Jimi Takamäki

Verkkoharavaohjelman toteutus Pythonilla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

4.4.2024

Tiivistelmä

Tekijä:	Jimi Takamäki
Otsikko:	Verkkoharavaohjelman toteutus Pythonilla
Sivumäärä:	36 sivua + 1 liite
Aika:	4.4.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander Ekonomisti Matti Sipiläinen

Insinöörityössä kehitettiin verkkoharavaohjelma käyttäen Python-ohjelmointikieltä. Ohjelmasta oli tarkoitus luoda mahdollisimman helppokäyttöinen ratkaisu automatisoidun tiedon keräykseen, ja se tulisi toimimaan useilla erilaisilla verkkosivuilla. Ohjelman tulevilla käyttäjillä ei olisi kattavaa ohjelmointiosaamista, joten verkkoharavoiden määrittely haluttiin toteuttaa ohjelmakoodista riippumattomasti.

Ohjelma toteutettiin komentoriville, josta sen käyttö onnistuu yksinkertaisten komentojen avulla. Yksittäisten haravoiden suorituksen lisäksi ohjelma tukee myös suurten haravamäärien ajastamista, mikä mahdollistaa automatisoidun, pitkäkestoisen ja säännöllisen datan keräyksen. Suoritettavat verkkoharavat konfiguroidaan erillisissä YAML-kielisissä konfiguraatiotiedostoissa, joista ohjelma lukee tarvittavat tiedot ajon aikana.

Konfiguraation tukemat asetukset mahdollistavat erilaisilla tavoilla toimivien sivujen haravoinnin. Konfiguroidut verkkoharavat kykenevät esimerkiksi vierittämään sivua alaspäin, siirtymään tuoteluettelosivujen välillä sekä lataamaan sivulle lisää sisältöä painikkeiden kautta. Näiden toiminnallisuuksien avulla tietoja saadaan kerättyä kattavasti erilaisilta sivuilta.

Avainsanat:	verkkoharavointi, automatisoitu tiedonkeräys, Python, Playwright, verkkosivut

---

Abstract

Author:	Jimi Takamäki
Title:	Developing Web Scraping Program with Python
Number of Pages:	36 pages + 1 appendix
Date:	4 April 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer
	Matti Sipiläinen, Economist

The objective of the study was to develop a web scraping program using the Python programming language. The program was intended to be a simple to use solution for automated data collection purposes, that would work on multiple different websites. The future users of the program would not have good programming knowledge, so the configuration of the scrapers needed to be implemented in a way not tied to the program code.

The program was implemented as a command line application that can be used with a few simple commands. Besides running individual scrapers, the program also supports scheduling of scraping tasks. This allows for regular, automated and long-lasting data collection. Individual scrapers are configured in separate configuration files written in the YAML language.

The parameters supported by the configuration allow scraping varying types of websites. The scrapers can scroll down pages, move between product catalogue pages and load more content by clicking buttons. With the help of these varying functionalities diverse kinds of websites can be scraped for desired information.

Keywords: web scraping, data extraction, Python, Playwright

Sisällys

1 Johdanto

Nykymaailmassa yhä suurempi osa tavaroista ja palveluista on saatavilla verkosta. Verkko toimii valtavana tietovarastona mitä erilaisimmille asioille. Kun lähes ilmaiseksi saatavilla olevan tiedon määrä on näin runsasta, syntyy tarpeita hyödyntää tätä tietoa erilaisiin käyttötarkoituksiin.

Käsin tapahtuva tiedon keräys on kuitenkin hidasta ja rajoittaa kerättävän tiedon määrää huomattavasti. Tässä kuvaan astuvat verkkoharavat (engl. web scrapers), jotka mahdollistavat automatisoidun tiedonkeräyksen internetistä ohjelmistojen avulla. Tyypillinen esimerkki verkkoharavoinnin hyödyistä on tuotteiden ja palveluiden hintojen seuranta, johon verkkoharavointi tarjoaa hyvin tehokkaat ja laajamittaiset mahdollisuudet. [1.]

Juuri erilaisten hintainformaatioiden seuranta on merkittävässä osassa työn tilaajan, Kilpailu- ja kuluttajaviraston, toimenkuvaa, mutta sillä ei ole vakiintunutta yhtenäistä menetelmää verkossa tapahtuvan seurannan toteuttamiseksi. Viraston eri yksiköt ovat kehittäneet omia ratkaisujaan seurannan automatisoimiseksi, mutta työ on usein ollut yksittäisiä tarpeita tukevaa, eikä monilla yksiköillä ole resursseja omien ratkaisujen kehittämiseen.

Tältä pohjalta insinöörityön aiheeksi muodostui verkkoharavaohjelma, joka tukee viraston eri yksiköiden tarpeita. Ohjelma mahdollistaa useiden erilaisten verkkoharavoiden mahdollisimman yksinkertaisen konfiguroinnin ja ajastetun suorittamisen. Verkkoharavoiden määrittely haluttiin toteuttaa mahdollisimman yksinkertaisella käyttäjäystävällisellä tavalla, joka olisi ymmärrettävissä myös aikaisemmin verkkoharavointiin perehtymättömille ihmisille.

Työn tilaaja, Kilpailu- ja kuluttajavirasto, on vuonna 2013 Kilpailuviraston ja Kuluttajaviraston yhdistyessä perustettu Työ- ja elinkeinoministeriön alainen virasto, jonka tehtävänä on varmistaa mahdollisimman tehokkaat ja reilut markkinat. Virasto jakautuu kahtia kilpailu- ja kuluttajavastuualueisiin.

Kilpailuvastuualueen tehtävänä on valvoa yritysten välistä kilpailua ja ehkäistä kilpailua rajoittavia menettelyjä. Kuluttajavastuualue puolestaan valvoo kuluttajansuojalainsäädännön toteutumista markkinoilla keskittyen erityisesti markkinointiin ja sopimusehtoihin. [2.]

Työn tavoitteena on toimia edistävänä kokeiluna viraston verkkoharavointitoiminnan yhtenäistämisessä ja tarjota samalla viraston eri yksiköille mahdollisimman helppokäyttöinen työkalu tiedon keräyksen automatisointiin.

2 Verkkoharavointi

2.1 Verkkoharavoinnista yleisesti

Verkkoharavointi tarkoittaa tiedon automatisoitua keräystä verkkosivuilta ohjelmistojen avulla. Ohjelmistorobotit selaavat verkkosivuja käyttäjien tavoin ja keräävät talteen toivottua dataa. Tätä dataa voidaan hyödyntää mitä erilaisimpiin käyttötarkoituksiin, joista yksinkertaisena esimerkkinä toimii hintatietojen seuranta. Verkkoharavoinnin on todettu tarjoavan huomattavasti kattavampaa, tarkempaa ja johdonmukaisempaa tietoa manuaaliseen työskentelyyn verrattuna. [1.]

Verkkoharavointi on tärkeää erottaa verkkorajapintojen (engl. Web API [Application Programming Interface]) käytöstä. Verkkorajapinnat tarjoavat pääsyn kattavaan määrään dataa, mutta ovat täysin palveluntarjoajien hallinnoitavissa. Palveluntarjoajat voivat yksinkertaisesti päättää olla tarjoamatta toivottua dataa rajapintojen kautta, jolloin verkkoharavointi voi olla välttämätöntä halutun datan keräämiseksi.

Verkkoharavointi eroaa verkkorajapintojen käytöstä siinä, mistä tietoa kerätään. Verkkorajapintojen tapauksessa tieto saadaan palveluntarjoajan tarjoaman rajapinnan kautta palveluntarjoajan päättämässä muodossa, josta sitä saadaan tehokkaasti hyödynnettyä ohjelmistojen sisällä. Tiedot ovat usein esimerkiksi

JSON-muodossa (JavaScript Object Notation), jota on helppo käsitellä eri ohjelmointikielien avulla.

Verkkoharavoinnissa tietoa kerätään suoraan verkkosivuilta, yleisimmin HTML-tiedostoista (Hypertext Markup Language). Verkkoharavoinnin avulla lähes kaikki verkkosivuilta löytyvä tieto voidaan haravoida talteen, vaikkei sitä olisi saatavilla palveluntarjoajan verkkorajapinnan kautta. Tämä tekee verkkoharavoinnista monissa tapauksissa välttämättömän menetelmän.

Esimerkiksi monet verkkokaupat eivät tarjoa verkkorajapintaa, jonka kautta voitaisiin saada tietoa myydyistä tuotteista. Monilla tahoilla voi olla mielenkiintona kerätä tuotteiden hintatietoja talteen erilaisia käyttötarkoituksia varten, johon verkkoharavointi voi usein olla ainoa toimiva ratkaisu.

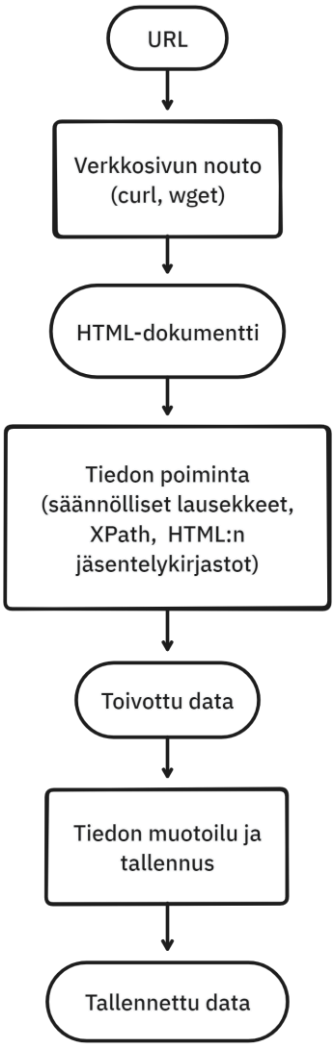
2.2 Verkkoharavoinnin vaiheet

Verkkoharavoinnin ymmärtämisen helpottamiseksi voidaan se jakaa kolmeen erilliseen vaiheeseen. Ensimmäinen vaihe on hakuvaihe (engl. fetching phase), jossa toivottu verkkosivu noudetaan HTTP-protokollan (Hypertext Transfer Protocol) avulla. Tämä on sama tekniikka, jota verkkoselaimet käyttävät verkkosivujen selaamiseen. [3.]

Toisessa vaiheessa toivottu tieto poimitaan ladatusta HTML-tiedostosta. Tyypillisimmät käytetyt teknologiat ovat erilaiset HTML:n jäsentelykirjastot (engl. HTML parsing libraries), säännölliset lausekkeet (engl. regular expression) sekä XPath-kyselyt. XPath on lyhenne englanninkielisistä sanoista XML Path Language ja sitä käytetään tyypillisesti tiedon löytämiseen XML-dokumenteista (Extensible Markup Language). [3.]

Kolmannessa ja viimeisessä vaiheessa kerätty tieto tallennetaan toivotulla tavalla järjestetyssä muodossa [3]. Data voidaan tallentaa esimerkiksi CSV-tiedostoihin (Comma-Separated Values) tai tietokantaan, joista sitä on helppo

hyödyntää myöhemmin toivotulla tavalla. Kuva 1 havainnollistaa edellä mainittuja verkkoharavoinnin vaiheita.



Kuva 1. Verkkoharavoinnin vaiheet. Perustuu Emil Perssonin laatimaan alkuperäiskuvaan. [3.]

2.3 Verkkoharavoinnin haasteet

Perinteisesti verkkoharavointi tapahtuu yksinkertaisimmillaan noutamalla HTML-tiedosto palvelimelta ja poimimalla siitä talteen toivotut tiedot, kuten edellisessä luvussa kuvattiin. Tämä on kohtuullisen yksinkertainen toimenpide toteuttaa eikä vaadi asiaa tuntevalta kehittäjältä suurta määrää töitä.

Nykyään, kun yhä suurempi osa verkkosivuista on rakennettu dynaamisesti, tarvitaan verkkoharavointiin yhä kehittyneempiä työkaluja tiedon keräyksen onnistumiseksi [4]. Dynaamisella verkkosivulla tarkoitetaan sivua, jonka tiedot luetaan tietokannasta ja tulostetaan sivulle jonkin ohjelmointikielen avulla. Tämä eroaa perinteisistä staattisista verkkosivusta siten, että staattisten verkkosivujen tiedot ovat kovakoodattuina verkkosivun HTML-tiedostoihin, ja palvelin lähettää käyttäjän selaimelle nämä tiedostot sellaisinaan. Dynaaminen verkkosivu saattaa esimerkiksi ladata sivulle lisää sisältöä napin painalluksesta, jolloin pelkkä sivun lataaminen HTML-protokollan kautta ei aina riitä tarvittavan tiedon löytämiseksi verkkosivulta. [5.]

