, että kaksi testiä suoritettiin ja molemmat läpäisivät testin. Aikaa komennon suorittamiseen kesti 3 sekuntia (Kuva 8).



Kuva 8. Testin suoritus ja tulokset.

On kuitenkin hyvä ottaa huomioon, että työn tarkoituksena ei ole testailla Airbnb-sivustoa vaan automatisoida se. Testin tuloksilla pyritään varmistamaan asennuksen suoriutuneen onnistuneesti, jonka jälkeen Playwright on valmis käytettäväksi. Automatisointi vaiheessa ohjelman tarkoituksena on löytää ja tallentaa oikeanlaiset tiedot. Ohjelmaa ajetaan erilaisella komennolla, joka on ”node tiedostonNimi.js”.

4.2 Suunnittelu ja toteutus

Suunnittelun tavoitteena on ymmärtää kokonaiskuva sovelluksesta sekä huomata kaikki mahdolliset ongelmat, joita ohjelman suoritus voi kohdata. Suunnittelun aikana on hyvä käydä manuaalisesti läpi kaikki vaiheet, jotka ohjelma suorittaa. Testisuunnitelman aikana on hyvä varmistaa, että sovellus suorittaa ohjelman turvallisesti niin onnistuessa kuin virhetilanteessakin. Sovellusta tehdessä on hyvä suorittaa testejä ”headless: false”-tilassa, joka helpottaa virheiden huomaamista. ”Headless: false”-tila tarkoittaa sitä, että ohjelma aukaisee selaimen näkyville samalla tapaa kuin normaali selain. Myöhemmässä vaiheessa, kun sovellus toimii ja virheilmoitukset pystytään käsittelemään onnistuneesti, voidaan sovellus suorittaa ”headless: true”- tilassa. Silloin sovellus ei enää avaa itse selainta vaan suorittaa ohjelman niin sanotusti taustalla.

Tarkastelun alkuvaiheessa huomattiin, että Airbnb-sivulle täytyy kirjautua. Näistä kirjautumisvaihtoehdoista paras oli käyttää kirjautumista sähköpostin avulla. Kirjautumisen jälkeen sivusto kuitenkin pyytää suorittamaan reCAPTCHA-testin, jonka suoritus ei tällä työkalulla ole automaattisesti mahdollista. Tässä tapauksessa paras vaihtoehto on käytännössä luoda toinen sovellus. Tämän sovelluksen tarkoituksena on ainoastaan kirjautua Airbnb-sivustolle ja odottaa käyttäjää suorittamaan Googlen apuohjelma manuaalisesti. Sen jälkeen ohjelma tallentaa sivuston evästeet. Evästeet ovat yleensä

