LAB University of Applied Sciences

RPA-työnkulun kehittäminen

LAB-ammattikorkeakoulu
Insinööri (AMK)
2022
Tommi Rantanen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Rantanen, Tommi	Opinnäytetyö, AMK	2022
	Sivumäärä 32	

Työn nimi

RPA-työnkulun kehittäminen

Tutkinto ja koulutusala

Insinööri (AMK), tieto- ja viestintätekniikan koulutus

Toimeksiantajan nimi, titteli ja organisaatio

Anssi Pekkarinen, CTO, QAutomate Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli luoda sovellus, joka muuntaa robottikoodia vuokaavioksi tai vuokaavioita robottikoodiksi. Toisena tavoitteena tutkittiin RPA-teknologiaa eli Robotic Process Automationia ja sen hyötyjä ja tulevaisuutta, sekä siihen liittyviä teknologioita. Sovelluksen tavoitteena on auttaa RPA-kehittäjiä luomaan nopeammin ja helpommin robottipohjia vuokaavioiden perusteella, sekä luoda prosessikaavioita koodista.

RPA-teknologiaa käytetään automatisoimaan toistettavia ja yksinkertaisia tehtäviä. RPA:n vahvuudet, joihin kuuluu ajan säästäminen, virheiden minimalisointi ja rajapintojen kiertäminen tekevät siitä viehättävän vaihtoehdon eri tehtävien automatisoimiselle. RPA:n tulevaisuus vaikuttaa lupaavalta, sillä sille on luvattu suurta kasvua ja kehitystä, erityisesti hyperautomaation kehittyessä.

Kaavioiden käsittely toteutettiin yEdin avulla, joka on ilmainen diagrammien luomiseen suunniteltu ohjelma. Koodaus toteutettiin Pythonilla ja Robot Frameworkilla, joka on automaatiorakenne, sekä käyttämällä ulkoisia kirjastoja kuten Playwrightia, jolla automatisoidaan selainta luomalla selaimesta kopio, johon lähetetään komentoja.

Opinnäyteyössä luotiin sovellus, joka toimii proof of conceptina. Sovellusta voidaan käyttää nopeammin ja helpommin luomaan robottipohjia tai vuokaavioita RPA-kehittäjille. Sovellus osaa myös automatisoida koodin luomista erilaisten avainsanojen avulla.

Asiasanat

Vuokaavio, RPA, Python, Robot Framework, Playwright

Abstract

Author(s)	Type of Publication	Published
Rantanen, Tommi	Thesis, UAS	2022
	Number of Pages 32	

Title of Publication

RPA workflow development

Degree and field of study

Bachelor of Engineering, Information Technology

Name, title and organisation of the client

Anssi Pekkarinen, CTO, QAutomate Oy

Abstract

The goal of the thesis was to create a software program, which converts flowcharts to executable robot code, or robot code to flowcharts. The second goal was to study RPA (Robotic Process Automation) and its benefits and future, as well as technologies related to RPA. The purpose of the program is to help create robot templates based on process flowcharts faster and easier for RPA developers.

RPA is used to automate repeatable and simple tasks. RPA's strengths, such as saving time, minimizing errors and circumventing the need for APIs makes it an alluring option for automating different tasks. RPA's future looks bright, as it is predicted to grow greatly in the coming years, especially as hyperautomation develops.

The flowcharts are created with yEd, a free software program to create diagrams. Programming was implemented with Python and Robot Framework, an automation framework for creating robots. Playwright, a library used to automate browsers by creating an instance, was used as well.

A software program was created during the thesis, which acts as a proof of concept. The program can be used to create robot structures or flowcharts easier and faster for RPA developers. The program can also automate the creation of code by using different keywords.

Keywords

Flowchart, RPA, Python, Robot Framework, Playwright

Sisällys

1 Johdanto ... 1
2 Kaavioiden käsittely ... 3
2.1 Esittely ... 3
2.2 Käyttötarkoitus ... 3
2.3 Kaavioiden käsittely ... 3
3 Robotic process automation ... 6
3.1 Esittely ... 6
3.2 Optical character recognition ... 6
3.3 Vahvuudet ... 7
3.4 Heikkoudet ... 8
3.5 Perinteisen ja RPA-mallien yhdistäminen ... 9
3.6 RPA:n käyttötapaukset ... 9
3.7 RPA:n tulevaisuus ... 9
4 Robot Framework ... 11
4.1 Esittely ... 11
4.2 Käyttäminen ... 11
4.3 Playwright ... 13
5 Koodin muuntaminen vuokaavioksi ... 14
5.1 Tavoitteet ... 14
5.2 Koodin lukeminen ... 14
5.3 Poikkeuksien käsittely ... 14
5.4 GraphML-tiedoston luominen ... 14
6 Vuokaavion muuntaminen koodiksi ... 17
6.1 Tavoitteet ... 17
6.2 Vuokaavion lukeminen ... 18
6.3 Erityismerkkien korvaaminen ja muut poikkeukset ... 18
6.4 Haarautuminen ... 19
6.5 Python-tiedoston kirjoittaminen ... 21
6.6 Robot-tiedoston kirjoittaminen ... 22
6.7 Selainmalli ... 22
7 Caset ... 25
7.1 Case: PDF-tiedostojen parsiminen ja tiedon tallentaminen JSON-tiedostoihin ... 25
7.2 Case: Robottikoodin muuntaminen vuokaavioksi ... 28
8 Yhteenveto ja pohdinta ... 29

Lähteet 31

Käsitteet ja lyhenteet

API/rajapinta	API (Application Programming Interface) tai rajapinta tarkoittaa mekanismia, jolla kaksi eri ohjelmistoa keskustelevat keskenään.
Edge	Kaaviossa oleva nuoli, joka yhdistää noodit toisiinsa.
GraphML	yEdin käyttämä tiedostotyyppi, joka on XML:n kaltainen.
GUI	GUI (Graphical User Interface) eli graafinen ruudulla oleva käyttöliittymä.
Lokaattori	Lokaattori on elementin uniikki tapa erottautua muista elementeistä.
Noodi	Kaaviossa oleva muotoelementti, joka sisältää tekstiä.
OCR	OCR (Optical Character Recognition) on teknologia, jolla tunnistetaan tekstiä kuvasta.
Playwright	Node.js-kirjasto, jolla automatisoidaan eri selaimia.
Python	Korkean tason ohjelmointikieli.
QAutoCapture	QAutomaten luoma ohjelma, jolla luodaan prosessikaavioita ja suunnitelmia.
RPA	RPA (Robotic Process Automation) on automaatioteknologia, jossa ohjelma kopioi ja jäljittää käyttäjän tallentamat vuorovaikutukset GUI:n kanssa.
Taski	Taski (eng. task) on Robot Frameworkissa funktiota muistuttava ominaisuus.

XML	Merkintäkieli ja tiedostotyyppi, jolla säilytetään tietoa ihmisluettavalla tavalla.
yEd	yWorksin luoma ohjelma, jolla luodaan kaavioita.

1 Johdanto

Maailmaa automatisoidaan jatkuvasti enemmän ja enemmän. Robotit sekä todellisessa että kybermaailmassa helpottavat ihmisten elämää ja liiketoimintaa tekemällä töitä nopeasti ja luotettavasti. Ohjelmistojen automatisoimisella sekä isot yritykset että yksittäiset henkilöt voivat helpottaa työtänsä, vähentäen tai kokonaan poistaen tarpeen tehdä yksinkertaisia ja haastetta vailla olevia tehtäviä. RPA (Robotic Process Automation) on kehitetty nimenomaan tätä varten. RPA tarkoittaa automaatioteknologiaa, jossa käyttäjä luo robotin, joka kopioi ja toistaa käyttäjän näppäimien painallukset (Gartner). RPA:n kasvaessa tarvitaan uusia työkaluja ja standardeja, jotka auttavat RPA:ta kasvamaan jatkossakin merkittäväksi osaksi tulevaisuuden teknologiamaailmaa.

Opinnäytetyön toimeksiantajana toimii QAutomate Oy, vuonna 2015 perustettu suomalainen yhtiö, joka tuottaa RPA-teknologioita eli ohjelmistorobotiikan ja automaation alustaratkaisuja. QAutomaten kotikaupunki on Helsinki, mutta sillä on toimisto myös Tampereella. QAutomate toimii yhteistyössä Q-Factory Oy:n kanssa, joka on suomalainen tietokoneohjelmien testaamiseen ja laadun takaamiseen keskittyvä yhtiö. Opinnäytetyön kirjoittaja työskenteli QAutomatelle opinnäytetyön kirjoittamisen aikana, ja yritys tarjosi tarvittavat resurssit opinnäytetyön toteutusta varten. QAutomaten tuotteisiin kuuluu muun muassa QAutoCapture-sovellus, jolla luodaan RPA-prosesseja ja QAutoFlow, joka on Chrome-laajennus, jolla kopioidaan käyttäjän toiminnot Chromessa ja automatisoidaan Chromea luomalla suoritettavaa koodia. Nämä esimerkit ovat suorassa yhteydessä opinnäytetyössä tuotettuun ohjelmaan. Opinnäytetyössä käsitellään vain niitä tuotteita tarkemmin, jotka ovat opinnäytetyön kannalta tärkeitä.

Opinnäytetyön tavoitteena on luoda sovellus, joka muuntaa vuokaavion koodipohjaksi RPA-robotille, tai muuntaa robottikoodin kuvaavaksi vuokaavioksi. Opinnäytetyössä myös käsitellään työkaluja ja menetelmiä, joita käytetään sovelluksen luomisessa. Sovellusta käytetään muuntamaan suuri määrä koodia vuokaavioiksi tai vuokaavioita koodiksi. Opinnäytetyössä myös tutkitaan RPA-teknologiaa, sen käyttämistä ohjelman luomisen yhteydessä ja sen hyötyjä ja tulevaisuuden ennustuksia. Sovelluksen luomista varten opinnäytetyössä tutkitaan useita teknologioita. Sovellus luodaan suurimmaksi osaksi Python-ohjelmointikielellä ja käyttämällä useita Python-kirjastoja. GraphML-tiedostojen käsittelyä varten tarvitaan tapa käsitellä XML-muotoista dataa, jonka vuoksi opinnäytetyössä tutkitaan yEdiä, kaavioiden käsittelyyn suunniteltua ohjelmaa. Opinnäytetyössä tutkitaan myös Robot Framework -rakennetta, jota käytetään robottikoodin ajamisessa. Sovellusta varten käytetään myös Exceliä tallentamaan tietoa, jonka mukaan sovellus käsittelee poikkeustapauksia. Muita tarvittavia teknologioita, kuten kaavioiden tuottamiseen tarkoitettuja oh-

jelmistoja, opinnäytetyössä ei tarkemmin tutkita kuin niitä, joita siinä käytetään. RPA:n tutkimista varten tutkitaan myös joitakin siihen liittyviä teknologioita, kuten OCR-teknologiaa (Optical Character Recognition) eli tekstintunnistusta.