Myös laiskan latauksen hyödyntäminen verkkosivukehityksessä vaikeuttaa verkkoharavoinnin toteuttamista. Laiska lataus on verkkosivukehityksessä käytetty menetelmä, jossa sivu lataa sisältöään dynaamisesti tarpeen mukaan mahdollistaen sivujen nopeamman latauksen [6]. Se on hyvin yleinen menetelmä esimerkiksi verkkokauppojen tuoteluetteloiden ja erilaisten uutissivujen toteutuksessa, joilla uutta sisältöä ladataan sivulle sitä mukaa, kun käyttäjä kelaa sivua alas.

Dynaamiset ja laiskaa latausta hyödyntävät verkkosivut monimutkaistavat verkkoharavoinnin toteuttamista huomattavasti. Yksinkertainen sivun lataus HTTP-protokollan kautta ja siitä tietojen talteen poimiminen ei enää ole riittävä ratkaisu tiedon keräämiseen. Kuvaan astuvat työkalut, jotka mahdollistavat vuorovaikutuksen verkkosivujen kanssa normaalin käyttäjän liikkeitä mukaillen. Pythonia tukevista työkaluista tunnetuimpia ovat muun muassa Selenium ja Microsoftin kehittämä Playwright, joista jälkimmäistä hyödynnetään projektin verkkoharavoinnin toteutuksessa. [4.]

Playwrightin avulla laiskaa latausta hyödyntävää dynaamista verkkosivua voidaan vierittää alas, kunnes kaikki toivotut tiedot ovat latautuneet sivun HTML-rakenteeseen. Tämän jälkeen tiedot voidaan kerätä talteen samoja menetelmiä hyödyntäen, joilla tieto poimitaan staattisten sivujen HTML-tiedostoista.

Joissakin tapauksissa dynaamiset sivut saattavat myös poistaa tietoja sivun alkupäästä näkymän siirtyessä sivulla alaspäin, mikä lisää entistä suuremman vaikeusasteen tietojen keräykselle. Tässä työssä kehitetylle ohjelmalle ei kuitenkaan lisätty tukea tämän kaltaisten sivujen haravoimiseen, sillä sitä ei nähty järkeväksi tavoitteeksi tapauksen monimutkaisuus huomioon otettuna.

Bottientunnistus

Verkkoharavointi voi aiheuttaa rasitetta haravoitaville verkkosivuille, ja jotkin verkkosivut pyrkivätkin estämään verkkoharavoita keräämästä tietojaan erilaisin menetelmin. Yleisesti onkin hyvien käytäntöjen mukaista noudattaa verkkosivujen robots.txt-tiedostoissaan ilmoittamia toiveita. Robots.txt-tiedostot ovat verkkosivujen sisältämiä verkkoharavoinnin ohjetiedostoja, joissa verkkosivun ylläpitäjät ilmoittavat sivun säännöistä koskien verkkoharavointia. Tiedosto on yleensä löydettävissä lisäämällä verkkosivun URL-osoitteen (Uniform Resource locator) perään tekstin ”/robots.txt”. [7.]

Yksi keino verkkosivun turhan kuormituksen välttämiseksi on hidastaa verkkoharavointia. Tämä on erityisen tärkeää, jos sivulta kerätään hyvin suuria määriä tietoja. Verkkoharavat kykenevät selaamaan verkkosivuja huomattavasti ihmisiä nopeammin, mikä voi aiheuttaa verkkosivulle rasitetta ja johtaa tilanteeseen, jossa verkkosivu estää verkkoharavan pääsyn sivulle. Pythonissa hidastus onnistuu esimerkiksi time.sleep()-funktiolla, joka keskeyttää suorituksen argumenttina annetun sekunttimäärän ajaksi. [8.]

Toinen tärkeä keino botiksi tunnistautumisen välttämiseksi on muuttaa haravoinnin mukana lähetettyjä HTTP-otsaketietoja (engl. HTTP headers). HTTP-otsaketiedot ovat HTTP-pyyntöjen mukana lähetettyjä avain-arvopareja (engl. key-value pairs), jotka sisältävät pyynnön kannalta oleellista metadataa [9]. Erityisen tärkeä on käyttäjäagenttitiedon (engl. user agent) vaihtaminen, sillä useat verkkoharavointikirjastot kertovat käyttäjäagentin avulla suoraan olevansa robotteja, mikä voi johtaa verkkoharavan estoon. [8.]

3 Toteutus

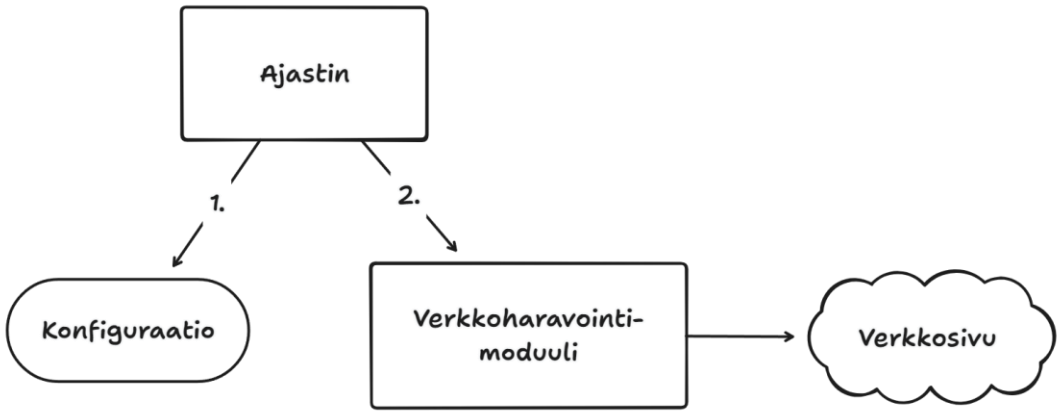
3.1 Yleiskatsaus ohjelmaan

Ohjelma toteutettiin tekstipohjaisena komentoriviohjelmana, joka lukee verkkoharavakohtaiset määritelmät erillisistä YAML-kielisistä konfiguraatiotiedostoista. Komentoriville toteutettu käyttöliittymä tarjoaa yksinkertaisen tavan käyttää eri ominaisuuksia, minkä lisäksi sen kehitys ja ylläpito onnistuvat helposti.

Rakenteellisesti ohjelma jaettiin toiminnallisuuksiltaan toisistaan erillisiin osiin eli moduuleihin. Näistä merkittävimmät ovat ajastin, verkkoharavointimoduuli sekä konfiguraatioiden käsittelijä. Ohjelman jako erillisiksi toisistaan riippumattomiksi osiksi tukee hyvien ohjelmistosuunnittelun periaatteiden mukaista modulaarisuutta. Tämä mahdollistaa eri osien toisistaan riippumattoman käytön esimerkiksi tilanteessa, jossa verkkoharavoita haluttaisiin suorittaa ilman ajastustoiminnallisuutta. Näin ohjelman osat ovat helpommin uusiokäytettävissä, ja niiden testaus onnistuu helpommin mahdollisimman lievillä sivuvaikutuksia.

Ajastin mahdollistaa verkkoharavoiden automatisoidun ja säännöllisen suorittamisen. Se suorittaa listalla olevat verkkoharavat niiden konfiguraatioissa määritettyinä ajankohtina kutsumalla verkkoharavointimoduulia määritellyillä asetuksilla. Verkkoharavointimoduuli huolehtii varsinaisen verkkoharavoinnin toteutuksesta. Ohjelma tukee myös yksittäisten verkkoharavoiden suorittamista ilman ajastamista, jolloin ajastinmoduulia ei käytetä.

Kuva 2 havainnollistaa ohjelman rakennetta tilanteessa, jossa verkkoharavoita ajastetaan ajastinmoduulin avulla. Ohjelman käyttäjän tulee muokata vain konfiguraatiotiedostoja, joista ajastin lukee ohjeet verkkoharavoinnin toteuttamiseksi. Verkkoharavointimoduuli huolehtii kanssakäymisestä verkkosivujen kanssa.



Kuva 2. Ohjelman perusrakenteen visualisointi vuokaavion muodossa.

Konfiguraatiotiedostot rakennetaan kohdesivun lähdekoodin perusteella. Jokaista uutta sivua varten tarvitaan oma konfiguraatio, joka perustuu kyseisen sivun HTML-rakenteeseen.

Kehitysprosessin edetessä ohjelmalle päätettiin antaa nimeksi Leaves viitaten kasvien pudonneisiin lehtiin, joita normaaleilla haravoilla siivotaan pois pihoilta. Leaves toimii ytimekkäänä nimenä, joka on helppo kirjoittaa komentoriville sovellusta käytettäessä.

Toteutuksen ohjelmointikieleksi valikoitui Python sen suuren suosion, monipuolisuuden ja helppokäyttöisyyden vuoksi. Pythonin hyvin aktiivisen käyttäjäkunnan ansiosta netistä on löydettävissä valtava määrä tietoa ja ohjeita sen käytöstä. Pythonille on myös kehitetty valtava määrä erilaisia kirjastoja, joita on helmikuussa 2024 julkaistu PyPIssä (The Python Package Index) jo yli 500 000 [10]. [11.]

Kehitys tapahtui Microsoftin tarjoamassa Visual Studio Code -kehitysympäristössä, joka tarjoaa monipuolisen, mukautettavan ja modernin kehityskokemuksen. Yhteisön kehittämät lisäosat tekevät Visual Studio Coden käytöstä hyvin sujuvaa moniin eri tarkoituksiin. [12.]

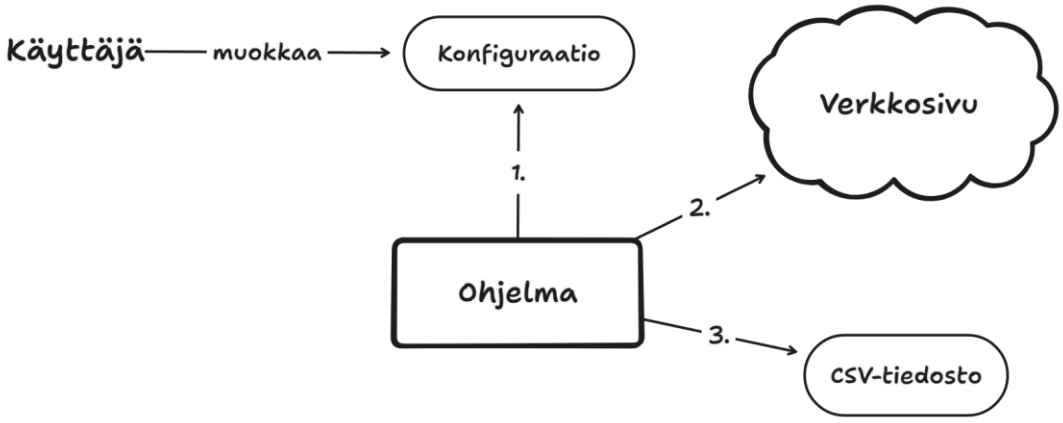
3.2 Konfiguraatio

Ohjelman tavoitteena on toimia helppona ratkaisuna erilaisiin yksinkertaisiin haravointihankkeisiin, eikä uusien haravoiden määrittelyn tulisi täten olla vaikeaa tai hidasta. Ohjelman tulevat käyttäjät tulevat olemaan teknisesti osaavia henkilöitä, mutta heiltä ei voida olettaa kattavaa ohjelmoinnin ja verkkoharavoinnin osaamista. Verkkoharavoiden määrittelyyn tarvittiin siis yksinkertainen ohjelmointitaidoista riippumaton tapa, jota olisi myös tehokasta käyttää.

Niinpä haravoiden määrittely päätettiin toteuttaa konfiguraatiotiedostojen avulla. Kullekin haravalle luodaan oma konfiguraatiotiedosto, johon määritellään sen asetukset, kuten haravan suoritusajankohdat, kerättävät tiedot ja verkkosivuilla suoritettavat toimenpiteet.