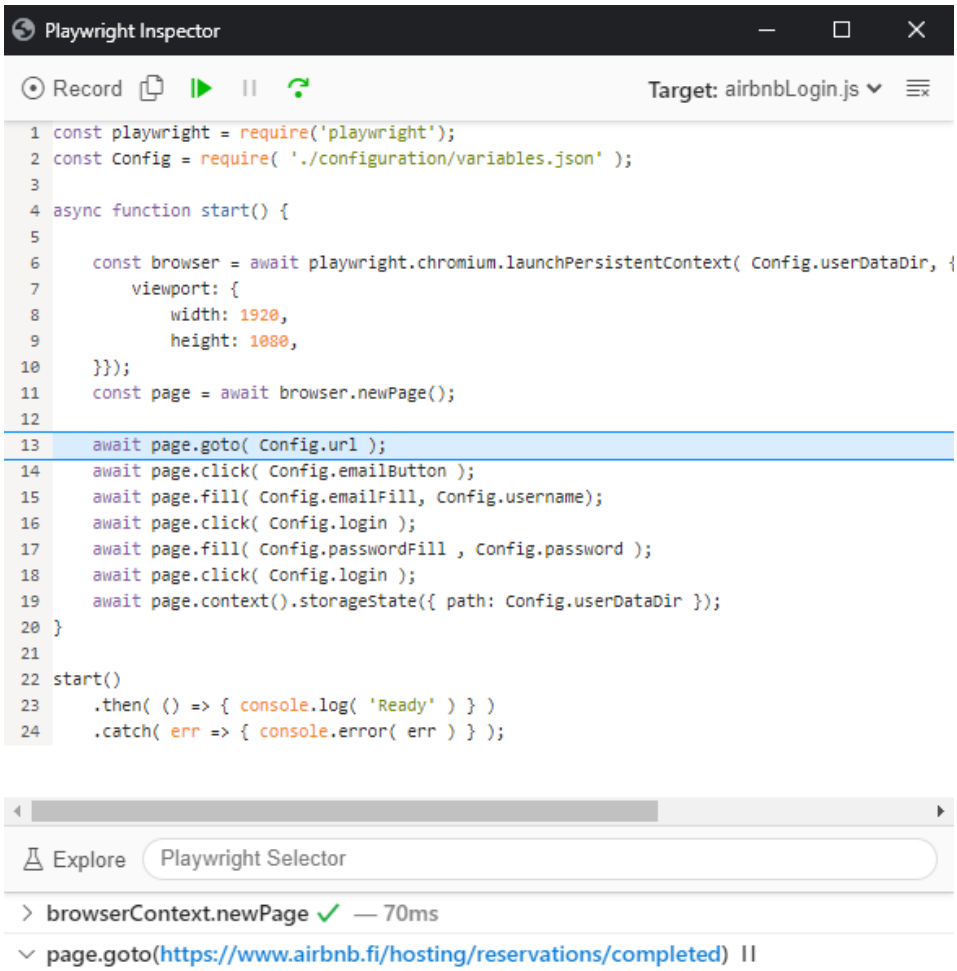
pieniä tekstitiedostoja, joissa on tunnustunnisteet, jotka tallennetaan tietokoneen selainhakemistoon tai ohjelmatietojen alikansioihin. Evästeet luodaan, kun käyttäjä vierailee jollakin verkkosivustolla, joka käyttää evästeitä seuratakseen käyttäjän liikkumista sivustolla, auttaakseen käyttäjää jatkamaan siitä mihin on jääty, muistamaan rekisteröityneen kirjautumisen, teeman valinnan, asetukset tai muita mukautustoimintoja. (all about Cookies n.d.)

Kirjautumissovellusta käytetään ensimmäisen kerran sovellusta käytettäessä, kun ohjelmaa käytetään eri tietokoneessa. Tämä sovellus toimii niin, että ohjelma käynnistetään ”headless: false”-tilassa ja kirjautumisen yhteydessä suoritetaan Google reCAPTCHA-apuohjelma manuaalisesti. Kirjautumisen jälkeen evästeet sisältävät tarvittavat tiedot kirjautumiseen, jolloin ne voidaan tallentaa ohjelman Config-tiedostoon. Config-tiedostoa käytetään myöhemmin avattaessa kuittienhaku sovellusta, jolloin ohjelma avaa selaimen Config-tiedostossa olevien evästeiden avulla ja kirjautumista ei enää vaadita (Kuva 9).

```
async function start() {
    const browser = await playwright.chromium.launchPersistentContext( Config.userDataDir,
        {
            headless: false,
            acceptDownloads: true,
            viewport: {
                width: 1920,
                height: 1080,
}});       You, 4 months ago • airbnb payments downloader
    const page = await browser.newPage();
```

Kuva 9. Sovellus käynnistää selaimen niillä evästeillä, jotka manuaalikirjautumisesta on saatu.