Sovelluksen tarkoituksena on auttaa RPA-kehittäjiä luomaan nopeasti ja helposti robotteja luomalla robottipohjan vuokaavioiden perusteella. Tyypillisesti asiakas luo vuokaavion joko QAutoCapture-sovelluksen avulla tai suoraan yEd-sovelluksella. Vuokaavion tarkoituksena on auttaa selventämään asiakkaan tarpeita robotille, jonka jälkeen robottipohjaa voidaan kehittää eteenpäin. Sovellusta käytetään tilanteissa, joissa yksinkertainen RPA-ratkaisu ei riitä, vaan robottiin tarvitaan monimutkaisempaa koodia. Robotti voidaan myös tarvittaessa muuntaa takaisin vuokaavioksi muokkauksien jälkeen.

Ohjelman käyttöä ja sen hyötyjä esitellään omassa kappaleessaan, jossa esitellään mahdollinen työprojekti, joka saadaan alkuun ohjelman avulla. Hyötyihin kuuluvat muun muassa ajan säästäminen, virheiden minimalisointi ja suhteellinen taloudellinen hyöty, kuin jos ohjelmaa ei olisi käytetty.

Sovellukseen syötetään GraphML-tiedostoja, joita luodaan yEd-sovelluksella. Muita tiedostotyyppejä ei voida sovelluksella muuntaa. Sovelluksella on myös rajoituksia koodin monimutkaisuudessa. Sovellus ei välttämättä myöskään tule olemaan käyttöönottovalmis, vaan se voi olla keskeneräinen. Sovellus toimii siis proof of conceptina.

2 Kaavioiden käsittely

2.1 Esittely

yEd on yWorksin luoma ohjelma, jolla luodaan ja muokataan kaavioita. Kaavioita luodaan lisäämällä elementtejä tyhjälle pohjalle, ja elementeille lisätään ominaisuuksia kuten tekstiä ja yhteyksiä toisiin elementteihin. yEd tukee myös muita ominaisuuksia, kuten automaattista tekstin sijoittamista, värien vaihtamista kaikille elementeille ja kuvien käyttämistä elementeissä. Toinen tapa luoda kaavioita on tuoda dataa esimerkiksi Excel- tai XML-tiedostoista. Valmis kaavio voidaan tallentaa eri muodoissa, kuten GraphML, PNG, PDF, JPG ja SVG. yEdistä on myös olemassa selaimessa käytettävä versio, yEd Live. yEd on saatavilla Windowsilla, Linuxilla ja macOS:illa. (yWorks.)

2.2 Käyttötarkoitus

Opinnäytetyössä yEdiä käytetään vuokaavioiden käsittelyyn opinnäytetyössä luodun sovelluksen yhteydessä. Opinnäytetyössä käytetään yEdiä, koska QAutomate käyttää yEdiä vuokaavioiden tuottamiseen, sekä sen helppouden ja ominaisuuksien vuoksi. yEd tukee useita formaatteja, jonka ansiosta kaavio voidaan tallentaa joko GraphML-tiedostona, tai se voidaan muuntaa kuvaksi, joka voi olla hyödyllistä suunnitteluvaiheessa. yEd tukee myös useita komentoja, joilla kaavion luomisprosessia helpotetaan.

yEd on myös asiakkaan kannalta helppo käyttää, erityisesti yEd Liven ansiosta, joka ei vaadi latausta. yEdin käyttöliittymä on yksinkertainen ja sitä on helppo ymmärtää. Asiakkaan kannalta yEd on siis hyödyllinen ohjelma visualisoimaan toivottua robotin rakennetta.

2.3 Kaavioiden käsittely

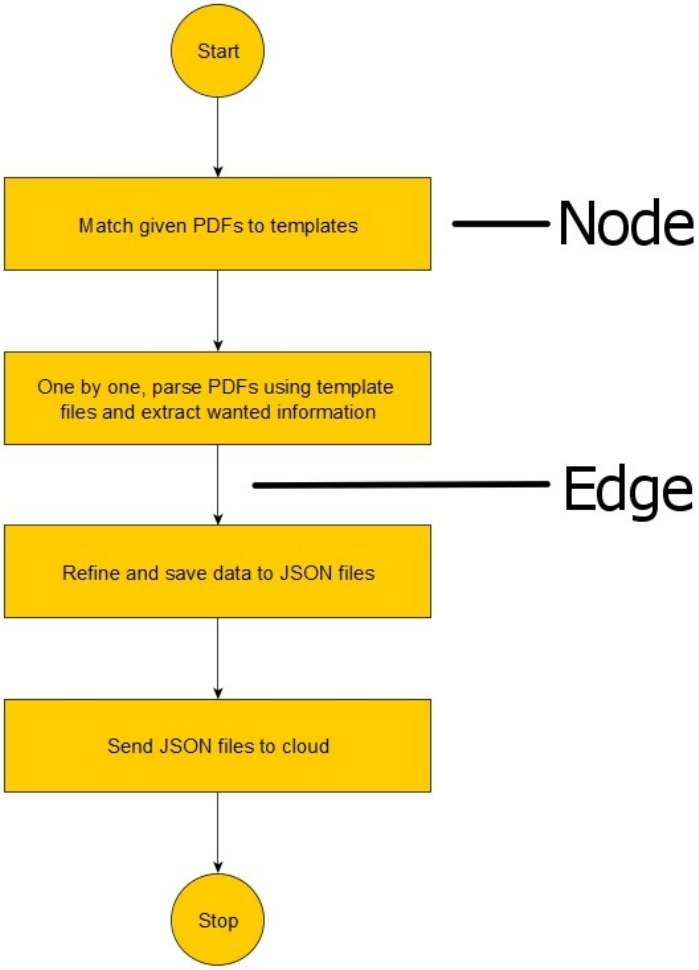
yEdin oletustiedostotyyppi on GraphML, joka on XML:ään perustuva. XML on merkintäkieli ja tiedostotyyppi, jolla säilytetään tietoa ihmisluettavalla tavalla. Kuvassa 1 nähdään kaikki GraphML-tiedoston tallentama tieto kaavioista, kuten elementtien sijainnit, tekstit ja värit. GraphML-tiedostoja voidaan lukea ja käsitellä Pythonin avulla.

```
<node id="n4">
  <data key="d6">
    <y:ShapeNode>
      <y:Geometry height="92.08000000000015" width="424.4000000000001" x="5
      <y:Fill color="#FFCC00" transparent="false"/>
      <y:BorderStyle color="#000000" raised="false" type="line" width="1.0",
      <y:NodeLabel alignment="center" autoSizePolicy="content" fontFamily="I
      <y:Shape type="rectangle"/>
    </y:ShapeNode>
  </data>
</node>
<node id="n5">
  <data key="d6">
    <y:ShapeNode>
      <y:Geometry height="92.08000000000015" width="92.08000000000015" x="7
      <y:Fill color="#FFCC00" transparent="false"/>
      <y:BorderStyle color="#000000" raised="false" type="line" width="1.0",
      <y:NodeLabel alignment="center" autoSizePolicy="content" fontFamily="I
      <y:Shape type="ellipse"/>
    </y:ShapeNode>
  </data>
</node>
<edge id="e0" source="n0" target="n1">
  <data key="d10">
    <y:PolyLineEdge>
      <y:Path sx="0.0" sy="0.0" tx="0.0" ty="0.0"/>
      <y:LineStyle color="#000000" type="line" width="1.0"/>
      <y:Arrows source="none" target="standard"/>
      <y:BendStyle smoothed="false"/>
    </y:PolyLineEdge>
  </data>
</edge>
<edge id="e1" source="n1" target="n2">
  <data key="d10">
    <y:PolyLineEdge>
      <y:Path sx="0.0" sy="0.0" tx="0.0" ty="0.0"/>
      <y:LineStyle color="#000000" type="line" width="1.0"/>
      <y:Arrows source="none" target="standard"/>
      <y:BendStyle smoothed="false"/>
    </y:PolyLineEdge>
  </data>
</edge>
```

Kuva 1. GraphML-tiedoston sisältö

Kaavioiden rakenne koostuu useista eri elementeistä. GraphML-tiedostossa on aluksi muutamia elementtejä, jotka kertovat tiedostolle, mitä ominaisuuksia se tarvitsee. Seuraavaksi alkaa näkyvä osuus: tiedostossa aluksi määritellään noodit, jotka ovat kaavion tietolaatikot. Noodeille määritellään tietoa kuten niiden muoto, koko, väri ja tekstin asetukset. Noodien jälkeen määritellään ”edge-elementit”, eli edget, noodien väliset nuolet, jotka yhdistävät noodit toisiinsa. Kuvassa 2 tarkennetaan noodit ja edget, ja kuvan 1 sisältö vastaa kuvan 2 rakennetta. Edgeille määritellään lähde- ja päätenoodit, eli mikä noodi johtaa

mihin noodiin. Edgen teksti, eli joko ”kyllä” tai ”ei” tarvitaan, kun noodi voi johtaa useaan eri noodiin. Tässä tapauksessa noodi esittää kyllä/ei-kysymyksen, ja edge saa sen mukaan tekstin, mihin noodiin se johtaa. Edgeille myös määritellään pisteet, eli kääntökohdat. Pisteitä käytetään, kun edge ei ole suora viiva, vaan siinä esiintyy käännekohtia. Pisteet pitävät sisällään niiden koordinaatit vuokaaviossa, eli missä kohtaa käännekohta esiintyy.



Kuva 2. Vuokaavion rakenne

3 Robotic process automation

3.1 Esittely

RPA eli Robotic Process Automation tarkoittaa automaatioteknologiaa, jossa käyttäjä luo robotin, joka kopioi ja toistaa käyttäjän toimenpiteet. Perinteisessä ohjelmistoautomaatiossa kehittäjä itse ohjelmoi listan askeleita, joita robotti suorittaa. RPA-mallissa ohjelma tallentaa ja kopioi käyttäjän vuorovaikutukset GUI:n (Graphical User Interface) eli graafisen käyttöliittymän kanssa. Robotti voi sitten jäljittää ja toistaa tallennetut askeleet. Askeleita tallentaessa ohjelma etsii tai luo uniikin lokaattorin elementeille, joiden avulla robotti osaa toistaa askeleet. Lokaattorilla tarkoitetaan tapaa erotella elementit toisistaan, ja yhdellä elementillä voi olla useita eri lokaattoreita.