Kuva 3 havainnollistaa yksinkertaistetusti verkkoharavaohjelman toimintarakennetta konfiguraation kannalta. Käyttäjän ei tarvitse muokata varsinaista ohjelmaa haravoinnin toteuttamiseksi – konfiguraation muokkaus riittää. Ohjelma suorittaa tietojen keräyksen ja tallennuksen konfiguraatioissa määritetyllä tavalla.

Kuvan vaiheessa 1 ohjelma lukee konfiguraatiotiedostoista verkkoharavoiden asetukset. Kun konfiguraatiossa määritelty haravointiaika koittaa, haravoi ohjelma määritellyltä verkkosivulta toivotut tiedot talteen vaiheen 2 mukaisesti. Lopulta kolmannessa vaiheessa ohjelma tallentaa kerätyt tiedot CSV-tiedostoon, joka tallennetaan samaan kansioon haravan konfiguraation kanssa.



Kuva 3. Vuokaavio ohjelman toiminnasta.

Suunnittelun alkuvaiheilla konfiguraation muodoksi kaavailtiin JSON-tiedostoa. JSON mahdollistaisi helposti ymmärrettävän ja tehokkaan tavan verkkoharavoiden konfigurointiin avain-arvoparien avulla. JSON on myös monille vähänkään enemmän tietotekniikkaan perehtyneille hyvin tuttu tiedon tallennusmuoto, joka ei vaatisi erillistä perehtymistä.

Projektin toteutuksen aloitusvaiheessa päädyttiin kuitenkin käyttämään YAML-kieltä konfiguraation toteuttamiseksi. YAML on JSONin kaltainen merkintäkieli, jonka yhtenä tavoitteena on mahdollisimman helppo luettavuus ihmisille [13]. YAML käyttää Pythonin kaltaisesti sisennystä tietueiden ryhmittelyyn, eikä se sisällä monia JSONin sisältämiä erikoismerkkejä, mikä tekee sen syntaksista helppolukuista ja lyhyttä.

Esimerkkikoodit 1 ja 2 havainnollistavat JSON- ja YAML-kielten eroja. Koodikatkelmat sisältävät saman tietorakenteen esitettynä kummallakin kielellä. YAML-kieli vastaa huomattavasti enemmän tapaa, jolla tietorakenteita saatettaisiin kirjoittaa käsin paperille, mikä tekee siitä helpommin lähestyttävän ja luettavan vaihtoehdon.

```
[
    {
        "nimi": "Pekka",
        "ikä": 25,
        "harrastukset": [
            "maalaus",
            "piirtäminen"
        ],
        "lemmikit": {
            "kissa": {
                "nimi": "Erik",
                "rotu": "maatiaiskissa",
                "väri": "harmaa"
            },
            "papukaija": {
                "nimi": "Ester",
                "laji": "aurinkoaratti"
            }
        },
        "lempiruuat": {
            "hedelmät": [
                "banaani",
                "mango"
            ]
        }
    }
]
```

Esimerkkikoodi 1. Esimerkki JSON-kielestä.

```
nimi: Pekka
ikä: 25
kotimaa: Suomi
harrastukset:
  - maalaus
  - piirtäminen
lemmikit:
  kissa:
    nimi: Erik
    rotu: maatiaiskissa
    väri: harmaa
  papukaija:
    nimi: Ester
    laji: aurinkoaratti
lempiruuat:
  hedelmät:
    - banaani
    - mango
```

Esimerkkikoodi 2. Esimerkki YAML-kielestä.

Konfiguraation lopullinen muoto on havainnollistettu esimerkkikoodissa 3. Konfiguraatiossa haravalle määritetään muun muassa sen nimi, suorituskellonaika, suorituskertojen tiheys päivissä, haravoitava URL-osoite sekä ohjeet kerättävien tietojen paikantamiseksi verkkosivuilta. Konfiguraation avaimet ovat ohjelman

kehityksessä itse määritettyjä termejä, joiden mukaan tiedot luetaan konfiguraatiosta ohjelman lähdekoodissa. Kaikki niistä eivät siis ole yleisesti tunnettuja termejä.

Verkkosivukohtaiset asetukset määritellään sivun lähdekoodin perusteella, eikä ohjelma itse tiedä, mitä se sivulta kerää. Ohjelma noudattaa tarkoin konfiguraatiossa määriteltyjä asetuksia ja hoitaa tiedon keräyksen niiden pohjalta. Ohjelma ei tiedä haravoitavasta verkkosivusta mitään etukäteen, joten käyttäjän tulee itse huolehtia, että ohjelma osaa paikantaa oikeat tiedot verkkosivulta.

Osa avain-arvopareista ovat pakollisia, eikä käyttäjä voi jättää niitä määrittelemättä. Pakollisten avain-arvoparien lisäksi konfiguraatio tukee myös vapaaehtoisia asetuksia. Näitä ovat asetukset, joita ei tarvita kaikkien verkkosivujen haravoimiseksi, mutta joita voidaan käyttää tarvittaessa. Näitä ovat esimerkiksi verkkosivun evästeiden hyväksymisnapin painamiseen käytetty asetus sekä tuoteluetteloiden haravointiin käytetyt asetukset. Vapaaehtoiset asetukset käydään tarkemmin läpi työn myöhemmissä kohdissa, joissa käsitellään kyseisiin asetuksiin liittyviä ohjelman ominaisuuksia.

```
scraper_name: Harava1
schedule_time: 09:28
run_every_x_days: 1
url: https://kuvitteellinen-sivu1.fi
selectors:
  name: h1.tuotteen-nimi
  price: p.tuotteen-hinta
cookie_selector: button.accept-all
catalog: true
lazy: true
parent_selector: div.product-container
pagination_next_selector: span.next-page
```

Esimerkkikoodi 3. Projektin konfiguraation lopullinen muoto. Esimerkissä on määritelty kuvitteellinen verkkoharava, joka käy määritellyssä URL-osoitteessa keräämässä tietoja joka päivä kello 9.28.

Selectors-avaimen alle listataan CSS-valitsimet (engl. CSS selectors) (Cascading Style Sheets), joiden avulla toivotut tiedot paikannetaan verkkosivulta. Listattujen valitsimien avaimet toimivat sarakkeiden otsikoina lopullisessa CSV-

tiedostossa, johon kerätty data tallennetaan. Käyttäjät määrittelevät CSS-valitsimet itse haravoitavan sivun lähdekoodin perusteella.

CSS-valitsimet ovat CSS-kielessä käytetty keino verkkosivuelementtien paikantamiseksi. Niitä käytetään yleensä tyylien määrittelyssä HTML-elementeille. Valitsimet perustuvat HTML-elementtien tyyppeihin, attribuutteihin, tiloihin sekä niiden sijainteihin verkkosivun HTML-rakenteessa. [14.]

Esimerkkikoodi 4 sisältää esimerkin CSS-valitsimesta. Valitsimen ensimmäinen määrite article.info valitsee article-elementin, jolle on määritetty luokaksi (engl. class) info. Merkki > valitsee merkkiä edeltävän elementin sisältä yhden tason alemman elementin, jota merkin oikealle puolelle kirjoitettu määrite vastaa. Määrite ":nth-child(n)" valitsee n:nnen kaksoispisteen edelle kirjoitetun elementin edeltävän elementin sisältä, eli tässä tapauksessa kuudennen section-elementin article-elementin sisältä. Viimeinen määrite p valitsee kaikki p-elementit edeltävän elementin sisältä. [15.]

```
article.info > section:nth-child(6) > p
```

Esimerkkikoodi 4. Esimerkki CSS-valitsimesta.

Tyylien määrittämisen lisäksi CSS-valitsimia voidaan käyttää hyödyksi myös esimerkiksi verkkoharavoinnissa oikeiden haravoitavien elementtien paikantamiseksi. Monet HTML:n jäsentelykirjastot tukevat CSS-valitsimien käyttöä elementtien valitsemisessa.

Vaikka CSS-valitsimien toiminta ei ole asiaan perehtymättömälle välttämättä kovinkaan itsestään selvää, ovat ne silti tehokas keino verkkosivuelementtien paikantamiseksi sivuilta ja mahdollisesti hyvinkin tuttuja verkkosivukehitykseen perehtyneille. Kehitetyn projektin dokumentaatioon kirjoitettiin kattava ohjeistus yleisimpiin käyttötapauksiin – miten CSS-valitsimet toimivat, miten verkkosivuelementin CSS-valitsimen saa selville sekä miten toimia tilanteissa, joissa määritetty valitsin palauttaakin sivulta vääränlaista tietoa. Ohjeistuksen ansiosta haravoiden konfiguroinnin tulisi onnistua myös asiaan aikaisemmin perehtymättömiltä käyttäjiltä.

3.3 Ajastin

Ohjelmasta haluttiin automatisoitu ratkaisu erilaisiin datan keräyksen tarpeisiin. Jotta kerätystä datasta olisi merkittävää hyötyä, täytyy sitä kerätä pitkältä aikaväliltä. Niinpä säännöllinen datan keräys haluttiin automatisoida ja tätä tarkoitusta varten päätettiin kehittää erillinen ajastinmoduuli.

Ajastin päätettiin rakentaa erilliseksi moduulikseen selkeyden ja hyvien ohjelmistokehityksen periaatteiden varmistamiseksi. Näin ajastin ei ole riippuvainen muiden moduulien toteutuksesta, mikä mahdollistaa sen tehokkaamman uusiokäytön ja helpomman testauksen. Tämä mahdollistaa myös yksittäisten ajastamattomien haravointitoimenpiteiden suorittamisen ohjelmalla, sillä ajastin voidaan sivuuttaa suorituksessa ja kutsua haravointimoduulia suoraan toivotuilla parametreilla.

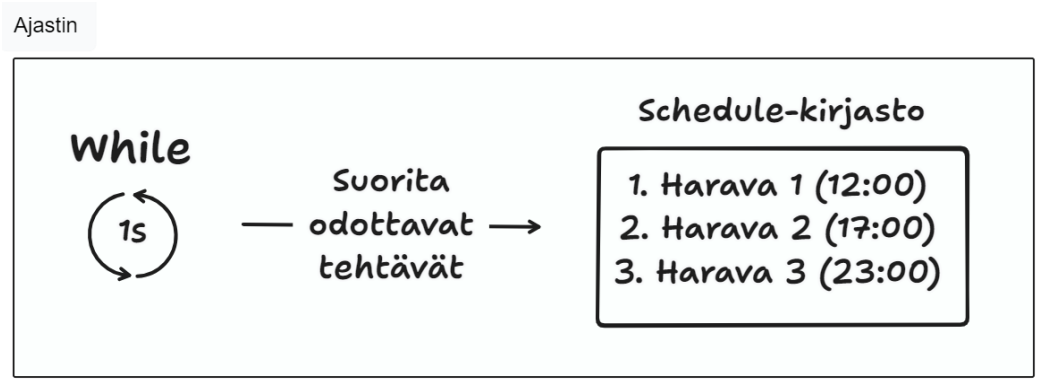
Ajastin kutsuu erillisen konfiguraatioiden käsittelyyn kehitetyn moduulin funktiota, joka lukee määritellyn kansion sisältämistä konfiguraatiotiedostoista verkkoharavakohtaiset asetukset ja palauttaa tiedot listana, joka sisältää Pythonin dataclass-muotoisia olioita. YAML-kielisen konfiguraatiotiedoston lataus tapahtuu PyYAML-kirjaston avulla, joka muuttaa tiedot Pythonin tarjoamaan sanakirjarakenteeseen (engl. dictionary).

Dataclass-luokat mahdollistavat selkeän ja helppokäyttöisen väliaikaisen avain-arvoparien tallennusmuodon, josta niitä voidaan uusiokäyttää tehokkaasti eri tarkoituksiin ohjelman sisällä. Pythonin versiossa 3.7 esitellyt dataclass-luokat tarjoavat normaaleja luokkia lyhyemmän luokkien määrittelytavan, ja ne on suunniteltu tiedon väliaikaista tallennusta varten. Käytännössä dataclass-luokat eivät kuitenkaan poikkea kirjoitusasuaan enempää Pythonin normaaleista luokista, sillä ne toimivat vain niiden kuorruttajina (engl. decorator). [16.]

Konfiguraation lukemisen jälkeen ajastin käyttää Schedule-nimistä avoimen lähdekoodin Python-kirjastoa verkkoharavoiden ajastamiseen. Haravoiden määritetyllä suoritushetkellä Schedule kutsuu tässä työssä kehitettyä erillistä verkkoharavointimoduulia dataclass-olioihin tallennetuilla tiedoilla.