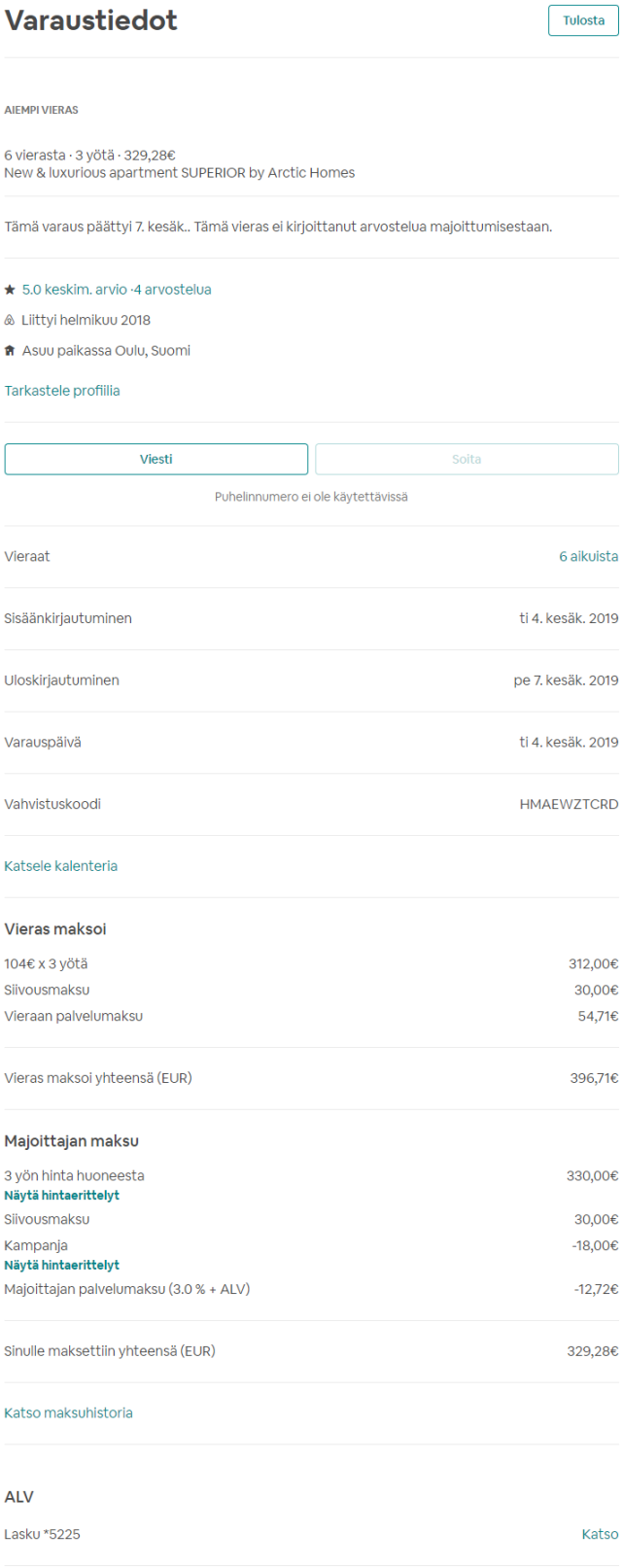
Playwrightilla voi navigoida selaimessa monella tavalla. Ohjelmaa kirjoittaessa virheiden mahdollisuus kasvaa ja kirjoittaminen on hidasta. Playwright kuitenkin tarjoaa työkalun, joka generoi koodia automaattisesti. Tämä työkalu on Playwrightin tarjoama debuggeri eli niin sanottu virheenjäljitin. Debuggeri-työkalun saa käyttöön komentokehotteesta komennolla ”set PWDEBUG = 1”. Ohjelmaa suorittaessa Playwright avaa debuggerin sovelluksen ensimmäiselle koodiriville ja odottaa käskyjä käyttäjältä. Debuggeri työkalun avulla voidaan edetä ohjelmaa rivi riviltä eteenpäin tai painaa playnappia. Tällöin ohjelma suoriutuu alusta loppuun näyttäen, milloin milläkin rivillä ohjelman suoritus on menossa (Kuva 10).



Kuva 10. Playwright debuggeri työkalu.

Kun debuggeri työkalun halutaan generoida koodia, aloitetaan silloin tallennus vasemman yläkulman Record-napista. Tällöin debuggeri luo komennon jokaisesta käyttäjän tekemästä painalluksesta selaimessa. Toisin sanoen käyttäjä suorittaa manuaalisesti sivulla tarvittavat toiminnot ja debuggeri tallentaa nämä käyttäjän painallukset valmiiksi koodiksi. Kuitenkaan kaikissa tapauksissa nämä tallennus mekanismin tunnisteet eivät aina ole erityisen tarkkoja ja komentoja joutuu muokkaamaan manuaalisesti. Tämä kuitenkin nopeuttaa koodin kirjoittamista huomattavasti.

Kuten kuvan (Kuva 10) alareunasta voidaan huomata, että ohjelmalle annetaan suora linkki oikealle sivulle. Tällöin ohjelman turhat komennot minimoidaan, eikä sovelluksen tarvitse navigoida sivulla erikseen. Linkki ohjaa sovelluksen sivulle, josta näkyy kaikki viimeaikaiset varaukset. Varaukset sivulta ohjelma avaa viimeisimmän kuitin ja ottaa kuitista sekä ALV-laskusta kuvakaappauksen (Kuva 11).