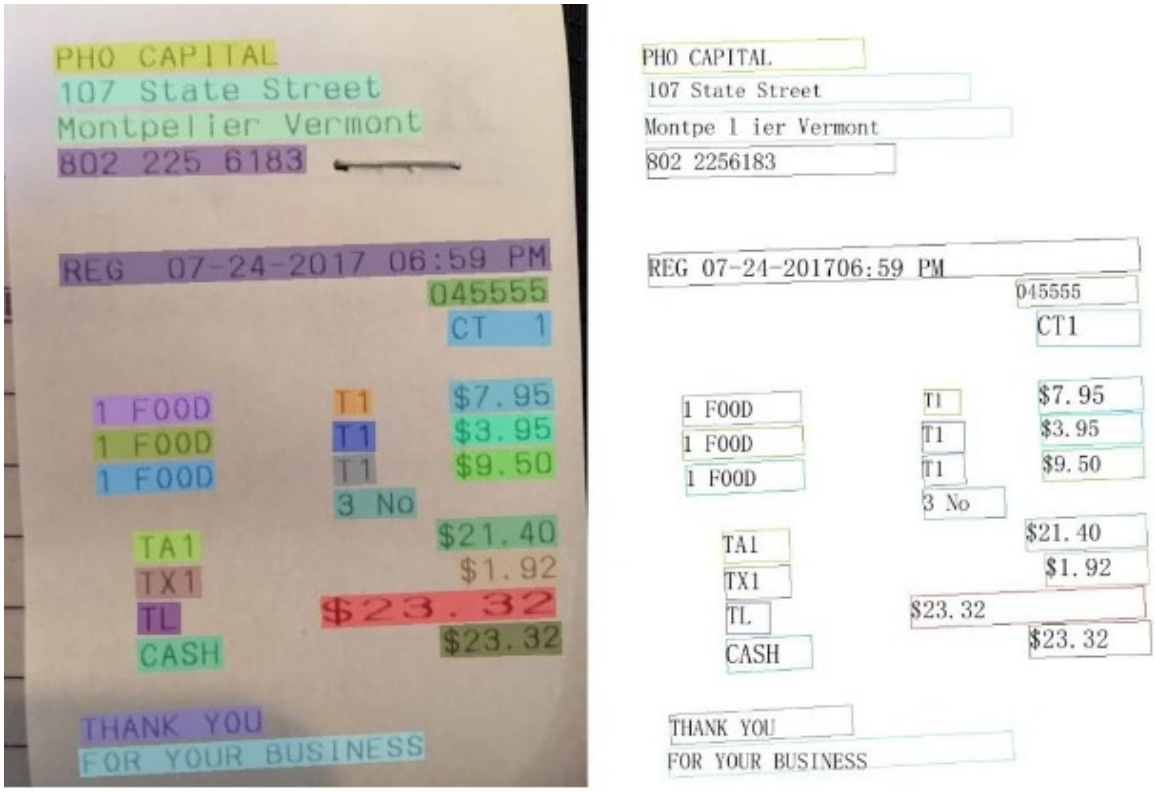
RPA voidaan jakaa kahteen eri ihmisen osallistumisen tarpeen mukaiseen kategoriaan: valvomattomaan ja valvottuun automaatioon. Valvomattomalla automaatiolla tarkoitetaan tapaa käyttää RPA-robottia, jossa ihmisen ei tarvitse erikseen tehdä mitään, jotta robotti käynnistyy ja jatkaa tehtäviään haluttuun pisteeseen asti. Valvomattomassa mallissa robotti on tyypillisesti sille omistetulla palvelimella. Valvottu automaatio taas tarkoittaa tapaa, jossa robotin tehtävä on sellainen, että se kerää ja valmistelee dataa ihmisen käytettäväksi. Tyypillisesti ihminen käynnistää robotin, ja robotti on paikallisella laitteella. (Automation Anywhere a.)

3.2 Optical character recognition

Optical Character Recognition (OCR) tarkoittaa teknologiaa, jolla tunnistetaan ja kerätään dataa skannatuista kuvista, kamerakuvista ja PDF-tiedostoista, jotka sisältävät kuvia (IBM 2022). OCR on osa ”screen scraping” -teknologioita, joilla tarkoitetaan tapaa kerätä dataa käyttäjän ruudulta. Suuri osa liiketoiminnan datasta on jäsentämättömässä muodossa, kuten käsin kirjoitetuissa ja tulostetuissa dokumenteissa, sähköposteissa, digitaalisissa kuvissa ja PDF-tiedostoissa. Näitä dokumentteja ei voida käsitellä paremmin, jos ne eivät ole jäsennetyssä formaatissa, kuten tekstitiedostoissa. (Automation Anywhere b.) OCR:llä kerätty tieto sen sijaan voidaan käsitellä pidemmälle.

OCR-prosessissa kuva aluksi skannataan ja kopioidaan. Kopioimisen jälkeen OCR-ohjelma muuttaa kuvan mustavalkoiseksi, jossa musta väri edustaa merkkejä. Valkoinen väri taas edustaa taustaa. Kuvassa 3 nähdään esimerkki, jossa vain teksti on mustaa ja kaikki muu valkoista. Mustat alueet käsitellään ja niistä etsitään merkit, joista luodaan sanoja ja kappaleita. Kuvassa 3 teksti on lajiteltu rivi kerrallaan, ja rivit on jaettu lohkoihin. Tekstintunnistus voi tapahtua kahden eri algoritmin avulla. Hahmontunnistuksessa OCR-

ohjelmalle syötetään tekstiä eri fonteissa ja formaateissa, jota verrataan skannattuun kuvaan. Ominaisuuksien havaitsemisessa (eng. feature detection) OCR-ohjelma käyttää sääntöjä merkkien löytämisessä. Esimerkiksi ison A:n ominaisuudet ovat kaksi diagonaalista viivaa, joiden välillä on yksi vaakasuora viiva. (IBM 2022.)



Kuva 3. OCR käytännössä (Boesch 2022)

OCR on tärkeä osa RPA:ta ja automaatioteknologioita. Yksi OCR:n suurimmista vahvuuksista on sen nopeus: OCR-sovellus voi kerätä tekstiä paljon nopeammin kuin ihminen voisi, mikä säästää paljon työtunteja. OCR on myös hyvin tarkka, tunnistaen oikein keskimäärin 98 % merkeistä (Automation Anywhere b). OCR on myös yhteensopiva lähes kaikkien sovelluksien kanssa, sillä se ei tarvitse pääsyä rajapintoihin.

3.3 Vahvuudet

RPA:n hyötyihin kuuluu muun muassa ajan säästäminen, koska kehittäjän ei tarvitse itse ohjelmoida askeleita, ainoastaan tallentaa ne kerran. Erityisesti selaimista elementtien hakeminen voi olla aikaa kuluttavaa työtä, koska elementit voivat olla sekavia, jos niiden koko polku otetaan. Elementeistä voi olla hankala hahmottaa lokaattori, joka ei aiheuta ristiriitaa muiden elementtien kanssa. Elementiltä voidaan kuitenkin tallentaa useita lokaattoreita sen jälkeen, kun käyttäjä on klikannut sitä tai on muuten vuorovaikutuksessa sen kanssa, ja lokaattoreista valitaan paras vaihtoehto. RPA-robotti voi myös tehdä töitä

vuorokauden ympäri, erityisesti jos on kyseessä valvomaton robotti. Robotti myös tekee työt paljon nopeammin kuin ihminen. Ajan säästäminen ja nopeus taas tarkoittavat säästöjä. (Automation Anywhere a.)

Virheiden minimalisointi kuuluu myös hyötyihin. Robotti aina toistaa samat askeleet samassa järjestyksessä ja estää esimerkiksi kirjoitusvirheitä. Kun ohjelma itse tallentaa askeleet, ei synny riskiä siitä, että robotin luoja joko valitsisi väärän elementin tai väärän toiminnan, kuten hiiren vasemman näppäimen oikean sijasta. (Automation Anywhere a.)

RPA poistaa tarvetta ihmisiltä tehdä toistettavia, yksitoikkoisia tehtäviä. Sen sijaan työntekijät voivat keskittyä tekemään haastavia ja olennaisia tehtäviä. (Automation Anywhere a.)

RPA voi tehdä tehtäviä ilman API-rajapintoja (Application Programming Interface). API tarkoittaa tapaa, jolla usea eri ohjelmisto voi kommunikoida keskenään. Joissakin ohjelmistoissa ei ole niitä, tai käyttäjällä ei ole niihin pääsyä. Esimerkiksi asiakas haluaisi automatisoida työpöytäsovellusta, jossa tarvitsisi painaa muutamaa nappulaa ladatakseen asiakirjan. Tämä tehtävä voidaan automatisoida esimerkiksi OCR-teknologian avulla. Kun asiakas nauhoittaa tehtävää, robotti voi päätellä usealla eri tavalla, kuten koordinaattien perusteella, mitä elementtiä painettiin. Elementin tiedot, kuten nappulan teksti, tallennetaan. Robotti sitten etsii ruudulta vastaavan elementin suorituksen aikana.

3.4 Heikkoudet

RPA:lla on kuitenkin myös heikkoutensa. Jos askeleet tallentaa kokematon henkilö, eikä kokenut ohjelmoija, hän saattaa vahingossa käyttää RPA-ohjelmistoa väärin. Esimerkiksi ”hover”-toiminta, joka avaa valikon, voi jäädä vahingossa tallentamatta. Robotti yrittää etsiä valikkoa tai klikata koordinaatteja, jossa valikon kuuluisi olla, mutta valikkoa ei vielä ole näkyvillä, joten robotti epäonnistuu.

Toisena heikkoutena on lokaattorit: jos elementin sijaintia, eli tarkkoja koordinaatteja ei voida käyttää aina ja varmuudella, elementille pitää löytää jokin toinen lokaattori. Mahdollisuuksina on OCR, joka yrittää paikallistaa elementin sen sisältämällä tekstillä, mutta se on prosessin kannalta raskasta käyttää, ja se voi tehdä virheitä. Vaihtoehtona on myös käyttää selektoreja erityisesti selaimessa, jossa jokaisella elementillä on uniikki lokaattori (Playwright). Yhdellä elementillä voi olla kuitenkin usea tapa tunnistaa se, ja RPA-ohjelma ei välttämättä aina osaa valita oikeaa tai parasta niistä. Ohjelma voi kuitenkin tarjota listan löydetyistä lokaattoreista, joista kehittäjä sitten valitsee sopivimman.

3.5 Perinteisen ja RPA-mallien yhdistäminen

RPA-robotit voivat tarvita apua perinteiseltä mallilta. Edellisessä kappaleessa käsitelty lokaattorien löytäminen on esimerkki tästä. Paraskin RPA-ohjelman löytämä lokaattori ei välttämättä riitä, jolloin ohjelmoijan täytyy itse hakea ja päätellä paras lokaattori elementille. Joskus mallien yhdistäminen on tehokkaampaa: jos esimerkiksi rajapintojen avulla saadaan suoraan tietoa tietokannasta, rajapinnan käyttäminen voi tehdä robotista paljon nopeamman.

Monimutkaiset robotit erityisesti tarvitsevat useammin apua perinteiseltä mallilta. Jos robotin tarkoitus ei ole ainoastaan ladata tiedostoja netistä vaan myös käsitellä niitä jotenkin, RPA yksin tuskin pystyy siihen.

3.6 RPA:n käyttötapaukset

RPA:ta käytetään usealla eri alalla sen hyötyjen vuoksi. RPA:ta käytetään laajasti vakuutus-, pankki- ja terveydenhuoltoaloilla. Vakuutusalalla päivitetään käyttäjien tietoja, tarkastellaan asiakkaan kelpaavuutta ja hallitaan tilintarkastuksia RPA:n avulla. Dai-ichi Life säästää 132 000 työtuntia vuosittain RPA:n ansiosta. Pankkialalla automatisoidaan asiakaspalvelua, uusien hakemuksien prosessointia ja tiedonsiirtoa. Santander säästi vuonna 2019 30 000 työtuntia ja kaksi miljoonaa Yhdysvaltain dollaria järjestelmien muutossa. Terveydenhuoltoalalla automatisoidaan terveystietojen ylläpitoa ja päivittämistä, elintärkeiden toimintojen monitorointia ja potilaiden ennalta valtuuttamista. R1 RCM automatisoi vuosittain 32 miljoonaa tehtävää. (Automation Anywhere a.)

RPA:n käyttötapaukset eivät kuitenkaan lopu näihin. RPA:ta voidaan käyttää kaikenlaisten ohjelmien automatisoimiseen.