Verkkoharavointimoduuli huolehtii varsinaisen verkkoharavoinnin toteuttamisesta ja tallentaa kerätyt tiedot CSV-tiedostoon samaan kansioon konfiguraatiotiedoston kanssa.

Kuva 4 havainnollistaa ajastimen sisäistä toimintaa piirroksen avulla. Ajastimen sisällä pyörii Pythonin while-toistorakenne, joka toistetaan sekunnin välein. Suoritushetkellä toistorakenne kutsuu schedule-kirjaston run_pending-funktiota, joka huolehtii jonossa odottavien haravoiden suorituksesta. Harava suoritetaan, jos sille määritelty kellonaika on koittanut.



Kuva 4. Visualisointi ajastimen toiminnasta.

Ajastimesta haluttiin rakentaa mahdollisimman vakaa. Ajastin ei saa kaatua esimerkiksi tilanteessa, jossa yksittäisen verkkoharavan konfiguraatio sisältää ohjelman toimintaan vaikuttavan kirjoitusvirheen. Ajastimen on tarkoitus toimia itsenäisesti pitkiäkin aikoja ja huolehtia sen hallinnoimien verkkoharavoiden suorituksesta.

Vakauden edesauttamiseksi ajastimen kehityksessä käytettiin runsaasti hyödyksi Pythonin poikkeustenhallintaa (try/except). Poikkeustenhallinnan avulla mahdolliset poikkeukset voidaan käsitellä hallitusti ja voidaan varmistua siitä, ettei ohjelma kaadu yllättäen.

3.4 Verkkoharavointimoduuli

Varsinainen verkkoharavointi haluttiin erottaa ajastimesta erilliseksi moduulikseen, jotta projektin rakenne pysyisi selkeänä ja jotta verkkoharavointitoimenpiteitä voitaisiin hyödyntää myös ilman ajastinta. Verkkoharavointimoduulin riippumattomuus ajastimesta mahdollistaa myös huomattavasti helpomman ohjelman testauksen, kun verkkoharavointimoduulin funktioita voidaan kutsua ilman ajastimen käynnistämistä. Näin verkkoharavoita voidaan suorittaa välittömästi ilman ajastamista ja toistuvaa konfiguraation muokkausta.

Koska ohjelman haluttiin pystyvän haravoimaan verkkokauppojen tuoteluetteloita ja toimimaan dynaamisilla verkkosivuilla, jotka vaativat käyttäjältä erilaisia toimenpiteitä, päätettiin projektissa hyödyntää Microsoftin kehittämää avoimen lähdekoodin verkkosivutestaukseen suunniteltua kirjastoa Playwrightia. Playwright mahdollistaa vuorovaikutuksen verkkosivujen kanssa käyttäjän tavoin tarjoamalla toiminnallisuuksia muun muassa painikkeiden klikkailuun, sivun vierittämiseen ja tekstikenttien täyttämiseen.

Esimerkkikoodi 5 havainnollistaa Playwrightin käyttöä yksinkertaisella esimerkillä. Esimerkissä Playwright käynnistää Chromium-selaimen, jossa avataan Playwrightin kotisivu. Sivun avaamisen jälkeen Playwright klikkaa kuvitteelliseen selector-muuttujaan tallennetun CSS-valitsimen mukaista elementtiä. Lopulta kyseisellä hetkellä selaimessa auki olevan sivun HTML-rakenne tallennetaan html-nimiseen muuttujaan, josta sitä voitaisiin hyödyntää myöhemmissä vaiheissa. Verkkoharavointimoduuli pohjautuu perusidealtaan tähän esimerkkiin, jota laajentamalla voidaan luoda kyvykäs erilaisia toiminnallisuuksia tukeva verkkoharava.

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch()
    page = browser.new_page()
    page.goto("http://playwright.dev")
    page.locator("css=" + selector).click()
    html = page.content()
    browser.close()
```

Esimerkkikoodi 5. Yksinkertainen esimerkki Playwright-kirjaston käytöstä, jossa klikataan määriteltyä verkkosivuelementtiä ja tallennetaan sivun HTML-rakenne html-nimiseen muuttujaan.

Vaikka Playwright mahdollistaisi myös tietojen talteen poimimisen sivulta, ei se ole tehokkain tai helpoin työkalu kyseiseen tehtävään. Perinteisemmät HTML:n jäsentelytyökalut hoitavat tehtävän nopeasti sekä helposti ymmärrettävällä tavalla, mikä johti päätökseen niiden käytöstä projektissa. Perinteisemmän HTML:n jäsentelytyökalun käyttö erottaa myös sivun kanssa tapahtuvan vuorovaikutuksen ja siltä tietojen keräyksen tehokkaasti toisistaan, mikä takaa näiden toimenpiteiden mahdollisimman laajan riippumattomuuden toisistaan.

Niinpä toivottujen tietojen keräämiseen HTML-rakenteesta käytetään avoimen lähdekoodin Selectolax-HTML-jäsennintä. Kun Playwright on hoitanut vuorovaikutustoimenpiteet sivun kanssa ja toivottu HTML-rakenne on latautunut sivulle, kerätään se talteen Playwrightin tarjoamalla content-funktiolla (kuten esimerkkikoodissa 5) ja syötetään erilliseen itse kehitettyyn funktioon jäsentelyä varten. Selectolax lukee syötetyn HTML-rakenteen ja poimii siitä talteen toivotut tiedot konfiguraatiossa määriteltyjen CSS-valitsimien avulla.

Tuoteluetteloiden haravointi

Yksinkertaisimmillaan työkalua käytetään yksittäisten tietojen keräämiseen verkkosivuilta, josta esimerkkinä toimii jonkin tietyn tuotteen hinnan päivittäinen haravointi. Tämä muuttuu kuitenkin hyvin nopeasti työlääksi prosessiksi, jos työkalulla halutaan kerätä useiden samankaltaisten tuotteiden hintatietoja samalta verkkosivulta. Jokaista tuotetta varten tulisi konfiguroida oma erillinen verkkoharava ja ohjelman hallinnoitavissa olisi suuri määrä haravoita, jotka ovat lähes identtisiä kopioita toisistaan.

Tästä syystä ohjelmaan haluttiin kehittää tuki tuoteluetteloiden haravoinnille. Tällä tarkoitetaan useiden samanlaisten tietueiden keräämistä yhdeltä verkkosivulta samojen CSS-valitsimien avulla – esimerkkinä tuotteiden nimien ja hintatietojen haravointi jonkin verkkokaupan tuoteluettelosta.

Tuen lisäys haluttiin suunnitella konfiguraation kannalta huolellisesti, sillä konfiguraation tuli pysyä helposti ymmärrettävänä ja käyttökelpoisena. Huolimaton uusien avain-arvoparien lisäys konfiguraatioon johtaisi helposti sen sekavoitumiseen, eivätkä tulevat käyttäjät kokisi sen käyttöä kovinkaan intuitiiviseksi.

Ensimmäisenä konfiguraatioon lisättiin tuki uudelle avaimelle nimeltä catalog. Catalog on totuusarvomuuttuja, joka määrittää, käytetäänkö kyseisellä sivulla tuoteluetteloiden haravointitoiminnallisuutta. Käyttäjä voi asettaa catalog-muuttujan arvoksi joko arvon true tai false.

Jos catalog-muuttujan arvo on asetettu todeksi, tulee käyttäjän määritellä arvo myös toiselle konfiguraatioon lisätylle arvolle nimeltä parent_selector. Parent_selector määrittää CSS-valitsimen HTML-elementeille, joista jokainen sisältää selectors-listassa määritellyt kerättävät tiedot. Usein tämä on esimerkiksi tuotekortti, joka sisältää tuotteen nimen, kuvan ja hinnan.

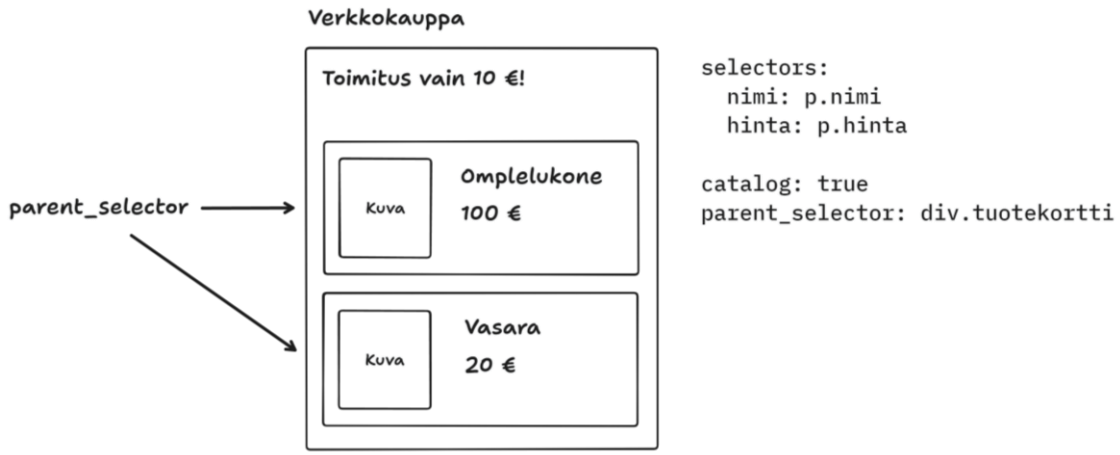
Terminä parent selector on itse keksitty. Sen ensimmäinen osa parent viittaa englanniksi jotakin elementtiä ylemmällä tasolla olevasta HTML-elementeistä käytettävään termiin parent. Jälkimmäinen osa selector tulee CSS-valitsimen englanninkielisestä nimestä CSS selector.

Parent_selectorilla pyritään varmistamaan, että kerätyt tiedot ryhmitellään oikein. Esimerkiksi jos verkkokaupan sivulta kerätään sadan tuotteen nimet ja hinnat, miten voidaan varmistua, että nimet ja hinnat viittaavat toisiinsa oikein, eivätkä mene keskenään sekaisin? CSS-valitsimet saattavat myös viitata arvaamattomasti elementteihin, joita ei haluta haravoida ja jotka sekoittaisivat kerätyn datan järjestystä.

Kuva 5 ja esimerkkikoodi 6 havainnollistavat parent_selector-muuttujan merkitystä. Kuvassa kuvitteellisen verkkokaupan sivulla on ilmoitettu tuotteiden lisäksi myös yleinen toimituskulu. Esimerkkikoodista voidaan todeta, että toimituskulun HTML-elementti on täysin samanlainen kuin tuotteiden hintojen HTML-elementit. Kuvan esimerkkikonfiguraation CSS-valitsin p.hinta viittaa mihin tahansa p-elementtiin, jolle on asetettu luokaksi ”hinta”. Esimerkkikoodissa näitä elementtejä on kolme, joista ensimmäinen on toimituskulu, mutta vain tuotekorttien sisältä löytyvät tuotteiden hinnat halutaan kerätä.

Jos tietoja kerättäisiin sivulta pelkässä ilmestymisjärjestyksessä ilman parent_selector-muuttujaa, voisi ompelukoneen hinnaksi yllättäen tulla ”Toimitus vain 10 €!” ja vasaran hinnaksi ”100 €”. Vasaran oikea hinta 20 euroa saattaisi jäädä CSV-tiedostossa jonkin nimeämättömän tuotteen hinnaksi.

Kuvassa parent_selector-muuttujan arvoksi asetettu div.tuotekortti viittaisi HTML-rakenteessa kahteen div-elementtiin (division), joiden luokiksi on asetettu arvo ”tuotekortti”. Teknisesti ajateltuna verkkoharava kerää ensin parent_selectoria vastaavat elementit sivun HTML-rakenteesta, ja siirtyy seuraavaksi keräämään selectors-listassa määriteltyjä arvoja löydettyjen elementtien sisältä. Näin esimerkin toimituskulu jää pois kerätyistä tiedoista, koska se ei ole parent_selectoria vastaavan div-elementin sisällä, ja ompelukone ja vasara saavat pitää todelliset hintansa.



Kuva 5. Visualisointi parent_selector-muuttujan merkityksestä.