Varaustiedot

Tulosta

AIEMPI VIERAS

6 vierasta · 3 yötä · 329,28€
New & luxurious apartment SUPERIOR by Arctic Homes

Tämä varaus päättyi 7. kesäk.. Tämä vieras ei kirjoittanut arvostelua majoittumisestaan.

★ 5.0 keskim. arvio ·4 arvostelua

Liittyi helmikuu 2018

Asuu paikassa Oulu, Suomi

Tarkastele profiilia

Viesti | Soita

Puhelinnumero ei ole käytettävissä

Vieraat	6 aikuista
Sisäänkirjautuminen	ti 4. kesäk. 2019
Uloskirjautuminen	pe 7. kesäk. 2019
Varauspäivä	ti 4. kesäk. 2019
Vahvistuskoodi	HMAEWZTCRD

Katsele kalenteria

Vieras maksoi

104€ x 3 yötä	312,00€
Siivousmaksu	30,00€
Vieraan palvelumaksu	54,71€
Vieras maksoi yhteensä (EUR)	396,71€

Majoittajan maksu

3 yön hinta huoneesta	330,00€
Näytä hintaerittelyt	
Siivousmaksu	30,00€
Kampanja	-18,00€
Näytä hintaerittelyt	
Majoittajan palvelumaksu (3.0 % + ALV)	-12,72€
Sinulle maksettiin yhteensä (EUR)	329,28€

Katso maksuhistoria

ALV

Lasku *5225	Katso

Kuva 11. Kuvakaappaus varauksen kuitista. (Airbnb, n.d.-e)

Kuvaa on rajattu sopivammaksi lukijalle. Kuvakaappauksen aikana ohjelma suorittaa vielä kaksi funktiota. Funktio nimeltään ”setFolderPath” pilkkoo selaimen osoitteen

saadakseen kyseisen varauksen vahvistuskoodin ja luo uniinkin kansion varauksen kuvakaappauksia varten. Toinen näistä on ”getPaymentPathName”-funktio, joka nimeää kuvakaappaukset tällä uniikilla vahvistuskoodilla ja lisää ne vahvistuskoodia vastaavaan kansioon. Tämän jälkeen kuittitiedot on tallennettu, mutta ohjelman suoritus on kuitenkin vielä kesken, sillä sovelluksen on tarkoitus tiedottaa käyttäjää onnistuneesta suorituksesta.

Niin kuin luvun alussa mainitaan, on ohjelman tarkoitus suorittaa sovellus niin sanotusti taustalla, piilossa käyttäjältä. Ohjelman loppuun tarvitaan vielä yksi funktio. Kun ohjelma suoritetaan taustalla, ei käyttäjä tällöin tiedä onko ohjelma suoriutunut onnistuneesti vai onko suorittaessa tapahtunut jonkinlainen virhe. Ohjelmaa tehdessä käytettiin viestintään sovellusta Slack, johon ohjelman suorituksen tulos lähetettiin. Ohjelman lopussa ”sendNotification”-funktio lähettää Slack-sovellukseen ilmoituksen ohjelman suorituksen tuloksesta. Kun kaikki suunnittelun ja toteutuksen vaiheet kirjautumisesta, selaimessa navigoinnista, kuvakaappauksien tallentamisesta sekä käyttäjälle informoinnista on valmiina, voidaan todeta, että työ on valmis ja toimiva.

5 POHDINTA

Tämän opinnäytetyön päätavoitteena oli luoda automaattisesti toimiva kuittitietojen haku Airbnb-sivustolta. Työn yhteydessä vertailtiin kahta yleisesti käytettyä testiautomaatio työkalua. Työkalujen vertailun pohjana käytettiin itse työtä ja näiden testiautomaatiojärjestelmien soveltuvuutta kyseiseen tehtävään. Vertailun jälkeen käytiin pikaisesti läpi miksi Playwright JS valittiin testiautomaatiojärjestelmäksi tähän projektiin. Vertaillessani näitä järjestelmiä valinta oli minulle selkeä ja valinta näiden kahden testiautomaatiojärjestelmien välillä perustui puhtaasti ohjelmointi kokemuksiini, sekä mielestäni selkeämpään ja laajempaan dokumentaatioon.