3.7 RPA:n tulevaisuus

RPA:lle ennustetaan suurta nousua seuraavina vuosina. Vuodelle 2023 RPA:n myynnin odotetaan nousevan 17,5 %. Verrattuna vuoden 2022 19,5 % nousuun se on pienempi, mutta edelleen suuri. (Gartner 2022.) RPA-markkinan ennustetaan saavuttavan 23,9 miljardia dollaria vuoteen 2030 mennessä. Vertauksena vuonna 2021 arvioitu markkina oli 2,65 miljardia dollaria. (GlobeNewsWire 2022.) Vuoteen 2024 mennessä 95 % RPA-myyjistä odotetaan tarjoavan automaatiota sekä UI- että API-integraation kautta (Ray ym. 2022).

Hyperautomaatio on seuraava osa automaation tulevaisuutta. Hyperautomaatiolla tarkoitetaan tapaa, jolla organisaatiot tunnistavat ja automatisoivat niin monta IT-prosessia kuin

mahdollista. Hyperautomaatiossa käytetään useita teknologioita, kuten AI:ta, RPA:ta ja koneoppimista. Hyperautomaation tarkoitus on olla kehittyneempi versio RPA:sta: se voi oppia itse tekemään monimutkaisempia taskeja ja sillä on useampia työkaluja, joilla automatisoida niitä. (Frye.)

Hyperautomaation markkinan oletetaan vuonna 2022 saavuttavan lähes 600 miljardin dollarin arvon. Vuonna 2021 tämä arvo oli noin 532 miljardia dollaria. Vuoteen 2024 mennessä organisaatioiden ennustetaan käyttävän ainakin kolmea kahdestakymmenestä prosessiagnostisesta teknologiasta, joista RPA on yksi. Prosessiagnostisella teknologialla tarkoitetaan teknologiaa, jota voidaan käyttää missä tahansa organisaatiossa IT- ja talouskäyttötapauksissa. (Gartner 2021.)

4 Robot Framework

4.1 Esittely

Robot Framework on avoimen lähteen automaatiorakenne, jota käytetään testiautomaatioon, RPA:han ja hyväksymistestaukseen. Se on toteutettu käyttäen Pythonia. Sen kehitystä tukee Robot Framework Foundation. Robot Frameworkin lähdekoodi löytyy GitHubista. (Robot Framework a.) Robot Frameworkilla luodaan robot-tiedostoja, jotka ajon aikana suorittavat niiden sisällä olevat ohjeet. Ohjeisiin kuuluu esimerkiksi sivun avaaminen selaimessa tai Python-tiedoston suorittaminen. Robot Framework sisältää myös muita ominaisuuksia, kuten Seleniumin ja Playwrightin integraation, jotka ovat selaimen automatisoimiseen luotuja kirjastoja. QAutomate käyttää Robot Frameworkia RPA-tarkoituksiin, joten opinnäytetyön tavoitteisiin kuuluu käyttää Robot Frameworkia tuottamaan robottikoodia.

Robot-tiedoston rakenne koostuu useasta eri osiosta. Settings-osiossa voidaan kertoa robotille, mitä muita tiedostoja ja RPA-ominaisuuksia se tarvitsee. Keywordit, eli avainsanat, ovat taas sisäänrakennettuja funktioita. Uusia avainsanoja voidaan myös luoda. Taskit sisältävät robotin toiminnot, kuten Robot Frameworkin omia avainsanoja tai kutsuja muihin kooditiedostoihin. (Robot Framework b.)

4.2 Käyttäminen

Robot Framework tarvitsee ensin Pythonin asennettuna. Sen jälkeen robot-tiedostoja voidaan ajaa suoraan komentoriviltä, jolloin niille voidaan antaa ajoparametreja, kuten mitä taskeja ajetaan. Ajon jälkeen Robot Framework luo raportin HTML-muodossa (Robot Framework c). Kuvassa 4 on epäonnistuneen robotin loki, josta nähdään, mitkä osat suoriutuivat. Lokista nähdään myös Playwrightin tietoa elementeistä ja toiminnoista, joita ajon aikana käsiteltiin. Lokista nähdään myös muuta tietoa, kuten aikaleimat ja robotin suorituksen kesto. Loki on hyödyllinen paikantamaan virheitä, ja siihen voidaan myös tallentaa tärkeää tietoa onnistuneista askeleista.

Total Statistics	Total	Pass	Fail	Skip	Elapsed
All Tasks	1	0	1	0	00:00:17

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed
No Tags					

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed
Robots	1	0	1	0	00:00:19
Robots . Phone Prices Robot	1	0	1	0	00:00:19
Robots . Phone Prices Robot . Phone Prices Robot	1	0	1	0	00:00:19

Task Execution Log

SUITE Robots
Full Name: Robots
Source: [redacted]
Start / End / Elapsed: 20221125 14:53:27.819 / 20221125 14:53:47.004 / 00:00:19.185
Status: 1 task total, 0 passed, 1 failed, 0 skipped

SUITE Phone Prices Robot
Full Name: Robots.Phone Prices Robot
Source: [redacted]
Start / End / Elapsed: 20221125 14:53:27.851 / 20221125 14:53:47.001 / 00:00:19.150
Status: 1 task total, 0 passed, 1 failed, 0 skipped

SUITE Phone Prices Robot
Full Name: Robots.Phone Prices Robot.Phone Prices Robot
Source: [redacted]
Start / End / Elapsed: 20221125 14:53:27.853 / 20221125 14:53:46.988 / 00:00:19.135
Status: 1 task total, 0 passed, 1 failed, 0 skipped

SETUP phone_prices_robot . Suite Setup Actions
TEARDOWN phone_prices_robot . Suite Teardown Actions

TASK phone_prices_robot
Full Name: Robots.Phone Prices Robot.Phone Prices Robot.phone_prices_robot
Start / End / Elapsed: 20221125 14:53:30.230 / 20221125 14:53:46.890 / 00:00:16.660
Status: FAIL
Message: TimeoutError: locator.elementHandle: Timeout 10000ms exceeded.
========== logs ==========
waiting for selector "text=S10"
==========

SETUP phone_prices_robot . Test Setup Actions
KEYWORD qautomatelibrary.BrowserExtension . New Page Windowed https://www.verkkokauppa.com/
KEYWORD shared_keywords.AllComponents . Browse Verkkokauppa
KEYWORD shared_keywords.AllComponents . Find Phones
Start / End / Elapsed: 20221125 14:53:36.844 / 20221125 14:53:46.887 / 00:00:10.043
14:53:46.887 FAIL TimeoutError: locator.elementHandle: Timeout 10000ms exceeded.
========== logs ==========
waiting for selector "text=S10"
==========
KEYWORD shared_keywords.AllComponents . Get Phone Prices
TEARDOWN phone_prices_robot . Test Teardown Actions

Kuva 4. Robotin loki

Robot Framework voi ajaa sen sisäisiä komentoja suoraan robottitiedostossa. Komentoihin kuuluu esimerkiksi ehtolausekkeet, viestien tuottaminen ja yli sadan eri avainsanan ajaminen. (Robot Framework b.) Robotille voidaan myös asettaa mitä sen kuuluisi tehdä,

jos robotin suoritus epäonnistuu. Lisäksi robottitiedosto voi kutsua Python-tiedostoja, tai käyttää esimerkiksi Playwright-kirjastoa.

4.3 Playwright

Playwright on Microsoftin kehittämä Node.js-kirjasto, jolla automatisoidaan useita selaimia, kuten Chromea, Firefoxia ja Microsoft Edgeä. Playwright on saatavilla Windowsille, Linuxille ja macOS:ille, ja se on saatavilla useille ohjelmointikielille, kuten Pythonille, Javalle ja JavaScriptille. Playwrightin avulla käytetään ”web scraping” -teknologiaa, eli tapaa kerätä dataa netistä. Playwright sisältää useita selaimen automaatioon käytettäviä komentoja, kuten hiiren klikkaaminen, näppäimistön näppäimien painaminen, tiedostojen lataaminen ja sivujen vaihtaminen. (Playwright.)

Playwright tukee useita ominaisuuksia, jotka helpottavat selaimen automatisointia. Playwright automaattisesti odottaa, että selaimessa oleva elementti on valmis. Playwright siis odottaa, että elementti on kokonaan latautunut sivulle, ennen kuin robottikoodi lähettää elementille komentoja. Jos elementtiä ei löydy määrätyn ajan sisällä, Playwright palauttaa virheen. Playwright myös tukee useita eri tapoja saada elementille uniikki lokaattori, kuten tekstin löytäminen ja CSS- ja XPath-selektorien löytäminen, ja niitä voidaan käyttää yhdessä. (Robocorp a.) CSS-selektoreilla valitaan HTML-elementtejä niiden ominaisuuksien perusteella (MDN Web Docs a). Esimerkiksi kaikki napit, joilla on luokka ”submit”, löydetään käyttämällä selektoria ”button.submit”. XPath taas on tapa navigoida XML-tyyppisiä tiedostoja, kuten HTML-tiedostoja (MDN Web Docs b). Esimerkiksi ”//html/body/div/button” löytää elementin vastaavasta polusta, jos elementti on olemassa.

Playwrightin selaimet koostuvat kolmesta eri tasosta: Browser, Context ja Page. Browser määrittelee, mitä selainta käytetään ajon aikaa. Browser tukee kaikkia Playwrightin selaimia. Oletuksena Browser luodaan ”headless”-tilassa, eli ilman GUI:ta. Tämä voidaan kuitenkin halutessa muuttaa. Context vastaa joukkoa itsenäisistä sivuista (Page). Contextit jakavat useita asetuksia, kuten cookies-, sessio- ja profiiliasetuksia. Sivut eri Contexteissa eivät jaa näitä asetuksia. Context on hyödyllinen testaamaan erilaisia käyttäjäsessioita ilman että tarvitsisi avata uutta Browser-kopiota. Contexteille voidaan myös asettaa eri asetuksia, kuten eri kieliasetukset ja selaimen koon. Page, eli sivu, vastaa yksittäistä sivua tai välilehteä. Page sisältää ladatun sivun sisällön ja selaushistorian. (Robocorp b.)

5 Koodin muuntaminen vuokaavioksi

5.1 Tavoitteet

Ensimmäinen opinnäytetyön tavoitteista on muuntaa robottikoodia vuokaavioksi. Kooditiedostot syötetään sovellukselle, joka lukee niiden sisällön ja muuntaa kaikki tarvittavat funktiot ja ehtolausekkeet noodeiksi ja edgeiksi vuokaavioon.

Robot-tiedosto luetaan ja muunnetaan GraphML-tiedostoksi. GraphML-tiedoston tarkoitus on auttaa kuvaamaan robotin prosessia, ja tiedostoa voidaan tarvittaessa muokata.

5.2 Koodin lukeminen

Robot-tiedosto luetaan rivi riviltä. Koodia aletaan käsitellä, kun ensimmäinen taski alkaa. Jokainen haluttu rivi lisätään Python-listaan, josta luodaan vuokaavion noodit. Askeleille annetaan myös oma ID, joka on numero, jolla osoitetaan, kuinka mones askel on kyseessä. Lisäksi luodaan askeleiden väliset yhteydet eli edget.