```
<div>
    <p class="hinta">Toimitus vain 10 €!</p>
    <div class="tuotekortti">
        <p class="nimi">Ompelukone</p>
        <p class="hinta">100 €</p>
    </div>
    <div class="tuotekortti">
        <p class="nimi">Vasara</p>
        <p class="hinta">20 €</p>
    </div>
</div>
```

Esimerkkikoodi 6. Esimerkki kuvattuna HTML-rakenteen avulla.

Verkkokauppojen tuoteluettelot on usein jaettu useille numeroiduille sivuille, joiden välillä pääsee liikkumaan sivunumeroinnin yhteydestä löytyvien nuolipainikkeiden kautta. Verkkoharavaohjelman haluttiin kykenevän keräämään talteen kaikki luettelon sisältämät tuotteet. Tämä mahdollistettiin lisäämällä konfiguraatioon tuki CSS-valitsimelle, jolla sivunumeroinnin yhteydestä löytyvät nuolipainikkeet saadaan paikannettua.

Useiden tuotteiden tietojen tallentaminen samaan CSV-tiedostoon johti myös tilanteeseen, jossa CSV-tiedostojen muotoilua tuli päivittää. Alkuperäinen muotoilu oli hyvin yksinkertainen: jokaiselle riville lisättiin kunkin haravointikerran päivämäärä ja kerätyt tiedot niitä vastaaviin sarakkeisiin. Uusien haravointikertojen yhteydessä tiedot lisättiin uudelle riville tiedoston loppuun.

Suunnittelun jälkeen muotoilu päädyttiin pitämään pohjimmiltaan samana. Ainoana erona muotoilussa on se, että uudessa versiossa useiden tuotteiden tiedot tallennetaan samaan CSV-tiedostoon. CSV-tiedoston ensimmäinen rivi sisältää sarakkeiden otsikot ja loput rivit sisältävät kerättyjä tietoja. Eri tuotteiden tiedot lisätään suoraan toistensa alle samoihin sarakkeisiin, mikä saattaa kuulostaa sekavalta lähestymistavalta, mutta toimii useimpiin tarkoituksiin todella hyvin. Erilaiset tietokoneohjelmat kykenevät lukemaan tätä muotoilua tehokkaasti. Myös ihmiset pystyvät tarkastelemaan yksittäisten tuotteiden hintahistoriaa esimerkiksi Excelin avulla, jossa sarakkeiden sisällön voi rajata vain yhteen tuotteeseen. Lopullinen muotoilu löytyy taulukosta 1.

Taulukko 1. Uusi CSV-tiedoston muotoilu.

Päivämäärä	**Tuotteen nimi**	**Tuotteen hinta**
2024-02-04	Vasara	20 €
2024-02-04	Ompelukone	100 €
2024-02-05	Vasara	19 €
2024-02-05	Ompelukone	110 €

3.5 Komentorivin käyttöliittymä

Ohjelmalle tarvittiin käyttöliittymä, jonka avulla käyttäjät kykenisivät käyttämään sen eri toiminnallisuuksia. Graafisen käyttöliittymän rakentaminen koettiin turhan monimutkaiseksi prosessiksi sen tarjoamiin hyötyihin nähden. Verkkoharavat konfiguroidaan käyttöliittymän ulkopuolella, eikä sovellus tue kuin muutamaa toimintoa.

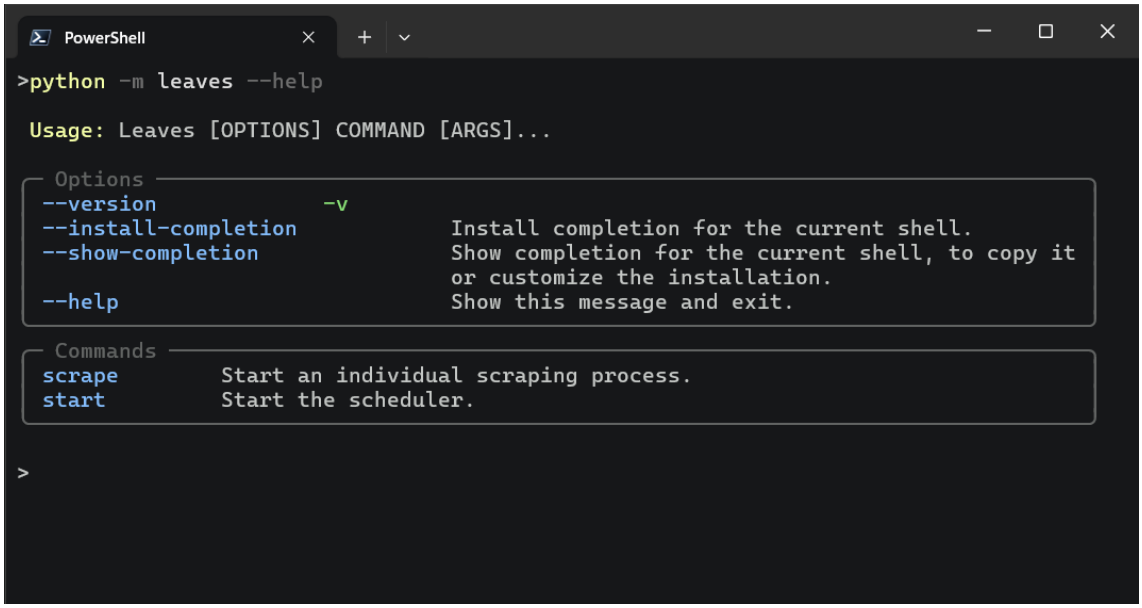
Käyttöliittymä päätettiin täten rakentaa komentoriville. Komentorivin käyttöliittymä on nopea ja helppo kehittää, minkä lisäksi se skaalautuu hyvin tulevaisuuden tarpeisiin, sillä uusia toimintoja voidaan lisätä käyttöliittymän kannalta todella yksinkertaisesti.

Käyttöliittymä tukee kahta komentoa: start ja scrape. Start-komento käynnistää ajastimen ja scrape-komento suorittaa yksittäisiä verkkoharavoita ilman ajastamista. Lisäksi ohjelma esimerkiksi tulostaa versionsa asetuksella ”--version" ja antaa käyttöohjeita asetuksella ”--help".

Ajastinta käytettäessä ohjelma olettaa verkkoharavakonfiguraatioiden sijaitsevan moduulin sisältämässä configs-kansiossa. Halutessaan käyttäjä voi myös määrittää konfiguraatiokansion itse lisäämällä komennon perään kansion tiedostopolun. Yksittäisiä haravoita suorittaessa käyttäjän tulee antaa tiedostopolku tiettyyn YAML-kieliseen konfiguraatiotiedostoon komennon yhteydessä.

Pythonin standardikirjasto (engl. standard library) tarjoaa argparse-nimisen kirjaston komentorivin käyttöliittymien rakentamiseen. Argparse ei kuitenkaan ole helpoin tai monipuolisin kirjasto tähän tarkoitukseen, ja lyhyen suunnittelun jälkeen käyttöliittymä päätettiinkin rakentaa käyttämällä avoimen lähdekoodin Typer-kirjastoa.

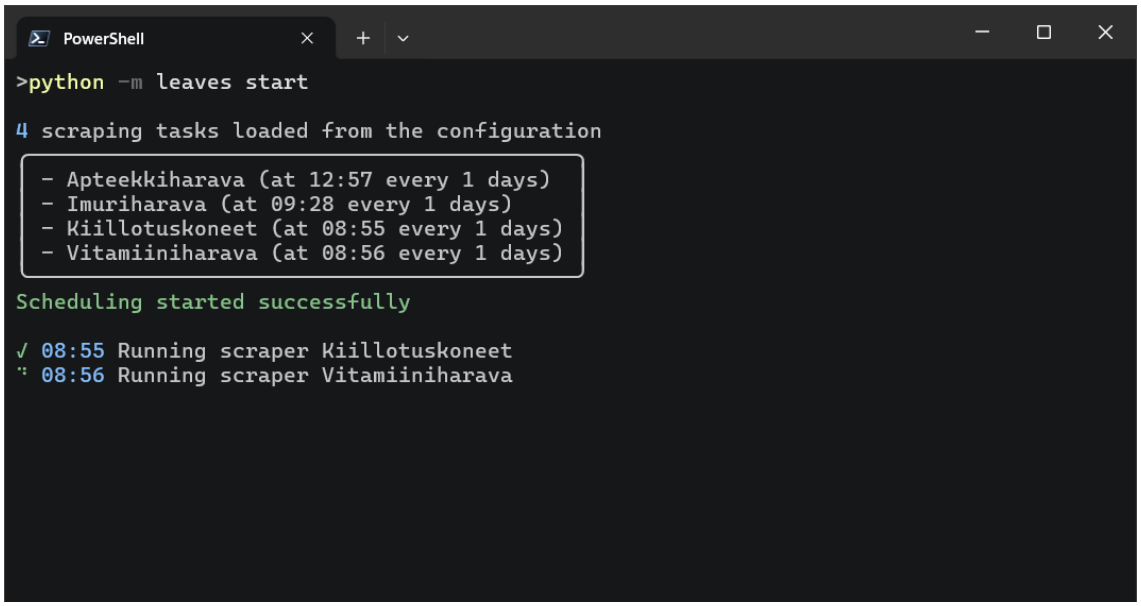
Typerin syntaksin on tarkoitus olla lyhyttä ja ymmärrettävää. Typerin avulla rakennetut ohjelmat sisältävät automaattisesti useita komentorivin käyttöliittymien perusominaisuuksia. Typer esimerkiksi muodostaa help-komennon tulostamat avustavat käyttöohjeviestit automaattisesti perustuen muun muassa koodiin kirjoitettuihin tyyppivihjeisiin (engl. type annotations). Halutessaan käyttäjä voi myös lisätä tarkentavat kuvaukset ohjeisiin. Typerin automaattisesti luoma käyttöohjeviesti on havainnollistettuna kuvassa 6. [17.]



Kuva 6. Typer-kirjaston automaattisesti luoma käyttöohje.

Typer asentaa mukanaan avoimen lähdekoodin rich-kirjaston, jota käytetään komentorivin tulosteiden muotoiluun. Typer itse käyttää tätä omien tulosteidensa, esimerkiksi käyttöohjetulosteen sekä virhetulosteiden, muotoiluun. [17.]

Rich-kirjaston avulla Leavesin käyttöliittymään luotiin värikkäät ja visuaalisesti miellyttävät tulosteet. Ajastinta käytettäessä käyttöliittymä tulostaa listan konfiguraatiotiedostoista ladatuista verkkoharavoista, jotka rajattiin visuaalisesti kehyksen sisään. Rich tarjoaa myös muun muassa pyörivän latausanimaation, jota käytetään ohjelmassa verkkoharavoinnin ollessa käynnissä. Leavesin ajastimeen liittyvät tulosteet ovat havainnollistettuna kuvassa 7.



Kuva 7. Leavesin ajastimeen liittyvät tulosteet.

3.6 Ohjelman yksikkötestaus ja lokitoiminnallisuus

Yksikkötestaus

Verkkoharavaohjelman on tärkeää olla vakaa ja luotettava. Ohjelma tulee olemaan käynnissä hyvin pitkiä aikoja ilman kattavaa ihmisvoimin toteutettavaa valvontaa, eikä se saa kaatua odottamattomasti virheiden seurauksena. Toivotun vakauden edesauttamiseksi projektissa käytetään hyödyksi yksikkötestausta, jonka avulla ohjelman eri moduulien yksittäisten toimintojen luotettavuus voidaan varmistaa.

Ohjelman eri moduulien yksikkötestaukseen käytetään Pythonin standardikirjaston tarjoamaa unittest-kirjastoa. Valinnassa painottuivat aikaisempi kokemus

unittest-kirjaston käytöstä sekä sen käyttöönoton helppous sen sisältyessä Pythonin standardikirjastoon, joka ei vaadi erillistä asennusta.

Esimerkkikoodi 7 havainnollistaa unittest-kirjaston käyttöä. Testattava funktio (validate_conf_values) on kehitetty varmistamaan, että Task-nimisiin olioihin tallennetut tiedot ovat oikeassa muodossa. Jos jokin annetun olion muuttujista on väärässä muodossa, toimeenpanee funktio poikkeuksen.

Yksikkötestin alkupuolella muodostetaan toisia listoja sisältävä lista nimeltään invalid_cases. Sen sisällä olevat listat sisältävät kukin joukon argumentteja, joista aina yksi on väärässä muodossa kutakin listaa kohden. Näiden argumenttien pohjalta muodostetaan Task-olioita, jotka annetaan testattavalle funktiolle argumentteina yksi kerrallaan. Samalla varmistetaan, että kukin syötetty argumentti aiheuttaa ValueError-poikkeuksen, eli testattava funktio tunnistaa, että olio sisältää väärässä muodossa olevan muuttujan.