Ennen kyseistä projektia minulla ei ollut minkäänlaista kokemusta Microsoft Playwright JS:n käytöstä ja oikeastaan koko testiautomaatiojärjestelmä oli minulle tuntematon. Kokonaisuudessaan projektia lähdettiin rakentamaan pienin askelein täysin puhtaalta pöydältä. Projektin alussa jouduin viettämään muutamia tunteja omassa testiympäristössäni, jotta ymmärsin minkälainen Playwrightin syntaksi on ja minkälaisia komentoja ohjelma tottelee. Alkukankeuden jälkeen ohjelman suunnittelu alkoikin olla jo valmis, kuitenkaan ongelmitta ei selvitty ja ongelmiin piti keksiä mielestäni aika luoviakin ratkaisuja.

Projekti alkoi sillä, että yksinkertaisuudessaan kirjauduttiin Airbnb-sivustolle ja tämä yksinkertaiselta kuulostava tehtävä olikin projektin yksi vaikeimmista vaiheista. Tällaisissa tapauksissa on otettava huomioon, että selaimet ja ylipäätään verkkosivut on tarkoitettu käytettäväksi ihmisille eikä niin sanotusti robotille. On hyväksyttävä, että varsinkin kirjautuessa näitä yritetään karsia pois. Tässä tapauksessa täytyikin käyttää kekseliästä ratkaisua, jotta koko kirjautumisvaihe pystyttiin ohittamaan. Kirjautumisen yhteydessä täytyi tehdä Googlen reCAPTCHA apuohjelma manuaalisesti

ensimmäisen kerran. Kirjautumisen yhteydessä käyttäjän evästeasetukset pystytään tallentamaan sovelluksen omaan kansioon. Tämän jälkeen ohjelmalla on tarvittavat tiedot avata sivu ilman kirjautumista. Evästeiden, eli käyttäjän tietojen tallentaminen toimii tavallaan kuin ”muista minut”-nappi. Muutoin projekti eteni mallikkaasti ja ohjelman lopputulos täytti kaikki kriteerit, joita ohjelmalle suunnitteluvaiheessa asetettiin. Sovellus hakee uusimmat kuittitiedot viimeisimmästä varauksesta, tallentaa kuitit haluttuun kansioon ja ilmoittaa käyttäjälle onko ohjelma suoriutunut tehtävästä onnistuneesti.

Playwrightilla työskentely on ollut antoisaa ja mielenkiintoista. Projekti eteni mielestäni tasaisesti ja oli aikataulussa. Lopputuloksena sovellus tekee oikeat asiat Airbnb-sivustolla ja tallentaa oikeat kuitit omiin kansioihin ja tiedottaa käyttäjää onnistuneesta suorituksesta. Luulen, että tulen tulevaisuudessa tekemään omia pieniä projekteja tällä testiautomaatiojärjestelmällä. Yksi näistä projekteista voisi olla esimerkiksi sähköpostini roskakorin ja roskapostin tyhjennys.

LÄHTEET

Airbnb. (n.d.). About us. Haettu 14.12.2021 osoitteesta https://news.airbnb.com/about-us/

All about Cookies. (n.d.). What is a cookie? Haettu 17.2.2022 osoitteesta https://www.allaboutcookies.org/cookies/

Google support. (n.d.). What is reCAPTCHA? Haettu 3.1.2022 osoitteesta https://support.google.com/recaptcha/answer/6080904?hl=en

HG Insights. (n.d.). Playwright. Haettu 15.2.2022 osoitteesta https://discovery.hgdata.com/product/playwright

Playwright. (n.d.). Supported languages. Haettu 29.3.2022 osoitteesta https://playwright.dev/docs/languages

Shain, D. (13.5.2021). Your 80 Questions about Playwright, Answered. https://applitools.com/blog/top-playwright-questions-answered/

Tiwari, G. (26.3.2021). Playwright vs Selenium: A Comparison. https://www.browserstack.com/guide/playwright-vs-selenium