Tällä hetkellä ohjelma käsittelee edget vain yksisuuntaisten, suoraan menevien askeleiden edget. Ohjelma voidaan mahdollisesti myöhemmin kehittää käsittelemään myös monimutkaisemmat edge-yhteydet.

5.3 Poikkeuksien käsittely

Kaikkea koodia ei voida suoraan muuntaa vuokaavioksi. Jotkin tapaukset pitää käsitellä erikseen ja omalla tavallaan. Esimerkiksi robot-tiedostossa muuttujat ovat muodossa ”${muuttujan_nimi}”. Vuokaavioon ei haluta ylimääräisiä merkkejä, kuten muuttujassa $-merkkiä tai aaltosulkuja, joten ne poistetaan välistä. Muuttujien nimistä voidaan myös tarvittaessa poistaa tai korvata merkkejä, eli muutetaan esimerkiksi alaviivat välilyönneiksi.

Joitakin rivejä ei haluta ollenkaan. Esimerkiksi jos rivi alkaa $-merkillä, se tarkoittaa, että sillä rivillä määritellään muuttuja. Sellaista tietoa ei haluta vuokaavioon, tosin tähänkin voi olla poikkeuksia. Poikkeukset määritellään Excel-tiedostossa, johon voidaan nopeasti lisätä, poistaa tai muuttaa tietoja, joiden perusteella poikkeukset käsitellään.

5.4 GraphML-tiedoston luominen

GraphML-tiedoston luomista varten tarvitsee noudattaa sen rakennetyyliä. Tätä varten luotiin kuvan 5 tiedosto, jossa on tarvittavat elementtipohjat, joiden avulla on helppo luoda useita GraphML-tiedostoja.

```
18  node_str = """
19      <node id="{id}">
20        <data key="d5"/>
21        <data key="d6">
22          <y:ShapeNode>
23            <y:Geometry height="68.0" width="{width}" x="{x}" y="{y}"/>
24            <y:Fill color="#FFCC00" transparent="false"/>
25            <y:BorderStyle color="#000000" raised="false" type="line" width="1.0"/>
26            <y:NodeLabel alignment="center" autoSizePolicy="content" fontFamily="Dialog"
27            <y:Shape type="{shape}"/>
28          </y:ShapeNode>
29        </data>
30      </node>"""
31
32  normal_edge = """
33      <edge id="{id}" source="{src}" target="{target}">
34        <data key="d9"/>
35        <data key="d10">
36          <y:PolyLineEdge>
37            <y:Path sx="0.0" sy="0.0" tx="0.0" ty="0.0"/>
38            <y:LineStyle color="#000000" type="line" width="1.0"/>
39            <y:Arrows source="none" target="standard"/>
40            <y:BendStyle smoothed="false"/>
41          </y:PolyLineEdge>
42        </data>
43      </edge>"""
44
45  branch_edge = """
46      <edge id="{id}" source="{src}" target="{target}">
47        <data key="d9"/>
48        <data key="d10">
49          <y:PolyLineEdge>
50            <y:Path sx="0.0" sy="0.0" tx="0.0" ty="0.0"/>
51            <y:LineStyle color="#000000" type="line" width="1.0"/>
52            <y:Arrows source="none" target="standard"/>
53            <y:EdgeLabel alignment="center" configuration="AutoFlippingLabel" distance="2
54            <y:BendStyle smoothed="false"/>
55          </y:PolyLineEdge>
56        </data>
57      </edge>"""
58
59  end_str = """
60    </graph>
61    <data key="d7">
62      <y:Resources/>
63    </data>
64  </graphml>"""
```

Kuva 5. GraphML-tiedoston elementtipohjat

Elementeille tarvitsee vain syöttää välttämättömät tiedot, joilla elementit erotetaan toisistaan. Lisäksi syötetään muuta tietoa, kuten noodin muoto. Tiedosto luodaan yhden merkkijonon avulla, jonka nimi on ”self.xml”. Aluksi merkkisarjaan lisätään GraphML-tiedoston välttämättömät elementit. Merkkisarjaan sitten lisätään noodien ja edgejen tiedot.

Tiedostoon luodaan ensin noodit. Aina aluksi luodaan Start-noodi, joka aloittaa vuokaavion. Sen jälkeen luodaan muut noodit kerrallaan. Niiden sisältämä teksti, muoto ja sijainti syötetään. Noodin Y-koordinaatti kasvaa jokaisen noodin jälkeen, jotta noodit eivät päädy päällekkäin. Lopuksi luodaan vielä Stop-noodi, joka kuvastaa vuokaavion loppua.

Seuraavaksi luodaan edget. Aluksi luodaan Start-noodin ja sitä seuraavan noodin välinen yhteys. Sitten luodaan yksitellen muut edget. Jokaiselle edgelle luodaan oma ID, ja yhteys luodaan antamalla lähtönoodi ja kohdenoodi. Lopuksi luodaan viimeisen noodin ja Stop-noodin välinen yhteys.

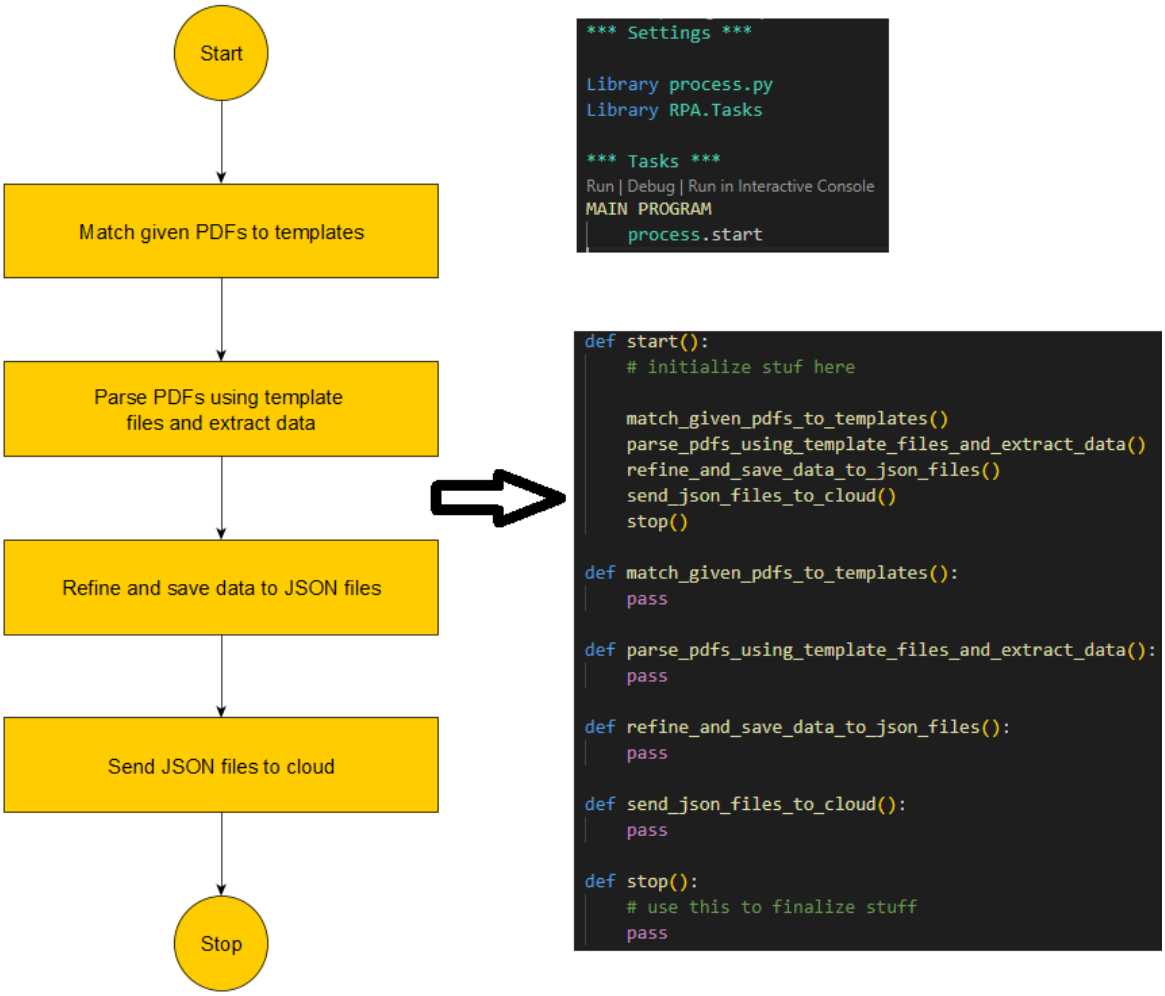
Merkkisarjaan lisätään vielä viimeinen elementtipohja, jossa päätetään kaavion luominen. Ennen tiedoston kirjoittamista luodaan ”graphs”-kansio, jos sitä ei vielä ole. Kansion alle luodaan alikansio, joka saa nimekseen robotin nimen. Lopuksi merkkisarja kirjoitetaan tiedostoon alikansion alle.

6 Vuokaavion muuntaminen koodiksi

6.1 Tavoitteet

Opinnäytetyön seuraava tavoite on muuntaa vuokaavio koodiksi. Vuokaavio, jonka esimerkiksi asiakas on luonut, syötetään sovellukselle, joka muuntaa sen robot-tiedostoksi sekä Python-tiedostoksi. Sovellus lukee vuokaavion sisällön ja muuntaa noodit ja edget funktioiksi ja funktiokutsuiksi. Funktiot ovat tyhjiä, paitsi jos koodi luodaan selainmallilla.

Kuvassa 6 osoitetaan haluttu tulos: jokainen vuokaavion noodi muunnetaan Python-funktioksi, joita kutsutaan oikeassa järjestyksessä. Robottitiedosto kutsuu ensimmäistä, eli Start-funktiota, joka laittaa robotin käyntiin. Kuvan esimerkkikaavio muunnettiin Python-painotteisella versiolla ohjelmasta.



Kuva 6. Vuokaavion muuntaminen koodiksi

6.2 Vuokaavion lukeminen

Vuokaavion tiedosto luetaan Pythonissa käyttämällä ”xml.etree.ElementTree” -moduulia. Sillä voidaan lukea XML-tyyppisiä tiedostoja, kuten GraphML-tiedostoja. Tiedostosta haetaan root eli juurielementti ja sen välittömästi alla oleva pääosuus. Pääosuudesta haetaan omiin listoihin noodit ja edget. Ne sisältävät kaiken tarvittavan tiedon.

Noodeista hankitaan niiden sisältämä teksti sekä ID. Edgeistä taas haetaan lähtönoodi, kohdenoodi sekä niiden sisältämä teksti, eli ”kyllä” tai ”ei” jos se on asetettu.