```
import unittest


class TestValidateConfValues(unittest.TestCase):
    def test_with_invalid_values(self):
        invalid_cases = [
            [123, "09:00", 1, "url", {"s1":"v1","s2":"v2"}, "out"],
            ["s2", 123, 1, "url", {"s1":"v1","s2":"v2"}, "out"],
            ["s3", "09:00", "1", "url", {"s1":"v1","s2":"v2"}, "out"],
            ["s4", "09:00", 1, 7, {"s1":"v1","s2":"v2"}, "out"],
            ["s5", "09:00", 1, "url", "not_a_dictionary", "out"],
            ["s6", "09:00", 1, "url", {"s1":123,"s2":"v2"}, "out"],
            ["s7", "09:00", 1, "url", {"s1":"v1","s2":"v2"}, False],
        ]

        scrapers = [Task(*params) for params in invalid_cases]
        for scraper in scrapers:
            with self.subTest(scraper=scraper.name):
                with self.assertRaises(ValueError):
                    confighandler.validate_conf_values(scraper)
```

Esimerkkikoodi 7. Esimerkki Pythonin standardikirjastoon kuuluvan unittest-kirjaston käytöstä.

Lokitoiminnallisuus

Edes kattavalla yksikkötestauksella ei voida varmistua ohjelman täydellisestä vakaudesta. Kun ohjelman suorituksessa ilmenee ongelmia, voi niiden

paikantaminen olla hyvinkin vaikeaa. Esimerkiksi tilanteet, joissa verkkoharava ei ole kerännyt kaikkia tietoja onnistuneesti joinakin ajankohtina, voivat tuottaa suuria vaikeuksia. Virheiden diagnosointi jälkikäteen voi osoittautua hyvinkin vaikeaksi prosessiksi.

Prosessin helpottamiseksi projektiin lisättiin kattava lokitoiminnallisuus, joka merkitsee ylös kaikki tärkeimmät suoritusvaiheet ja esiintyneet virheet lisätietoineen. Jos kesken seurannan huomataan, että jokin verkkoharava ei ole hoitanut työtään toivotulla tavalla, voidaan mahdolliset virheet paikantaa jälkikäteen lokitiedostoista.

Virheiden paikantamisen lisäksi lokeista voi olla apua projektin uusille kehittäjille, kun pyritään ymmärtämään ohjelman toimintaa ja erilaisten ajonaikaisten tapahtumien järjestystä. Lisäksi lokit mahdollistavat ohjelman käytön kattavan kirjanpidon, jota voidaan käyttää hyödyksi erilaisissa käyttötarkoituksissa. [18.]

Ilman lokitiedostoja myös käyttäjien raportoimia virheitä voi olla huomattavasti vaikeampi uusia korjausten suunnittelun yhteydessä. Kun kaikki ohjelman merkittävimmät tapahtumat ja tilat on kirjattu ylös, voidaan virhetilanteet uusia huomattavasti helpommin. Lokit eivät tule kattamaan kaikkia mahdollisia virhetilanteita, mutta niiden minimaalisiin kustannuksiin nähden niistä on huomattavaa apua.

4 Esimerkki verkkoharavan konfiguroinnista

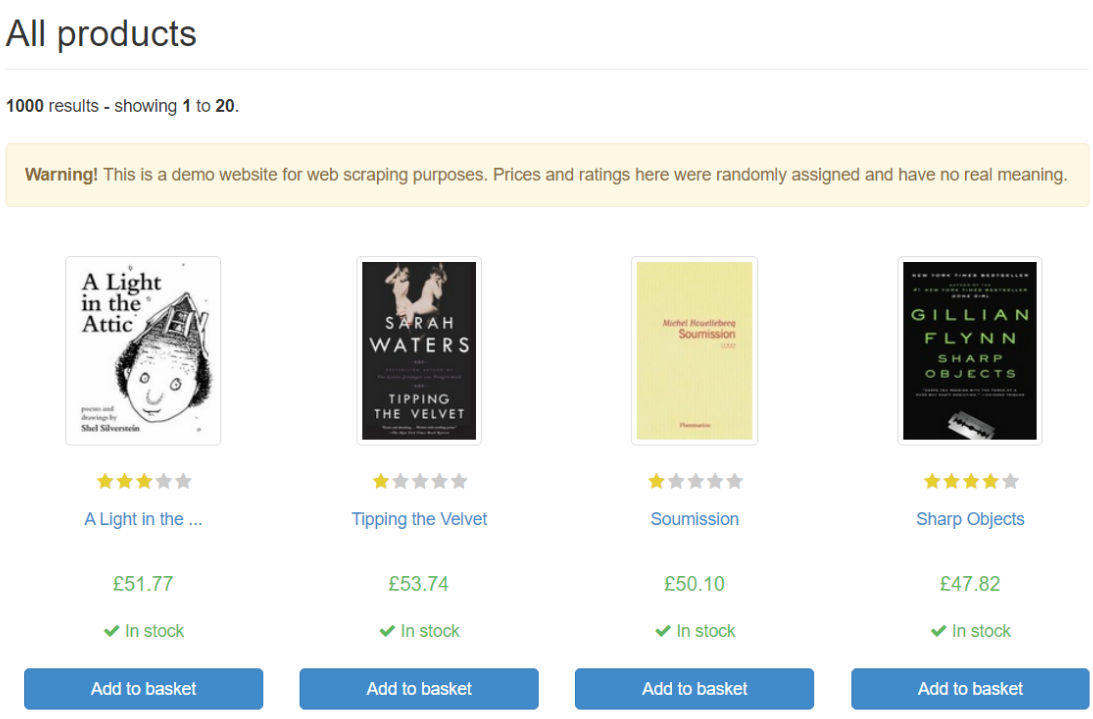
Tässä luvussa käydään läpi konkreettinen esimerkki verkkoharavan konfiguroinnista. Esimerkissä keskitytään hintatietojen keräämiseen verkkokaupasta, jonka tuoteluettelo sisältää suuren määrän tuotteita jaettuna useille sivuille. Tarkoituksena on antaa kattava peruskuva ohjelman toiminnasta sen tyypillisessä käyttötarkoituksessa.

Osoitteesta https://books.toscrape.com/ on löydettävissä verkkoharavointitarkoituksiin rakennettu demosivusto, joka jäljittelee kuvitteellista kirjojen

verkkokauppaa (kuva 8). Sivuston tekijät tarjoavat sen testiympäristöksi verkkoharavoinnin harjoittelua ja testausta varten. Sivu on vapaasti löydettävissä verkosta, ja sitä on hyödynnetty myös tämän työn testiaineistona.

Kuvitteellinen verkkokauppa sisältää 1000 tuotetta, joista näytetään yhdellä sivulla kerrallaan 20. Tuotesivujen välillä pääsee liikkumaan sivun alalaidasta löytyvän next-napin avulla. Tuoteluettelossa kullekin kirjalle on ilmoitettu nimi ja kuva sekä satunnaisesti arvotut hinta-, arvostelu- ja varastosaldotiedot.

Hintatiedot ovat helppo käytännöllinen esimerkki työssä kehitetyn ohjelman käyttötarkoituksesta. Tuoteluetteloiden haravointi sekä sivulta toiselle siirtyminen ovat myös hyviä esimerkkejä ohjelman kyvykkyyksistä. Niinpä tässä esimerkissä käydään läpi, miten kirjojen nimet ja hinnat kerätään talteen ja miten siirtyminen tuotesivujen välillä tapahtuu.



Kuva 8. Verkkoharavointitarkoituksiin rakennettu kirjojen verkkokauppa.

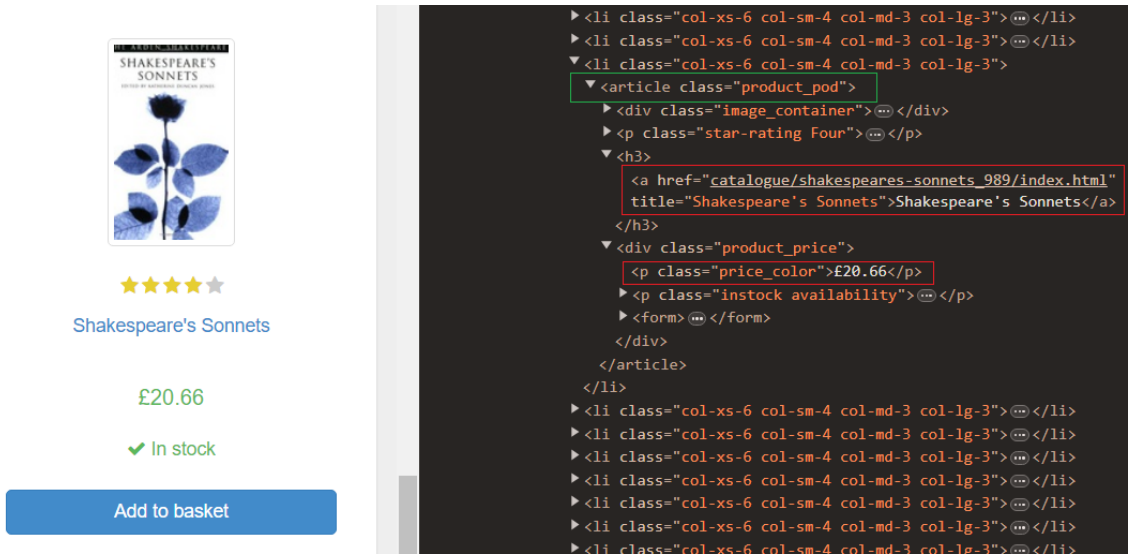
Konfiguraatiossa kerättävät tiedot määritellään CSS-valitsimien avulla, jotka ovat CSS-kielessä käytetty tapa HTML-elementtien valitsemiseen. CSS-

valitsimet määritellään kohdesivun lähdekoodin perusteella, jota voidaan tarkastella kätevästi esimerkiksi Chromen DevTools-näkymässä. Valitsimet kirjoitetaan konfiguraatioon selectors-nimisen listan alle.

Kuvassa 9 on ympyröitynä punaisella kirjan nimen ja hinnan HTML-elementit. Verkkoharava tulee paikantamaan nämä elementit niistä määritettyjen CSS-valitsimien avulla.

Pelkkä tuotteiden nimien ja hintojen CSS-valitsimien määrittely ei kuitenkaan riitä. Sivu sisältää useita nimi- ja hintatietoja, jotka tultaisiin ilman tarkempaa määrittelyä haravoimaan ja yhdistämään niiden esiintymisjärjestyksessä. Tämä saattaisi joissain tapauksissa toimiakin, mutta riski nimi- ja hintatietojen väärästä ryhmittelystä on liian suuri.

Oikean ryhmittelyn takaamiseksi konfiguraatioon on rakennettu tuki parent_selector-nimiselle muuttujalle. Parent_selector määrittää CSS-valitsimen HTML-elementeille, joista kukin sisältää haravoitavat tiedot, eli niin sanotusti vanhemmalle tai ylemmän tason elementille (engl. parent element). Yleensä tämä on niin sanottu tuotekortti, joka sisältää muun muassa tuotteen kuvan, nimen ja hinnan. Teknisesti ajateltuna verkkoharava kerää ensin kaikki parent_selectoria vastaavat elementit ja etsii tuotteiden nimi- ja hintatiedot löydettyjen elementtien sisältä yksi kerrallaan. Kuvassa 9 tuotekorttia vastaava HTML-elementti on ympyröitynä vihreällä.



Kuva 9. Toivotut HTML-elementit ympyröitynä sivun lähdekoodissa.

CSS-valitsimet voitaisiin määrittää käsin silmäilemällä sivun lähdekoodia. Tämä ei kuitenkaan ole nopein keino, sillä valitsimet voidaan usein kopioida suoraan selainten kehitysnäkymistä. Chromen DevTools-näkymässä CSS-valitsimen voi kopioida painamalla HTML-koodin elementtiä oikealla hiirinäppäimellä ja valitsemalla copy-valikosta selector.

Usein kopioidut valitsimet eivät kuitenkaan toimi verkkoharavoinnin kannalta sellaisinaan, ja niitä täytyy vielä muokata. Selaimet pyrkivät muodostamaan kopioitavista valitsimista juuri kyseiseen elementtiin viittaavia. Tällaiset valitsimet toimivat hyvin esimerkiksi yksittäisen tuotteen haravoinnissa, jossa oikeita haravoitavia elementtejä on sivulla juuri yksi. Tuoteluetteloiden haravoinnissa sivu sisältää useita samankaltaisia HTML-elementtejä, joita yritetään haravoida. Jos kopioitu valitsin viittaa vain yhteen näistä, eivät kaikki tule valituiksi.

Esimerkkikoodi 8 sisältää elementtien kopioidut CSS-valitsimet sekä niistä toimintakuntoisiksi muokatut versiot. Parent_selectorina toimivan article-elementin valitsin sisälsi määritteen ”li:nth-child(12)”, joka viittaa 12. listaelementtiin edeltävän ol-elementin (ordered list) sisällä. Tämä valitsisi vain kyseisen kirjan, joka on kahdestoista sivulla luetelluista kirjoista. Jotta kyseinen valitsin viittaisi kaikkiin sivulla lueteltuihin kirjoihin, poistettiin li-elementin perästä valitsin ”nth-

child()”. Aikaisemmin esiintyvä samanlainen valitsin ei haittaa, sillä kaikki kirjat sijaitsevat saman ol-elementin sisällä.

Article-elementin valitsin toimisi jo tällä muutoksella, mutta se sisältää yhä suuren määrän turhaa tekstiä. Sivun lähdekoodia katsomalla voidaan todeta, että sivu sisältää vain yhden section-elementin. Kun koodi sisältää vain yhden samanlaisen HTML-elementin, viittaa sen nimi CSS-valitsimessa aina siihen itseensä. Tämän takia kaikki edeltävät valitsimet voidaan poistaa, jolloin kokonaisuudesta tulee huomattavasti helppolukuisempi.

Myös kirjojen nimien ja hintojen valitsimia tulee muokata. Kun konfiguraatiossa määritellään arvo parent_selector-asetukselle, etsii ohjelma kerättäviä tietoja vain näiden elementtien sisältä. Tämä tarkoittaa, ettei ohjelma enää näe koko sivun HTML-rakennetta. Haku tapahtuu siis article-elementin sisältä, joten kaikki tätä edeltävät valitsimet tulee poistaa article-elementti mukaan luettuna.