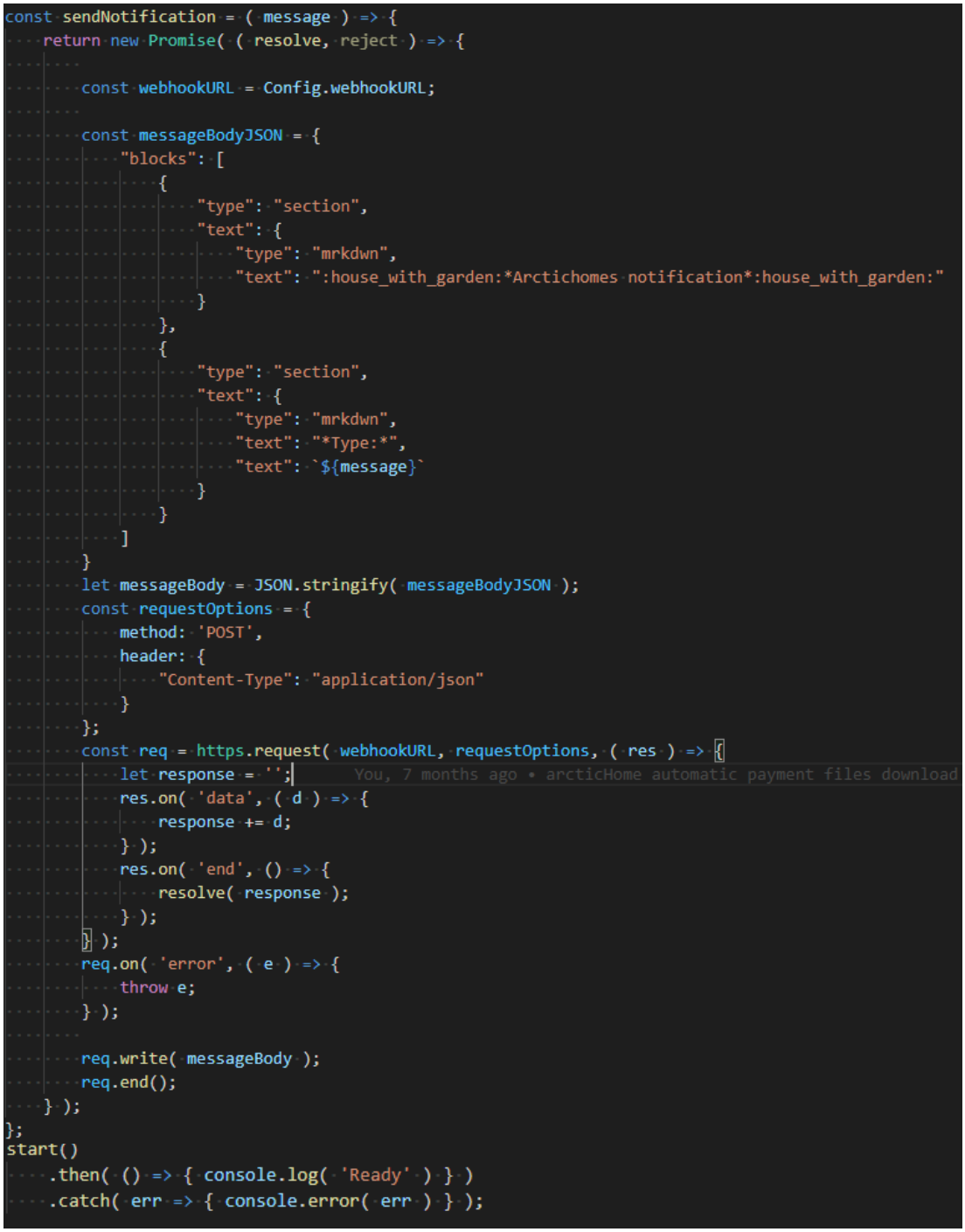
Wawes, E. & Lacoma, T. (19.12.2021). What is Airbnb? What to know before becoming a guest or host. https://www.digitaltrends.com/home/what-is-airbnb/

```
const playwright = require( 'playwright' );
const https = require( 'https' );
const fs = require( 'fs' );
const Config = require( './configuration/variables.json' );

async function start() {
    const browser = await playwright.chromium.launchPersistentContext( Config.userDataDir,
        {
            headless: false,
            acceptDownloads: true,
            viewport: {
                width: 1920,
                height: 1080,       You, 8 minutes ago • Uncommitted changes
    }});
    const page = await browser.newPage();

    await page.goto( Config.url );
    const pageUrl = page.url();

    if ( pageUrl == Config.url ) {

        const [ page2 ] = await Promise.all([
            page.waitForEvent( 'popup' ),
            page.click( Config.customer )
        ]);

        let folder = setFolderPath( page2.url() );
        let name = getPaymentPathName( page2.url() );
        await page2.screenshot( { path: folder + name, fullPage: true } );

        if ( await page2.isVisible( Config.twentySix ) ) {
            const [ page3 ] = await Promise.all([
                page2.waitForEvent( 'popup' ),
                page2.click( Config.twentySix )
            ]);
            name = getPaymentPathName( page3.url() );
            await page3.screenshot( { path: folder + name, fullPage: true } );
        }
        if ( await page2.isVisible( Config.twentySeven ) ) {
            const [ page4 ] = await Promise.all([
                page2.waitForEvent( 'popup' ),
                page2.click( Config.twentySeven )
            ]);
            name = getPaymentPathName( page4.url() );
            await page4.screenshot( { path: folder + name, fullPage: true } );
        }
        if ( await page2.isVisible( Config.twentyEight ) ) {
            const [page5] = await Promise.all([
                page2.waitForEvent( 'popup' ),
                page2.click( Config.twentyEight )
            ]);
            name = getPaymentPathName( page5.url() );
            await page5.screenshot( { path: folder + name, fullPage: true } );
        }
        const success = Config.success;
        await sendNotification( success );
        await browser.close();

    } else {
        const alert = Config.alert;
        await sendNotification( alert );
        await browser.close();
    }
}
```