6.3 Erityismerkkien korvaaminen ja muut poikkeukset

Python-funktioiden nimet ovat samat kuin vuokaavion noodien tekstisisällöt. Python-funktioiden nimissä ei kuitenkaan voi olla erityismerkkejä, joten erikoismerkit pitää käsitellä. Esimerkiksi jos tekstissä on matemaattisia symboleja, ne pitää jotenkin korvata tai poistaa. Kuvassa 7 määritellään taulukko, jonka mukaan merkit korvataan. Funktion nimessä taulukon avain korvataan sen vastinparilla. Jos noodin teksti on esimerkiksi ”tallenna tulot + menot”, funktion nimeksi tulee ”tallenna_tulot_plus_menot”.

```
26 replace_dict = {
27     "+": "plus",
28     "-": "minusOrDash",
29     "/": "fSlash",
30     "=": "equals",
31     "#": "hashtag",
32     "$": "dollarSign",
33     "%": "percent",
34     "&": "and",
35     "<" : "less_than",
36     ">": "greater_than",
37     "*": "asterisk",
38     '"': "doubleQuote",
39     "'": "singleQuote",
40     ".": "dot",
41     ":": "colon",
42     ";": "semiColon",
43     "@": "atSign",
44     "  ": "_",
45     " ": "_",
46     "ä": "a",
47     "ö": "o"
48 }
```

Kuva 7. Taulukko korvaamiseen

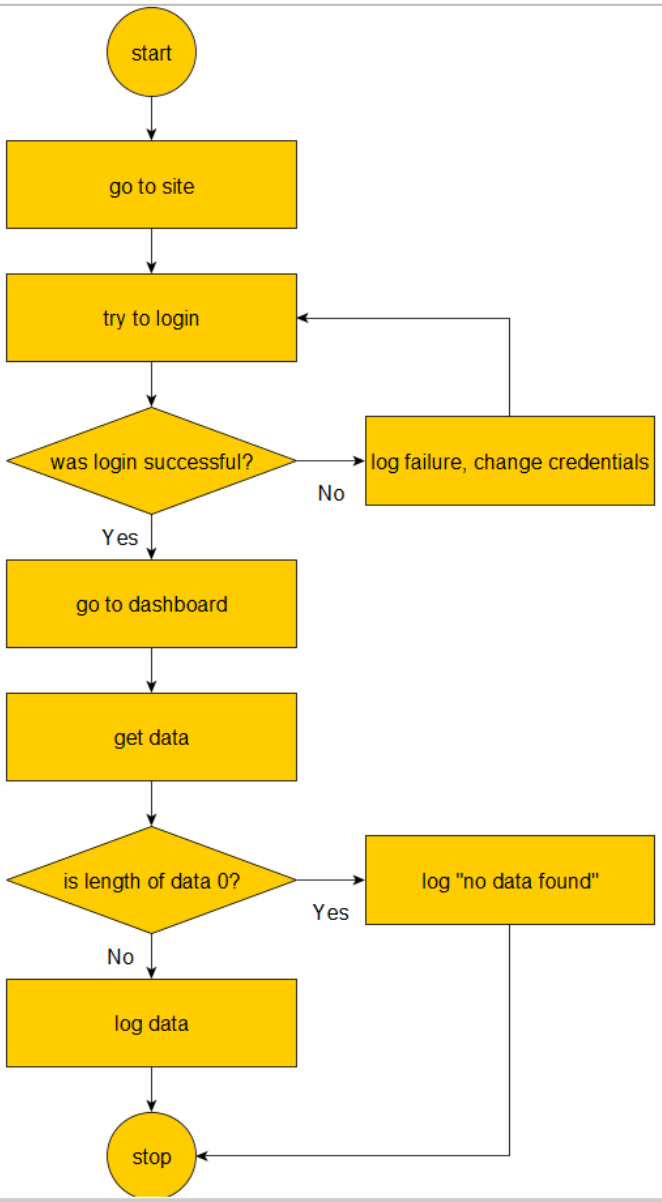
Python-funktion nimi ei voi myöskään alkaa numerolla. Tässä tapauksessa funktion nimen alkuun lisätään ”num”, jolloin nimi toimii. Tämä ei ole paras ratkaisu, mutta sovelluksessa se riittää.

Kaikki merkit, jotka eivät esiinny kuvassa 7, poistetaan funktion nimestä kokonaan. Merkit poistetaan regexin (regular expression) avulla. Regexit ovat merkkisarjoja, joilla etsitään tekstistä kuvioita (MDN Web Docs c). Koodissa käytetään ”\W”-regexiä, jolla poistetaan kaikki merkit paitsi kirjaimet, numerot ja alaviivat.

6.4 Haarautuminen

Vuokaaviot voivat olla todella monimutkaisia. Esimerkiksi yksi noodi voi johtaa useaan noodiin, tai yhteen noodiin voidaan tulla useasta noodista. Haarautuminen pitää saada toteutettua myös koodin puolella.

Koodi jaetaan osiin, jotta koodin pituudesta ei tule loputon. Jos esimerkiksi olisi noodisilmukka, jossa osa noodeista voi kiertää tietyissä tapauksissa ympyrää, sitä ei voisi kirjoittaa koodiksi ilman koodin jakamista osiin. Koodi hajotetaan taskeiksi sen mukaan, kuinka monta tulevaa yhteyttä jokaisella askeleella on. Jos niitä on kaksi tai enemmän, siitä tehdään oma taski. Tämä estää loputtomat koodit sekä siistii koodia. Kuvan 8 vuokaaviossa noodiin ”try to login” johtaa kaksi muuta noodia, joten siitä tehdään oma osio. Kun koodi on jaettu osiin, osat voivat jatkua normaalisti, kunnes ne osuvat toiseen osaan. Silloin luodaan robottiin uusi taski, joka koostuu seuraavasta osasta.



Kuva 8. Haarautuva vuokaavio

Vuokaavio voi myös jakautua kyllä-ei-valinnan kautta. Koodissa tällainen ilmaistaan if-else-lauseella. Robot-tiedostossa noudatetaan sen omaa syntaksia ja luodaan sopivat sisennykset, jotta koodi toimii. Robot-tiedostossa luodaan aina Start- ja Stop-taskit, sekä väliin kaikki haarautuvat taskit. Kuvassa 9 robot-tiedostossa on lisäksi ”try to login” -taski. Robotti aina alkaa Start-taskista ja loppuu -Stop-taskiin. Haarautuneessa mallissa käytetään robot-painotteista mallia, joten suurin osa logiikasta on robot-tiedoston puolella.

```
*** Settings ***

Library proper.py
Library RPA.Tasks

*** Tasks ***
Run | Debug | Run in Interactive Console
start
    proper.start
    proper.go_to_site
    Set next task   try_to_login
Run | Debug | Run in Interactive Console
try_to_login
    proper.try_to_login
    ${cond} =   proper.was_login_successful
    IF  ${cond}
        proper.go_to_dashboard
        proper.get_data
        ${cond} =   proper.is_length_of_data_0
        IF  ${cond}
            proper.log_doubleQuoteno_data_founddoubleQuote
            Set next task   stop
        ELSE
            proper.log_data
            Set next task   stop
        END
    ELSE
        proper.log_failure_change_credentials
        Set next task   try_to_login
    END
Run | Debug | Run in Interactive Console
stop
    proper.stop
```

```
def start():
    # use this to initialize stuff
    pass

def go_to_site():
    pass

def try_to_login():
    pass

def was_login_successful():
    if True:
        return True
    return False

def go_to_dashboard():
    pass

def get_data():
    pass

def is_length_of_data_0():
    if True:
        return True
    return False

def log_data():
    pass

def log_failure_change_credentials():
    pass

def log_doublequoteno_data_founddoublequote():
    pass

def stop():
    # use this to finalize stuff
    pass
```

Kuva 9. Haarautunut robottikoodi

6.5 Python-tiedoston kirjoittaminen

Python-tiedoston kirjoittaminen on suurimmaksi osaksi yksinkertaista. Jokainen vuokaavi-on noodi suoraan kirjoitetaan tiedostoon tyhjänä funktiona, paitsi jos kyseessä on noodi, jossa on kyllä-ei-yhteys. Siinä tapauksessa funktion sisällöksi tulee kuvassa 9 kuvattu funktion sisältö.

Jos kyseessä on start- tai stop-funktio, niihin lisätään kommentti ilmaisemaan, että niitä on tarkoitus käyttää joko ajamisen aloittamiseen tai lopettamiseen. Python-painotteisessa mallissa funktiokutsut kirjoitetaan start-funktioon, kuten kuvassa 6 sivulla 17 on kuvattu.

6.6 Robot-tiedoston kirjoittaminen

Robot-tiedoston alkuun kirjoitetaan muutama tarvittava osa, joita robotti tarvitsee toimiakseen. Settings-osuuteen lisätään kirjastona luotu Python-tiedosto, jotta sitä voidaan ajaa, sekä RPA.Tasks -kirjasto, joka sisältää tärkeitä ominaisuuksia. Sen jälkeen luodaan Tasks-osuus, jonne tulee koodin ajaminen.

Tasks-osuuteen kirjoitetaan kaikki taskit, jotka on jaettu haarautumisen avulla. Ensimmäinen taski on aina Start-taski, joka aloittaa robotin ajamisen. Siitä siirrytään tarvittaessa muihin välissä oleviin taskeihin. Lopuksi lisätään aina Stop-taski, joka vain kutsuu omaa Python-funktiotaan.

6.7 Selainmalli

RPA:ta käytetään usein selaimessa tapahtuvien toimintojen automatisoimiseen. Selaimen automatisoimiseen Robot Frameworkissa ja Pythonissa käytetään Playwrightia. Koska selaimen automatisoiminen on yleinen tarve, sille luodaan oma pohja, joka muuntaa vuokaaviot koodiksi.

Sovellus luo sopivamman pohjan robotille, joka olettaa, että selainta käytetään. Pohja sisältää QAutomaten sisäisiä kirjastoja, sekä yleisen rakenteen, kuten luokan ”component” luomisen ja tyhjän ”__init__”-funktion, kuten kuvassa 10. Yksi sisäisistä kirjastoista käytetään Plawrightin käyttämiseen.

```
1  from robot.api.deco import keyword
2  from qautomatelibrary.BrowserExtension import BrowserExtension
3  browser = BrowserExtension()
4
5  class component:
6      def __init__(self):
7          pass
```

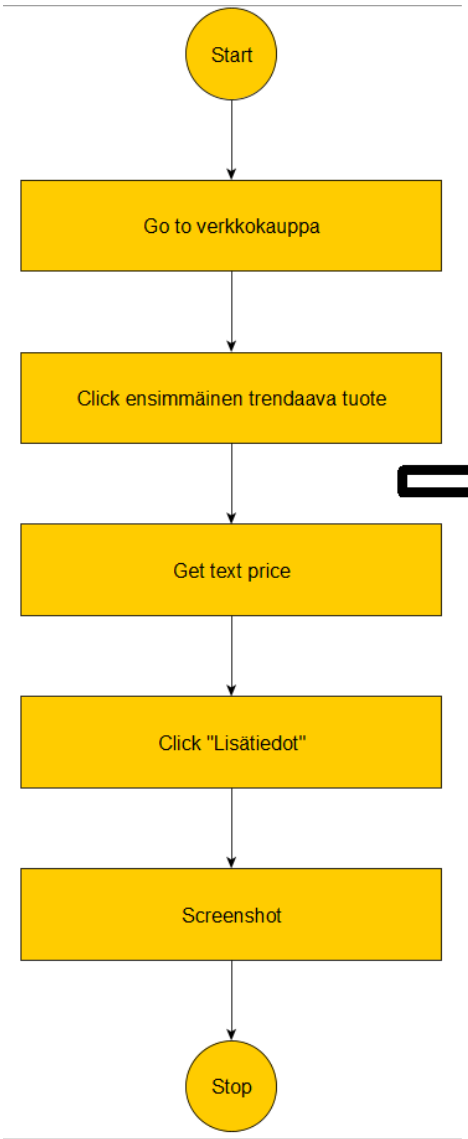
Kuva 10. Tyhjä selainmallin pohja

Sovelluksien luomia funktiota voidaan automatisoida pidemmälle selainmallissa. Vuokaavion noodeihin voidaan laittaa tarkempaa tekstiä, jonka sovellus käsittelee ja kääntää parhaan mukaan koodiksi. Jos esimerkiksi noodin teksti on ”click Kirjaudu sisään”, sovellus olettaa, että kyseessä on klikkaus. Sovelluksella on kuvassa 11 kuvattu lista eri avainsanoista, ja jos se löytää jonkin niistä noodin tekstin alussa, se voi muuntaa ne koodiksi. Kaaviossa avainsanat ovat järkevämmässä muodossa, mutta ne muunnetaan koodissa ajettaviksi Playwright-avainsanoiksi.

```
keyword_dict = {
    "click": "click",
    "hover": "hover",
    "go to": "go_to",
    "input text": "type_text",
    "input secret": "type_secret",
    "get text": "get_text",
    "press keys": "press_keys",
    "download": "download",
    "count": "get_element_count",
    "screenshot": "take_screenshot"
}
```

