OULUN AMMATTIKORKEAKOULU

Mika Lipponen

VERKKOSIVUJEN HARAVOINNILLA TOTEUTETTU MUSIIKKISOITIN

VERKKOSIVUJEN HARAVOINNILLA TOTEUTETTU MUSIIKKISOITIN

Mika Lipponen
Opinnäytetyö
Syksy 2020
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma, Internet-palvelut ja digitaalinen media

---

Tekijä: Mika Lipponen
Opinnäytetyön nimi: Verkkosivujen haravoinnilla toteutettu musiikkisoitin
Työn ohjaaja: Tuula Ijäs
Työn valmistumislukukausi ja -vuosi: Syksy 2020 Sivumäärä: 47+1

Opinnäytetyössä ohjelmoidaan Python-ohjelmointikielellä musiikkisoitin, joka tarjoaa vaihtoehtoisen tavan käyttää Bandcamp-musiikkipalvelua. Tämä projekti sai alkunsa henkilökohtaisesti koetuista puutteista Bandcamp-palvelun selainpohjaisessa käyttöliittymässä, ja ajatuksesta toteuttaa näiden ongelmakohtien ratkaisemiseen keskittyvä, tiedonharavointiin perustuva Python-ohjelma. Kyseisen projektiohjelman ohjelmoiminen toimii kehyksenä opiskella ja tutustua sekä tiedonharavoinnin perusteisiin että saatavilla oleviin työkaluihin.

Projektiohjelman kohdelaitteistona toimii Raspberry Pi -minitietokone, joka on paremmin mitoitettu laite musiikin toiston tarpeisiin kuin tavallinen tietokone. Ohjelman käyttäminen tapahtuu pääte-emulaattorin ja SSH-yhteyden kautta miltä tahansa lähiverkossa olevalta laitteelta.

Tämän opinnäytetyön käytännön toteutus jakautui kahteen vaiheeseen. Ensimmäisessä vaiheessa tutustuttiin erilaisiin saatavilla oleviin tiedonharavoinnin työkaluihin sekä muihin ohjelmakomponentteihin, ja opeteltiin niiden käyttämistä projektiohjelman ohjelmoimisen lomassa. Käytännössä kaikki projektin ohjelmakomponentit olivat aloittaessa miltei kokonaan tuntemattomia. Kun projektiohjelma oli saatu ohjelmoitua sille asetetut tavoitteet täyttävään pisteeseen, siirryttiin kirjoittamaan suoritetun ohjelmointiprojektin ja sen opettamien asioiden pohjalta varsinaista opinnäytetyötä.

Opinnäytetyö toimii tiedonharavointiin perustuvan ohjelmointiprojektin läpileikkauksena, esitellen sekä tiedonharavoinnin perusteita että erilaisia tiedonharavoinnin työkaluja ja niiden käyttämistä. Ajatuksena on, ettei lukijan tarvitse välttämättä tietää tiedonharavoinnista mitään ennen opinnäytetyön lukemista, ja että opinnäytetyö toimii itsenäisenä ja käytännönläheisenä tutustumismateriaalina aiheen pariin.

Tuloksena syntynyt projektiohjelma osoittaa, että Pythonille saatavilla olevat tiedonharavoinnin työkalut ovat riittävän yksinkertaisia, että niillä on mahdollista toteuttaa ohjelmaprojekti jopa hyvin vähäisellä aikaisemmalla kokemuksella ohjelmointikielestä. Tärkeintä on löytää itseään kiinnostava aihe, jonka avulla asioita jaksaa opiskella ja kokeilla.

Asiasanat: Verkkosivujen haravointi, Python, pääte-emulaattori, komentorivi, ohjelmointi, Raspberry Pi.

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems, Internet-services and Digital Media

---

Author: Mika Lipponen
Title of thesis: Music player based on web scraping
Supervisor: Tuula Ijäs
Term and year when the thesis was submitted: Autumn 2020
Number of pages: 47+1

This thesis documents the creation of a Python CLI program that provides an alternative way to use a certain part of the Bandcamp music service. The resulting project program uses web scraping in lieu of a nonexistent Bandcamp Web Service API to retrieve the information it requires. It is a project born out of personal frustration with some of the aspects and shortcomings present on Bandcamp's default browser user interface, specifically when it comes to going through the discography of a certain publisher. Fixing those specific shortcomings of the default user interface formed the basis of the project, and provided a real world target for learning about different aspects of web scraping, including the usage of different software tools available for scraping.