```
/*article-elementti*/
#default > div > div > div > div > section > div:nth-child(2) > ol >
li:nth-child(12) > article

/*tuotteen nimi*/
#default > div > div > div > div > section > div:nth-child(2) > ol >
li:nth-child(12) > article > h3 > a

/*tuotteen hinta*/
#default > div > div > div > div > section > div:nth-child(2) > ol >
li:nth-child(12) > article > div.product_price > p.price_color


/*MUOKATUT CSS-VALITSIMET*/

/*article-elementti*/
section > div:nth-child(2) > ol > li > article

/*tuotteen nimi*/
h3 > a

/*tuotteen hinta*/
div.product_price > p.price_color
```

Esimerkkikoodi 8. Suoraan Chromen DevTools-näkymästä kopioidut CSS-valitsimet sekä alla niistä muokatut toimintakuntoiset versiot. Kommentit on merkitty CSS-kielen mukaisesti vinoviiva- ja tähtimerkkien sisään.

Näiden muutosten jälkeen valitsimet ovat valmiita lisättäviksi konfiguraatiotiedostoon. Alustava konfiguraatiotiedosto on kuvattuna esimerkkikoodissa 9. Tämä verkkoharava tultaisiin suorittamaan joka päivä kello 9 aamupäivällä. Koska kyseessä on tuoteluettelon haravointi, määritetään konfiguraatioon myös asetus catalog, jonka arvoksi asetetaan ”true”. Tämä kertoo ohjelmalle, että kyseessä on tuoteluettelo.

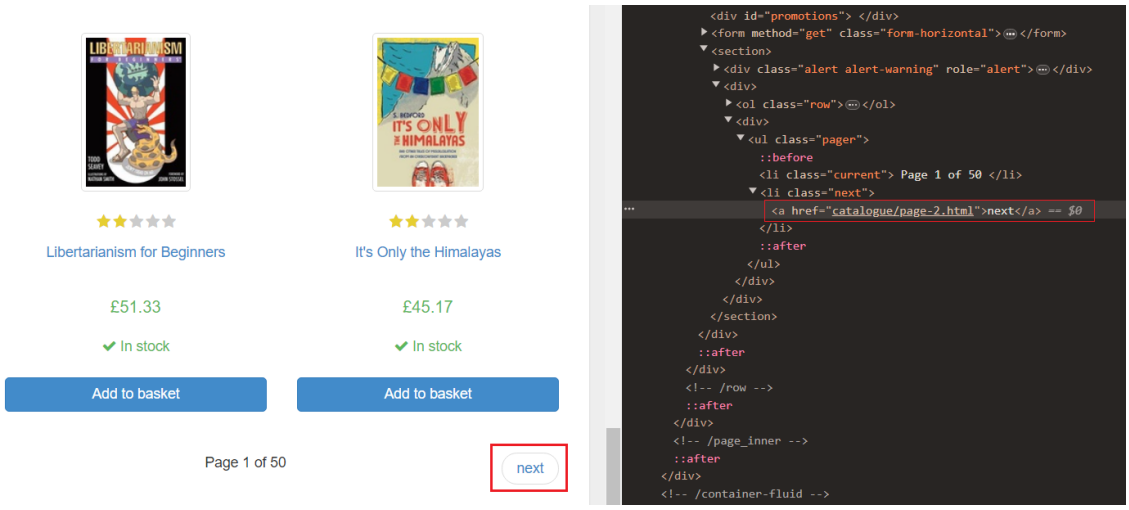
```
scraper_name: "Kirjaharava"
schedule_time: "09:00"
run_every_x_days: 1
url: "https://books.toscrape.com/"

selectors:
  nimi: "h3 > a"
  hinta: "div.product_price > p.price_color"
catalog: true
parent_selector: "section > div:nth-child(2) > ol > li > article"
```

Esimerkkikoodi 9. Haravan alustava YAML-kielinen konfiguraatio.

Tällä konfiguraatiolla voitaisiin jo haravoida ensimmäisen sivun kahdenkymmenen kirjan tiedot. Sivulta halutaan kuitenkin kerätä kaikki sen sisältämät tuhat kirjaa, mitä varten verkkoharavan tulee kyetä siirtymään tuotesivujen välillä.

Sivun oikeasta alalaidasta on löydettävissä next-niminen painike, jota painamalla sivu lataa seuraavan sivun ja sen sisältämät 20 kirjaa. Kuvassa 10 kyseinen painike sekä sitä vastaava elementti sivun lähdekoodissa on ympyröity punaisella.



Kuva 10. next-painike, josta siirrytään seuraavalle sivulle.

Ohjelma etsii kyseisen napin CSS-valitsimen avulla. Valitsin määritetään konfiguraatiossa arvolle pagination_next_selector. CSS-valitsin kopioidaan talteen edellä mainitulla menetelmällä. Tämä valitsin toimisi jo sellaisenaan, mutta sitä on myös mahdollista lyhentää helpommin luettavaan muotoon. Sivulla on vain yksi listaelementti (li), jolle on määritetty luokaksi arvo next. Tämä tarkoittaa, että valitsin ”li.next” viittaa aina juuri tähän tiettyyn listaelementtiin, ja kaikki tätä edeltävä teksti voidaan poistaa valitsimesta. Muotoiltu valitsin voidaan kirjoittaa konfiguraatioon.

Verkkoharavan konfiguraatio on nyt valmis. Jos harava haluttaisiin ajastaa, tulisi se ja muut ajastettavat haravat tallentaa omiin kansioihinsa yhteisen konfiguraatiokansion sisään. Tämä ylimmän tason kansio annettaisiin ohjelmalle argumenttina sen käynnistyksen yhteydessä, ja ohjelma tallentaisi kerätyt tiedot CSV-tiedostoihin samoihin kansioihin konfiguraatiotiedostojen kanssa. Harava on myös mahdollista suorittaa manuaalisesti ilman ajastamista. Lopullinen konfiguraatio on kuvattu esimerkkikoodissa 10.

```
scraper_name: "Kirjaharava"
schedule_time: "09:00"
run_every_x_days: 1
url: "https://books.toscrape.com/"

selectors:
  nimi: "h3 > a"
  hinta: "div.product_price > p.price_color"
catalog: true
parent_selector: "section > div:nth-child(2) > ol > li > article"