Kuva 11. Taulukko avainsanoista

Jos termi on avainsanan jälkeen lainausmerkkien sisällä, robotti olettaa sen olevan tekstiä, joka löytyy selaimesta. Esimerkiksi kuvassa 12 noodin teksti ”click ’Lisätiedot’” muutetaan seuraavaksi koodiksi: ”browser.click(”text=’Lisätiedot’)”. Jos selaimessa on vain yksi elementti sillä hetkellä, joka sisältää vastaavan tekstin, robotti osaa klikata sitä. Jos termi sen sijaan ei ole lainausmerkkien sisällä, sovellus luo muuttujan, jonka nimeksi tulee muunneltu versio termistä. Muuttujan arvoksi tulee tyhjä merkkijono, joka toimii paikanpitäjänä. Muuttujalle voidaan myöhemmin määritellä lokaattori.




```python
from robot.api.deco import keyword
from qautomatelibrary.BrowserExtension import BrowserExtension
browser = BrowserExtension()

class component:
    VERKKOKAUPPA = ""
    ENSIMMAINEN_TRENDAAVA_TUOTE = ""
    PRICE = ""

    def __init__(self):
        pass

    @keyword
    def start(self):
        browser.go_to(self.VERKKOKAUPPA)
        browser.click(self.ENSIMMAINEN_TRENDAAVA_TUOTE)
        browser.get_text(self.PRICE)
        browser.click("text='Lisätiedot'")
        browser.take_screenshot()
```

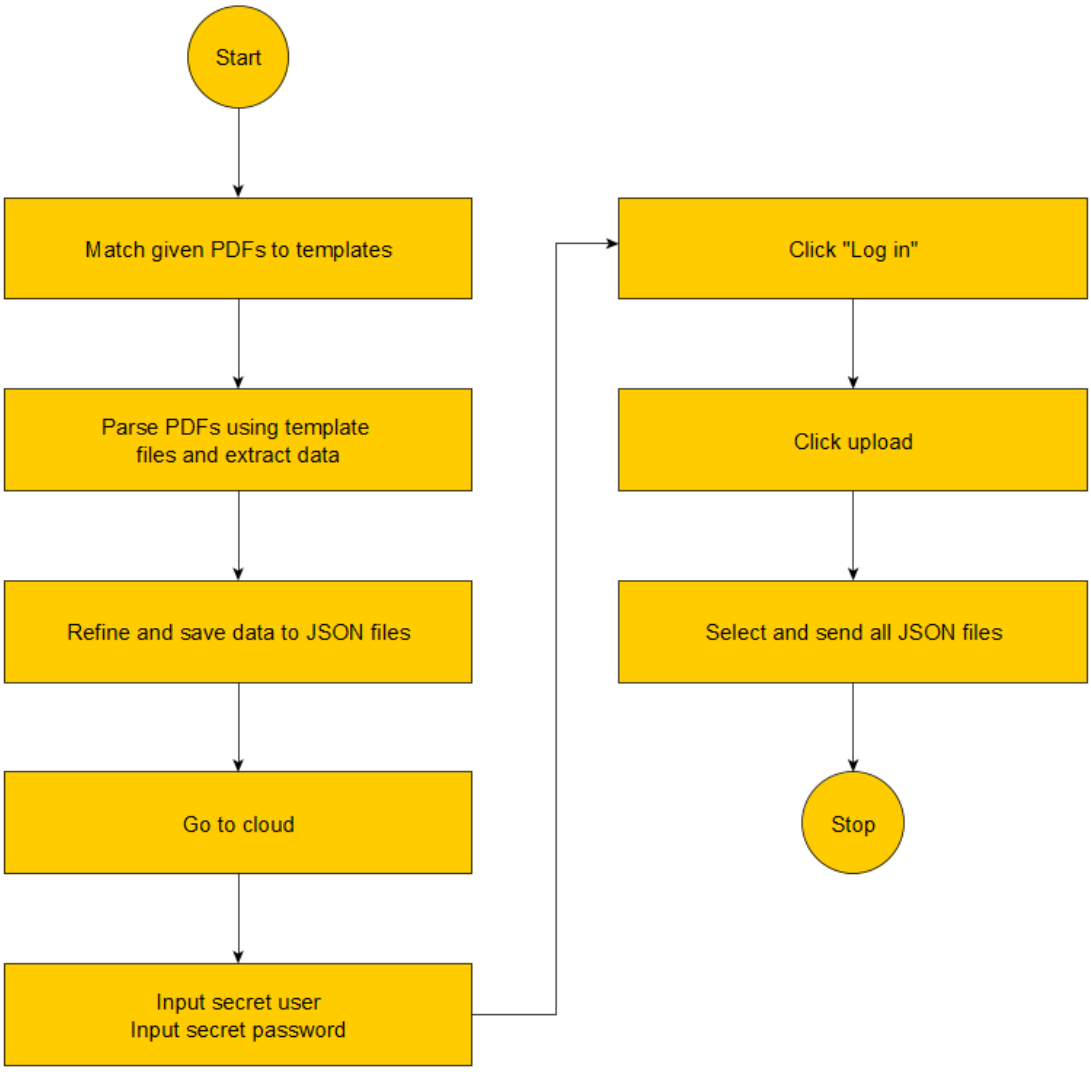
Kuva 12. Vuokaavion muuntaminen selainmalliksi

7 Caset

7.1 Case: PDF-tiedostojen parsiminen ja tiedon tallentaminen JSON-tiedostoihin

Tässä luvussa esitellään todellinen tilanne, jossa opinnäytetyössä tuotettu sovellus auttaa tuottamaan pohjan työprojektille. Kyseessä on projekti, jossa halutaan käsitellä kymmeniä PDF-tiedostoja, lukea niistä halutut tiedot ja tallentaa ne JSON-tiedostoihin, jotka sitten ladataan pilvipalveluun. Koska robotti tarvitsee yhteyden pilvipalveluun selaimessa, robotille määritellään selainpohja.

Aluksi projektin havainnollistamista varten luotiin kuvan 13 vuokaavio, jossa kuvataan tarkoin projektin askeleet. Asiakas voi luoda vuokaavion itse yEdin tai QAutoCapturen avulla. Tässä tapauksessa opinnäytetyön kirjoittaja loi vuokaavion QAutoCapturen avulla. Ensin varmistetaan, että vuokaavio on oikean mallinen, vaikka sovelluksella on viansietoa. Aluksi prosessissa tehdään asioita selaimen ulkopuolella, joita ei voida automatisoida pidemmälle. Koska prosessissa myöhemmin käsitellään selaimessa tapahtuvia toimintoja, vuokaaviota voidaan kehittää pidemmälle eri avainsanojen ansiosta. Useita toimintoja, kuten klikkauksia ja navigointia automatisoidaan vuokaavion avulla. Joitakin elementtejä etsitään tekstin avulla, osa jätetään tyhjiksi. Koska vuokaaviossa ei ole haaroja, prosessissa ei esiinny kohtia, joissa robotin pitäisi muuttaa toimintaansa.



Kuva 13. Case-vuokaavio

Sovelluksen parametreihin syötetään haluttu robottimuoto ja luotu vuokaavio. Sovellus käynnistetään ja se käsittelee ja muuntaa vuokaaviota parhaan mukaan ajettavaksi robotti- ja Python-koodiksi. Kuvassa 14 nähdään sovelluksen tuottama lopputulos. Robot-tiedosto on lähes tyhjä, mutta Python-tiedosto on luotu käyttäen selainmallia, joka on Python-painotteinen. Koodi ensin suorittaa funktioita, joihin ei tarvita selainta, jonka jälkeen suoritetaan Playwrightin avainsanoja. Lopuksi koodi vielä suorittaa yhden funktion.

```
*** Settings ***

Library RPA.Tasks

*** Tasks ***
Run | Debug | Run in Interactive Console
MAIN PROGRAM
    Start
```

```python
from robot.api.deco import keyword
from qautomatelibrary.BrowserExtension import BrowserExtension
browser = BrowserExtension()

class component:
    CLOUD = ""
    USER = ""
    PASSWORD = ""
    UPLOAD = ""

    def __init__(self):
        pass

    @keyword
    def start(self):
        self.match_given_pdfs_to_templates()
        self.parse_pdfs_using_template_files_and_extract_data()
        self.refine_and_save_data_to_json_files()
        browser.go_to(self.CLOUD)
        browser.type_secret(self.USER, "")
        browser.type_secret(self.PASSWORD, "")
        browser.click("text='Log in'")
        browser.click(self.UPLOAD)
        self.select_and_send_all_json_files()
        self.stop()

    def match_given_pdfs_to_templates():
        pass

    def parse_pdfs_using_template_files_and_extract_data():
        pass

    def refine_and_save_data_to_json_files():
        pass

    def select_and_send_all_json_files():
        pass

    def stop():
        # use this to finalize stuff
        pass
```