Sovelluksen pääohjelma.

```
const getPaymentPathName = ( payment ) =>{
    // take end of url
    // set it to file's name
    let paymentPath;
    for ( let index = 0; index < payment.length; index++ ) {
        payment = payment.split( '/' ).pop();
    }
    paymentPath = payment + '.png';
    return paymentPath;
}

const setFolderPath = ( payment ) =>{
    //--FOLDERPATH--
    const folderPath = './payments/';
    let paymentFolder;
    for ( let index = 0; index < payment.length; index++ ) {
        payment = payment.split( '/' ).pop();
    }
    paymentFolder = folderPath + payment + '/';
    return paymentFolder;
}
```

Funktiot, jotka nimeää kuvakaappaukset.



```
const sendNotification = ( message ) => {
    return new Promise( ( resolve, reject ) => {

        const webhookURL = Config.webhookURL;

        const messageBodyJSON = {
            "blocks": [
                {
                    "type": "section",
                    "text": {
                        "type": "mrkdwn",
                        "text": ":house_with_garden:*Arctichomes notification*:house_with_garden:"
                    }
                },
                {
                    "type": "section",
                    "text": {
                        "type": "mrkdwn",
                        "text": "*Type:*",
                        "text": `${message}`
                    }
                }
            ]
        }
        let messageBody = JSON.stringify( messageBodyJSON );
        const requestOptions = {
            method: 'POST',
            header: {
                "Content-Type": "application/json"
            }
        };
        const req = https.request( webhookURL, requestOptions, ( res ) => {
            let response = '';      You, 7 months ago • arcticHome automatic payment files download
            res.on( 'data', ( d ) => {
                response += d;
            } );
            res.on( 'end', () => {
                resolve( response );
            } );
        } );
        req.on( 'error', ( e ) => {
            throw e;
        } );

        req.write( messageBody );
        req.end();
    } );
};
start()
    .then( () => { console.log( 'Ready' ) } )
    .catch( err => { console.error( err ) } );
```

Funktio, joka lähettää ilmoituksen suorituksesta.

```
const playwright = require('playwright');
const Config = require( './configuration/variables.json' );

async function start() {

    const browser = await playwright.chromium.launchPersistentContext( Config.userDataDir, {
        headless: false,
        acceptDownloads: true,      You, 8 seconds ago • Uncommitted changes
        viewport: {
            width: 1920,
            height: 1080,
    }});
    const page = await browser.newPage();

    await page.goto( Config.url );
    await page.click( Config.emailButton );
    await page.fill( Config.emailFill, Config.username);
    await page.click( Config.login );
    await page.fill( Config.passwordFill , Config.password );
    await page.click( Config.login );
    await page.context().storageState({ path: Config.userDataDir });
}

start()
    .then( () => { console.log( 'Ready' ) } )
    .catch( err => { console.error( err ) } );
```

Kirjautumissovellus, jolla tallennetaan evästeet.