pagination_next_selector: "li.next > a"
```

Esimerkkikoodi 10. Verkkoharavan lopullinen konfiguraatio.

Selectors-listan avaimet määrittävät CSV-tiedoston sarakkeiden nimet. Jotta CSV-tiedostosta kävisi ilmi, milloin tiedot on kerätty, lisää ohjelma automaattisesti sarakkeisiin myös time-nimisen sarakkeen, josta löytyvät haravointikertojen päivämäärät ja kellonajat. Eri haravointikertojen tiedot kertyvät samaan CSV-tiedostoon edellisillä kerroilla kerättyjen tietojen alle. CSV-tiedostot voidaan avata katseltavaksi esimerkiksi Excelissä, joka tarjoaa erilaisia keinoja tietojen rajaukseen. Kerätyt tiedot ovat nähtävissä kuvassa 11 avattuna Excelissä.

time	nimi	hinta
28.02.2024 12:25	A Light in the ...	£51.77
28.02.2024 12:25	Tipping the Velvet	£53.74
28.02.2024 12:25	Soumission	£50.10
28.02.2024 12:25	Sharp Objects	£47.82
28.02.2024 12:25	Sapiens: A Brief History ...	£54.23
28.02.2024 12:25	The Requiem Red	£22.65
28.02.2024 12:25	The Dirty Little Secrets ...	£33.34
28.02.2024 12:25	The Coming Woman: A ...	£17.93
28.02.2024 12:25	The Boys in the ...	£22.60
28.02.2024 12:25	The Black Maria	£52.15
28.02.2024 12:25	Starving Hearts (Triangular Trade ...	£13.99
28.02.2024 12:25	Shakespeare's Sonnets	£20.66
28.02.2024 12:25	Set Me Free	£17.46
28.02.2024 12:25	Scott Pilgrim's Precious Little ...	£52.29
28.02.2024 12:25	Rip it Up and ...	£35.02
28.02.2024 12:25	Our Band Could Be ...	£57.25
28.02.2024 12:25	Olio	£23.88

Kuva 11. Kirjojen verkkokaupasta kerätyt tiedot avattuna Excelissä.

5 Ohjelman käyttöohjeiden laatiminen

Kehitetyn ohjelman käyttö voi tuntua hyvinkin itsestään selvältä sen kehittäjälle, mutta mitä todennäköisimmin ei ollenkaan ensikertalaiselle käyttäjälle. Jotta verkkoharavaohjelman tulevat käyttäjät ymmärtäisivät, miten ohjelmaa tulee käyttää ja millaisiin erilaisiin käyttötarkoituksiin sitä voidaan hyödyntää, täytyi sille kirjoittaa selkeä käyttöohje.

Käyttöohje kirjoitettiin Markdown-kielellä, joka tarjoaa kattavan ja yhtenäisen tekstinmuotoilun ilman riippuvuutta mistään tietystä tietokoneohjelmasta. Markdown-tiedostot ovat normaaleja tekstitiedostoja, joissa tekstin muotoilut määritellään kielen oman syntaksin avulla. Muotoilu on nähtävillä, kun tiedostoa katselee ohjelmassa, joka tukee Markdown-tiedostojen renderöintiä. Yksinkertaisessa tekstitiedostomuodossa olevat ohjeet on tarvittaessa helppo siirtää paikasta toiseen tai muuttaa toisenlaiseen muotoon.

Readme-tiedosto käy läpi ohjelman asennuksen, tuetut komennot ja yleiset käyttöohjeet. Erilliseen ohjeeseen (liite 1) lisättiin tarkat kuvaukset kunkin konfiguraation tukeman asetuksen käytöstä. Mukaan liitettiin havainnollistavat esimerkit sekä kuvaukset kunkin asetuksen tyypistä.

6 Yhteenveto

Työssä kehitettiin Leaves-niminen verkkoharavointiohjelma työn tilaajalle Kilpailu- ja kuluttajavirastolle. Ohjelmasta oli tavoitteena luoda mahdollisimman helppokäyttöinen ratkaisu automatisoidun tiedon keräykseen viraston eri yksiköiden tarpeisiin. Kaikilla ohjelman käyttäjillä ei tulisi olemaan kattavaa ohjelmointiosaamista, joten verkkoharavoiden määrittely haluttiin toteuttaa itse ohjelmakoodista riippumattomasti.

Ohjelma toteutettiin komentoriville, josta sen käyttö onnistuu yksinkertaisten komentojen avulla. Yksittäisten haravoiden suorituksen lisäksi ohjelma tukee

myös suurten haravamäärien ajastamista, mikä mahdollistaa automatisoidun, pitkäkestoisen ja säännöllisen datan keräyksen.

Suoritettavat verkkoharavat konfiguroidaan erillisissä YAML-kielisissä konfiguraatiotiedostoissa, joista ohjelma lukee tarvittavat tiedot ajon aikana. Erilliset konfiguraatiotiedostot mahdollistavat ohjelmakoodista riippumattoman haravoiden määrittelyn, joka on myös käyttäjien kannalta helposti lähestyttävä. Koska jokaista haravaa varten luodaan oma konfiguraatiotiedosto, eivät muutokset vaikuta muihin käytössä oleviin haravoihin.

Konfiguraation tukemat asetukset mahdollistavat erilaisten sivujen haravoinnin. Konfiguroidut verkkoharavat kykenevät esimerkiksi vierittämään sivua alaspäin, siirtymään tuoteluettelosivujen välillä sekä lataamaan sivulle lisää sisältöä painikkeiden kautta. Näiden toiminnallisuuksien avulla tietoja saadaan kerättyä kattavasti erilaisilta sivuilta.

Lähteet

1 Khder, Moaiad Ahmad. 2021. Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application. Verkkoaineisto. International Journal of Advances in Soft Computing and its Application. <https://ijasca.zuj.edu.jo/PapersUploaded/2021.3.11.pdf>. 11.2021. Luettu 4.1.2024.

2 Tietoa virastosta. Verkkoaineisto. Kilpailu- ja kuluttajavirasto. <https://www.kkv.fi/tietoa-virastosta/>. Luettu 6.1.2024.

3 Persson, Emil. 2019. Evaluating tools and techniques for web scraping. Master's thesis. KTH Royal Institute of Technology. DiVA-tietokanta.

4 Meschenmoser, Philipp; Meuschke, Norman; Hotz, Manuel & Gipp, Bela. 2016. Scraping Scientific Web Repositories: Challenges and Solutions for Automated Content Extraction. D-Lib Magazine. Vol. 22, numero 9/10.

5 Static and Dynamic Websites. Verkkoaineisto. UIC Library. <https://researchguides.uic.edu/digital-uic/static-dynamic>. Päivitetty 18.10.2023. Luettu 14.1.2024.

6 Lazy loading. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading>. Päivitetty 21.12.2023. Luettu 14.1.2024.

7 Heath, Anthony. 2023. Web Scraping 101: Tools, Techniques and Best Practices. Verkkoaineisto. Medium. <https://medium.com/geekculture/web-scraping-101-tools-techniques-and-best-practices-417e377fbeaf>. 22.3.2023. Luettu 10.3.2024.

8 Andaur, Cierra. 2021. Web Scraping 101: Avoiding Detection. Verkkoaineisto. Medium. <https://cierra-andaur.medium.com/web-scraping-101-avoiding-detection-4d5132fdb4bd>. 25.5.2021. Luettu 10.3.2024.

9 What are HTTP headers? 2023. Verkkoaineisto. Postman Blog. <https://blog.postman.com/what-are-http-headers/>. 11.7.2023. Luettu 10.3.2024.

10 PyPI · The Python Package Index. Verkkoaineisto. <https://pypi.org/>. Luettu 17.2.2024.

11 Van Deusen, Anna. 2023. Python Popularity: The Rise of A Global Programming Language. Verkkoaineisto. Flatiron School. <https://flatironschool.com/blog/python-popularity-the-rise-of-a-global-programming-language/>. 31.1.2023. Luettu 17.2.2024.

12 Uspenski, Anastasija. 2023. Why VS Code remains a developer favorite, year after year. Verkkoaineisto. ShiftMag. <https://shiftmag.dev/vs-code-171/>. 25.4.2023. Luettu 17.2.2024.

13 YAML Ain't Markup Language (YAML™) version 1.2 Revision 1.2.2 (2021-10-01). Verkkoaineisto. The Official YAML Web Site. <https://yaml.org/spec/1.2.2>. 1.10.2021. Luettu 11.1.2024.

14 CSS selectors. Verkkoaineisto. Mdn web docs. <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_selectors>. Päivitetty 9.2.2024. Luettu 3.3.2024.

15 CSS Selector Reference. Verkkoaineisto. W3Schools. <https://www.w3schools.com/cssref/css_selectors.php>. Luettu 3.3.2024

16 Smith, Eric V. 2017. PEP 557 – Data Classes. Verkkoaineisto. Python Enhancement Proposals. <https://peps.python.org/pep-0557/>. 2.6.2017. Luettu 12.2.2024.

17 Typer. Verkkoaineisto. <https://typer.tiangolo.com/> Luettu 19.2.2024.

18 Li, Heng; Shang, Weiyi; Adams, Bram; Sayagh, Mohammed & Hassan, Ahmed E. 2020. A Qualitative Study of the Benefits and Costs of Logging From Developers' Perspectives. IEEE Transactions on Software Engineering. Vol. 47, numero 12, s. 2858–2873.

Konfiguraatioarvojen käyttöohje

Konfiguraatiot

Tämä tiedosto sisältää tietoja konfiguraatioiden tukemista määritteistä. Tietoa ohjelman asennuksesta ja käytöstä on löydettävissä `README.md`-tiedostosta.

Tuetut asetukset

- **name**
- **schedule_time**
- **run_every_x_days**
- **start_date**
- **end_date**
- **url**
- **selectors**
- **cookie_selector**
- **catalog**
- **parent_selector**
- **lazy**
- **pagination_next_selector**
- **load_more_selector**

name

`string`

Vapaavalintainen nimi haravalle.

```
name: "Hintaharava"
```

schedule_time

`string`

Mihin kellonaikaan haravointi tulisi suorittaa? Ilmoitettava 24 tunnin muodossa neljällä numerolla erotettuna kaksoispisteellä.

```
# kelpaa
schedule_time: "09:00"

# EI KELPAA!
schedule_time: "9:00"
```

run_every_x_days

integer

Kuinka monen päivän välein haravointi tulisi suorittaa? Ilmoitettava kokonaislukuna. Numero 1 vastaa päivittäistä suorittamista.

```
run_every_x_days: 1

# HUOM: ei lainausmerkkejä, koska muotoa integer.
```

start_date

string

Päivämäärä, jona haravointi tulisi aloittaa.

```
start_date: "2023-03-04" #YYYY-MM-DD
```

end_date

string

Päivämäärä, jona haravointi tulisi lopettaa. Haravointi suoritetaan vielä määritettynä päivänä.

```
end_date: "2023-06-04" #YYYY-MM-DD
```

url

string

Haravoitavan verkkosivun URL-osoite.

```
url: "https://verkko-osoite.fi"
```

selectors

`list`

Lista avain-arvopareja, jotka määrittävät, mitä verkkosivulta halutaan kerätä. Kukin avain-arvopari vastaa yhtä saraketta CSV-tiedostoissa. Avaimet määrittävät nimen kerättävälle tiedolle ja sarakkeet CSV-tiedostoissa nimetään niiden mukaan. Voit itse valita, mitä annat avainten nimiksi. Avainten arvot puolestaan määrittävät CSS-valitsimet, joiden avulla oikea tieto paikannetaan verkkosivuilta. Avain-arvo -pareien määrälle ei ole määriteltyä rajaa, ja niitä voi listata konfiguraatioon tarvitun määrän.

HTML

```
<p class="tuote">Pesusieni</p>
<p class="hinta">20 €</p>
```

YAML-konfiguraatio

```
selectors:
  name: "p.tuote"
  price: "p.hinta"
```

Esimerkiksi Chrome-selaimessa elementin CSS-valitsimen saa kopioitua helposti DevTools-näkymästä painamalla elementtiä oikealla hiirinäppäimellä ja valitsemalla kopiointivalikosta "Copy selector". Kopioituun valitsimeen voi olla vielä tarpeellistä tehdä muutoksia haravoinnin onnistumiseksi.

Lisätietoa CSS-valitsimista löytyy mm. osoitteesta https://www.w3schools.com/cssref/css_selectors.php.

cookie_selector

`string` | valinnainen

Evästeikkunan hyväksymisnapin CSS-valitsin.

```
cookie_selector: "button.accept"
```

catalog

`true`/`false` | valinnainen

Totuusarvo, joka määrittää, käytetäänkö selectors-listan valitsimia useiden elementtien keräämiseen. Esimerkkinä tuotekatalogi, josta halutaan kerätä jokaisen tuotteen nimi ja hinta. Katalogitoiminnallisuutta käytettäessä parent_selector tulee myös olla määriteltynä.

```
# katalogien haravointi käytössä
catalog: true

# katalogien haravointi pois käytöstä
catalog: false
```

parent_selector

`string` | valinnainen

Jos catalog-asetuksen arvoksi on määritelty `true`, parent_selector määrittää CSS-valitsimen elementeille, joista kukin sisältää selectors-listan valitsimien elementit. Tätä käytetään varmistamaan, että haravoitavat tiedot ryhmitellään oikein katalogiharavoinnissa.

Esimerkki tuotekatalogin HTML-koodista:

```
<div class="tuotekortti">
  <p class="tuote">Pesusieni</p>
  <p class="hinta">20 €</p>
</div>
<div class="tuotekortti">
  <p class="tuote">Lyijykynä</p>
  <p class="hinta">5 €</p>
</div>
```

Konfiguraatio kyseistä HTML-koodia varten. parent_selector varmistaa, että kerätyssä datassa lyijykynän hintana pysyy 5 euroa ja pesusienen 20, eivätkä ne mene ristiin.

```
catalog: true
parent_selector: "div.tuotekortti"
selectors:
  name: "p.tuote"
  price: "p.hinta"
```

Kun parent_selector on käytössä, tapahtuu haravoitavien elementtien haku vain sen sisältä. Tämä tarkoittaa, että selectors-listan valitsimet eivät voi enää käyttää tietoja parent_selector-elementin ulkopuolelta, ja täten niiden valitsimien täytyy olla suhteutettuna parent_selectorin elementtiin. Esimerkkinä seuraava HTML-rakenne:

```
<div class="tuoteluettelo">
  <div class="tuotekortti">
    <p class="tuote">Pesusieni</p>
    <p class="hinta">20 €</p>
  </div>
  <div class="tuotekortti">
    <p class="tuote">Lyijykynä</p>
    <p class="hinta">5 €</p>
  </div>
</div>
```

```
parent_selector: "div.tuoteluettelo div.tuotekortti"
selectors:

  # Ei toimi, sillä div.tuoteluettelo on parent_selector-elementin ulkopuolella.
  nimi: "div.tuoteluettelo div.tuotekortti p.tuote"

  # Tomii
  nimi: "div.tuotekortti p.tuote" # tai vain "p.tuote"
```

Lisätietoa CSS-valitsimista: https://www.w3schools.com/cssref/css_selectors.php

lazy

`true`/`false` | valinnainen

Määrittää, käyttääkö haravoitava sivu laiskaa latausta, eli latautuuko sivulle lisää elementtejä, kun sitä vieritetään alaspäin. Kun arvoksi asetetaan `true`, harava kelaa sivun alas ennen seuraavia toimenpiteitä.

```
lazy: true
```

pagination_next_selector

`string` | valinnainen

Määrittää CSS-valitsimen HTML-elementille, jota painamalla siirrytään seuraavalle sivulle katalogiharavoinnin yhteydessä.

```
pagination_next_selector: "li.pages span.icon"
```

load_more_selector

`string` | valinnainen

Määrittää CSS-valitsimen HTML-elementille, jota painamalla sivulle latautuu lisää sisältöä. Jos lazy on asetettu todeksi, kelataan sivu aina alas ennen lataa lisää -napin painamista. Voidaan käyttää pagination_next_selector-arvon kanssa yhdessä.

```
load_more_selector: "button.load-more"
```