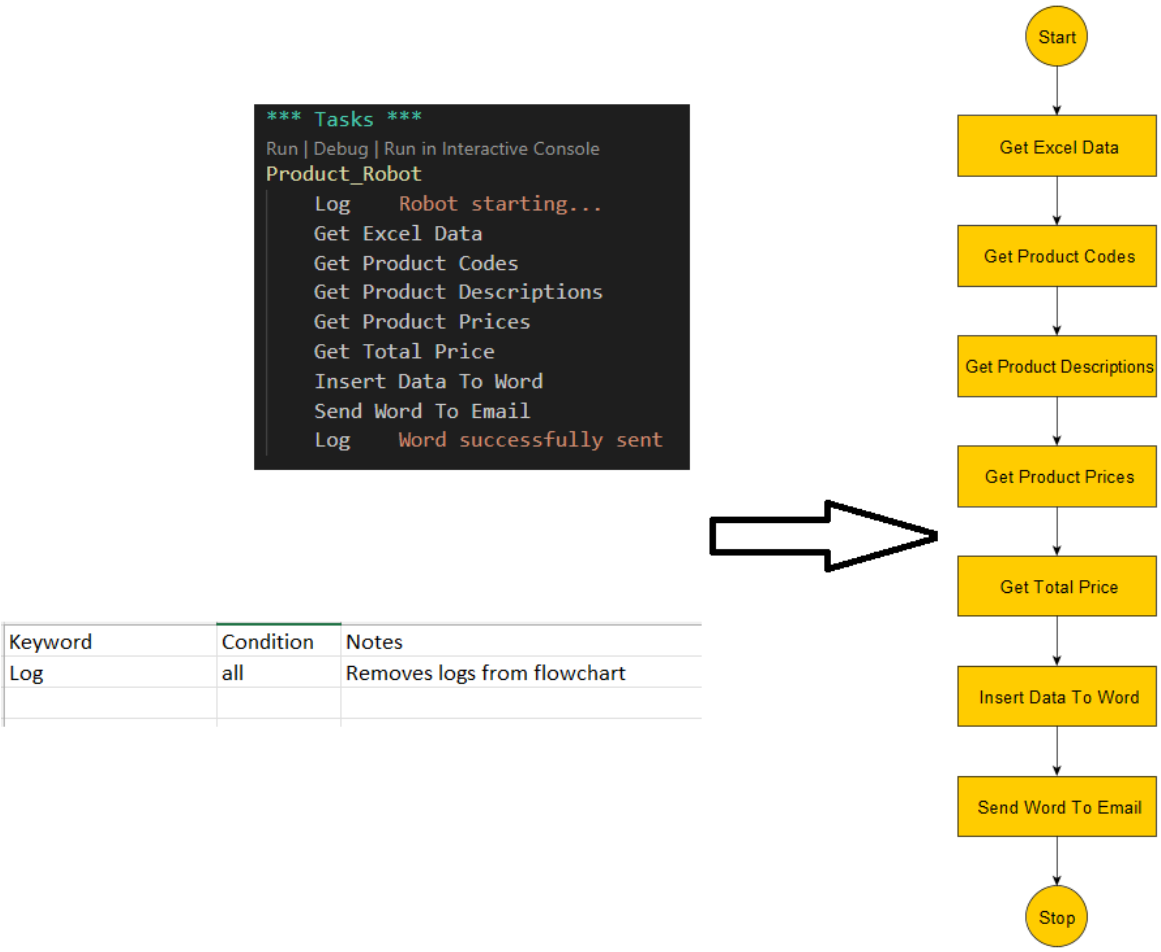
Kuva 14. Case-robottikoodi

Sovellusta voidaan käyttää hyödyksi myös toisin päin: jos koodia on jo luotu ja halutaan havainnollistaa asiakkaalle robotin toimintaa, voidaan koodi muuntaa vuokaavioksi. Aluksi luodaan säännöt Excel-tiedostoon, jonka mukaan ohjelma luo vuokaavion. Sen jälkeen ohjelma vain ajetaan. Ohjelma muuntaa koodin paremmin luettavaan muotoon.

7.2 Case: Robottikoodin muuntaminen vuokaavioksi

Tässä luvussa esitellään tilanne, jossa hypoteettinen robottikoodi muunnetaan vuokaavioksi. Koodi halutaan muuntaa vuokaavioksi asiakasta tai dokumentaatiota varten. Hypoteettisessa robotissa käsitellään datan hakemista Excel-tiedostosta. Tieto sitten siirretään Word-tiedostoon, joka lähetetään sähköpostilla asiakkaalle.

Kuvassa 15 näkyy hypoteettinen robottikoodi, joka muunnettiin sovelluksen avulla vuokaavioksi. Robot-tiedostosta luettiin taskin avainsanat, jotka muunnettiin noodeiksi. Robot-tiedostosta ei haluta lokeja noodeiksi, joten niistä tehtiin poikkeus Excel-tiedostoon, jonka avulla poistetaan kaikki lokit. Lopuksi luotiin noodien väliset edget. Vuokaavio voidaan tarvittaessa luoda uudelleen, jos robot-tiedostoon tulee lisää koodia.



Kuva 15. Robottikoodi vuokaavioksi

8 Yhteenveto ja pohdinta

Opinnäytetyön tavoitteisiin kuului sovelluksen luominen, joka muuntaa vuokaavioita robotti- ja Python-tiedostoiksi, joita voidaan käyttää pohjana robottiprojekteille. Sovellusta voidaan myös käyttää muuntamaan koodia vuokaavioiksi. Sovelluksen tavoitteena on auttaa RPA-kehittäjiä luomaan nopeammin ja helpommin pohjia robottiprojekteille. Toisena tavoitteena oli tutkia RPA-teknologiaa ja sen hyötyjä ja tulevaisuuden ennustuksia. RPA:n yhteydessä tutkittiin myös siihen liittyviä teknologioita, kuten OCR-teknologiaa, jonka avulla voidaan automatisoida erityisesti työpöytäsovelluksia löytämällä ruudulta tekstiä. Playwright-kirjastolla taas automatisoidaan useita selaimia luomalla kopio selaimesta, johon voidaan lähettää useita komentoja. Robot Framework taas on rakenne, jota käytetään robottien luomisessa. Opinnäytetyössä vuokaavioita käsiteltiin yEdin avulla, joka on ilmainen kaavioiden luomiseen tarkoitettu sovellus.

RPA on hyödyllinen työkalu, jonka avulla voidaan automatisoida yksinkertaisia ja toistettavia tehtäviä. RPA-mallia voidaan käyttää yhdessä perinteisen mallin kanssa, jolloin automatisoimisella ei melkein ole rajoja. RPA-teknologialle ennustetaan suurta kasvua tulevaisuudessa, ja se tulee olemaan tärkeä osa hyperautomaatiota, joka on automaation seuraava askel tulevaisuudessa. Kasvun taustalla on monet RPA:n vahvuudet, kuten ajan säästäminen, virheiden vähentäminen ja rajapintojen tarpeen kiertäminen.

Opinnäytetyössä luotu sovellus osaa muuntaa vuokaavion koodiksi, kunhan vuokaavio noudattaa tiettyä rakennetta. Vuokaavion noodien tekstit muunnetaan ajettavaksi koodiksi, ja niistä tarvittaessa poistetaan merkit, joita ei voida käyttää koodissa. Avainsanoja käytetään automatisoimaan prosessia pidemmälle. Jos vuokaaviossa esiintyy haarautumista, se muutetaan ehtolausekkeiksi. Kun koodia muunnetaan vuokaavioiksi, koodista poistetaan ylimääräiset merkit, ja vuokaavion rakenne luodaan käyttämällä valmiita merkkijonoja, joihin syötetään koodissa olevien funktioiden nimet. Noodit sitten järjestellään ja yhdistetään edgeillä. Sovellus voi säästää kehittäjien aikaa robottiprojektien tekemisessä tai se voi auttaa tekemään vuokaavioita.

Opinnäytetyössä luotua sovellusta voitaisiin kehittää pidemmälle lisäämällä siihen erilaisia tapoja automatisoida vuokaavioiden muuntamista koodiksi. Esimerkiksi lisäämällä OCR-teknologia työpöytäsovelluksia voitaisiin automatisoida löytämällä tekstiä ja klikkaamalla löydettyjä koordinaatteja. Sovellukseen voitaisiin myös lisätä avainsanoja, joita käyttää automatisoimisessa. Sovellus voisi myös luoda koko robottirakenteen. Sovellus voisi myös olla enemmän virheensietoinen, kun vuokaavioita muunnetaan koodiksi. Sovelluksessa merkkien korvaamista voitaisiin kehittää pidemmälle niin, että tuloksesta tulisi hel-

pommin luettavaa ja järkevämpää, jos alkuperäinen teksti sisältää runsaasti korvattavia merkkejä.

Lähteet

Automation Anywhere a. What is Robotic Process Automation (RPA)? Artikkeli. Viitattu 14.11.2022. Saatavissa https://www.automationanywhere.com/rpa/robotic-process-automation

Automation Anywhere b. What is OCR? Artikkeli. Viitattu 17.11.2022. Saatavissa https://www.automationanywhere.com/rpa/ocr

Boesch, G. 2022. Example of Optical Character Recognition (OCR). viso.ai. Artikkeli. Viitattu 17.11.2022. Saatavissa https://viso.ai/computer-vision/optical-character-recognition-ocr

Frye, M. What is hyperautomation? MuleSoft. Artikkeli. Viitattu 17.11.2022. Saatavissa https://www.mulesoft.com/resources/api/what-is-hyperautomation?

Gartner. 2021. Gartner Forecasts Worldwide Hyperautomation-Enabling Software Market to Reach Nearly $600 Billion by 2022. Lehdistötiedote. Viitattu 17.11.2022. Saatavissa https://www.gartner.com/en/newsroom/press-releases/2021-04-28-gartner-forecasts-worldwide-hyperautomation-enabling-software-market-to-reach-nearly-600-billion-by-2022

Gartner. 2022. Gartner Says Worldwide RPA Software Spending to Reach $2.9 Billion in 2022. Lehdistötiedote. Viitattu 17.11. 2022. Saatavissa https://www.gartner.com/en/newsroom/press-releases/2022-08-1-rpa-forecast-2022-2q22-press-release

Gartner. Gartner Glossary. Sanasto. Viitattu 21.11.2022. Saatavissa https://www.gartner.com/en/information-technology/glossary/robotic-process-automation-rpa

GlobeNewswire. 2022. Robotic Process Automation Market to Hit USD 23.9 Bn by 2030. Artikkeli. Viitattu 17.11. 2022. Saatavissa https://www.globenewswire.com/news-release/2022/01/06/2362208/0/en/Robotic-Process-Automation-Market-to-Hit-USD-23-9-Bn-by-2030.html

IBM. 2022. What Is Optical Character Recognition (OCR)? Blogi. Viitattu 17.11. 2022. Saatavissa https://www.ibm.com/cloud/blog/optical-character-recognition

MDN Web Docs a. Artikkeli. Viitattu 21.11.2022. Saatavissa https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors

MDN Web Docs b. Artikkeli. Viitattu 21.11.2022. Saatavissa https://developer.mozilla.org/en-US/docs/Web/XPath

MDN Web Docs c. Artikkeli. Viitattu 21.11.2022. Saatavissa https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

Playwright. Verkkosivusto. Viitattu 21.11.2022. Saatavissa https://playwright.dev/

Ray, S., Villa, A., Alexander, M., Guttridge, K., Wang, A. & Vincent, P. 2022. Magic Quadrant for Robotic Process Automation. Gartner. Artikkeli. Viitattu 17.11.2022. Saatavissa https://www.gartner.com/doc/reprints?id=1-2AOVKDE5&ct=220728&st=sb

Robocorp a. A new option for web automation: Using the Robot Framework Browser library, based on Playwright. Artikkeli. Viitattu 21.11.2022. Saatavissa https://robocorp.com/docs/development-guide/browser/playwright

Robocorp b. Dokumentaatio. Viitattu 21.11.2022. Saatavissa https://robocorp.com/docs/libraries/rpa-framework/rpa-browser-playwright

Robot Framework a. Verkkosivusto. Viitattu 14.11. 2022. Saatavissa https://robotframework.org/

Robot Framework b. Dokumentaatio. Viitattu 17.11.2022. Saatavissa https://robotframework.org/robotframework/latest/libraries/BuiltIn.html

Robot Framework c. Dokumentaatio. Viitattu 21.11.2022. Saatavissa https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#different-output-files

yWorks. Verkkosivusto. Viitattu 17.11.2022. Saatavissa https://www.yworks.com/products/yed