The target device for the project program was the Raspberry Pi single-board computer, as it is much better suited for music player duty than a normal computer, and lends itself well for continuous operation. The CLI project program would be controlled from a terminal emulator, through a SSH connection.

The main conclusion from the project was that Python provides easy-to-use tools for web scraping, which can be used even without extensive prior experience with the programming language. A much bigger impediment for using web scraping in projects is the uncertainty that comes with every structure change on the source sites you use, as every update on that front can easily end up breaking your scraping solution.

Keywords: Web scraping, scraping, Python, coding, command line, CLI, API

SISÄLLYS

1 JOHDANTO

Erilaisia Internet-palveluita on nykyään tarjolla loputtomasti kaikkien aiheiden tiimoilta. Käyttökokemus ja käyttöliittymä ovat tärkeimpiä palvelun menestykseen vaikuttavia asioita. Hyvästä palvelusta löytyy kattavasti asiakkaiden haluamia toimintoja, ja niiden käyttäminen on loogista ja vaivatonta. Tämän päämäärän täyttyminen jokaisen potentiaalisen asiakkaan kohdalla on kuitenkin mahdotonta, koska ihmisten käyttötarpeet ja mieltymykset eivät ole identtisiä. Suuremmat internet-palvelut tarjoavatkin usein erilaisia kolmannen osapuolen sovellusten kehittämisen mahdollistavia ohjelmointirajapintoja. Niiden avulla voidaan luoda vaihtoehtoisia palvelun käyttötapoja, joiden käyttöliittymä ja logiikka voi poiketa merkittävästikin palvelun alkuperäisestä käyttökokemuksesta.

Ohjelmointirajapintojen tarjoaminen laajentaa palvelun asiakasmäärää, koska vaihtoehtoiset käyttötavat lisäävät ihmisten todennäköisyyttä löytää juuri heidän mieltymyksiään ja tarpeitaan vastaava tapa palvelun käyttämiseen. Suurimpien palveluiden käyttämiseen löytyykin yleensä myös laaja kirjo näitä vaihtoehtoisia, kolmannen osapuolen kehittämiä käyttötapoja. Esimerkiksi Twitter kertoi jo vuonna 2011 julkaistussa blogipostauksessaan palvelun ohittaneen miljoonan rekisteröidyn applikaation rajan yli 750 000 eri kehittäjän voimin. Samalla Twitter kertoi, että palveluun rekisteröitiin uusi applikaatio 1,5 sekunnin välein. (Twitter Inc. 2011, viitattu 13.10.2020.) Vaikka uutisessa ei tarkasti määritelty sitä, mitä rekisteröity applikaatio tarkoittaa, luku kuitenkin kertoo kolmannen osapuolen ratkaisujen kehittämiseen suuntautuvasta mielenkiinnosta.

Kaikki palvelut eivät kuitenkaan syystä tai toisesta tarjoa ohjelmointirajapintaa uusien ratkaisujen kehittämiseen. Tällöin ainoa mahdollinen tapa vaihtoehtoisten sovellusten kehittämiseksi on jonkinlainen tiedonharavointiin perustuva ratkaisu.

Tässä lopputyössä kehitetään tiedonharavointiin pohjautuva, vaihtoehtoinen tapa käyttää Bandcamp-musiikkipalvelua, hyödyntäen erilaisia Python-ohjelmointikielelle saatavilla olevia ohjelmakirjastoja ja lisäosia. Erityisen huomion alla ovat luonnollisesti itse tiedonharavoinnin mahdollistavat Python-lisäosat, mutta mainitsen lyhyesti myös muita huomionarvoisia ohjelmakokonaisuuden osia.

Kolmannen osapuolen kehitysprojektit saavat usein alkunsa halusta luoda ohjelmakokonaisuus, joka mahdollistaa tai helpottaa jonkin tärkeäksi koetun asian tekemistä alkuperäiseen käyttökokemukseen verrattuna. Niin syntyi myös tämä projekti, ajatuksesta itselleni tärkeiden käyttöominaisuuksien lisäämisestä Bandcamp-musiikkipalveluun. Kyse oli siis uusien asioiden selvittämisestä ja opettelemisesta henkilökohtaisesti mielenkiintoisen kehyksen parissa; tarkoituksena ei ollut palvelun kyseenalainen hyödyntäminen, eteenkään minkäänlaisessa julkisessa mittakaavassa tai siihen tähdäten.

Tämän opinnäytetyön tietoperustassa käsitellään ensiksi tiedonharavoinnin perusidean, minkä jälkeen käydään läpi projektin rakennuspalikat: esitellään kohteena oleva palvelu, projektiohjelman kohdelaitteisto, käytetyt ohjelmakirjastot ja muut ohjelmakomponentit. Tietoperustan lopuksi tarkastellaan myös eettistä näkökulmaa. Tämän jälkeen siirrytään työn toiminnalliseen osuuteen, jossa kerrotaan ohjelmointiprojektin toteuttamisesta sekä erilaisten ohjelmakomponenttien käytöstä esimerkein. Työn lopuksi esitellään valmis projektiohjelma ja sillä saavutetut tavoitteet.

2 MILLOIN TIEDONHARAVOINTIA TARVITAAN?

Jos haluaa kehittää sovelluksen tai palvelun, joka hyödyntää jotain jo Internetissä saatavilla olevaa tietoa, on olemassa kaksi perustavasti erilaista keinoa tuoda haluttu tieto oman ohjelman käyttöön. Vaikka tämä lopputyö käsittelee näistä kahdesta vaihtoehdosta työläämpää, on silti tarpeellista ensiksi puhua muutama sana ensimmäisestä, suositeltavasta vaihtoehdosta, jota kannattaa hyödyntää jos se vain on mahdollista. Ensimmäisenä vaihtoehtona on siis selvitettävä, onko käytettävissä www-sovelluspalvelua, jonka ohjelmointirajapinta eli API (engl. Application Programming Interface) voisi tarjota halutut tiedot.

Koska tätä aihetta tarkastellessa suomenkieliset termit voivat helposti aiheuttaa väärinkäsityksiä, on syytä käsitellä ne tässä vaiheessa. Termin 'www-sovelluspalvelu' määrittely Sanastokeskus TSK ry:n Termipankki-sivuilla on "verkkopalvelimessa toimiva ohjelma, joka tarjoaa standardoitujen internetyhteyskäytäntöjen avulla palveluja sovellusten käytettäväksi". Www-sovelluspalvelu on suomennos englannin kielen termistä "Web service", joka kuitenkin voidaan myös kääntää "verkkopalveluksi" (sekä rinnakkaisiksi muodoiksi "Internet-palvelu" ja "nettipalvelu") jolloin sen määritelmä on Sanastokeskuksen mukaan "verkkosivuston kautta tarjottava palvelu". "Web service"-termistä on siis suomennettu kaksi eri asiaa tarkoittavaa termiä eli www-sovelluspalvelu, joka tarjoaa tietoja tietoteknisten järjestelmien ja ohjelmistojen käytettäväksi, ja verkko- tai Internet-palvelu, joka taas on ihmisille suunnattua sisältöä. (Termipankki 2012, viitattu 13.10.2020.)

API (Application Programming Interface) eli suomeksi ohjelmointirajapinta taas tarkoittaa pohjimmiltaan vain tietyn ohjelmiston määriteltyjä käytäntöjä ja tietomuotoja, joita noudattaen kommunikointi ulkopuolisten ohjelmien tai palvelujen kanssa on mahdollista. Www-sovelluspalvelua ei siis voi olla ilman APIa, mutta jokainen API ei ole www-sovelluspalvelu. Nykyisessä arkipuheessa vain usein tarkoitetaan jonkin www-sovelluspalvelun ohjelmointirajapintaa, kun puhutaan APIsta. Asiaan liittyvät termit demonstroivat hyvin, kuinka tietotekniikasta puhuminen erityisesti suomen kielellä tuottaa aika ajoin vaikeuksia, ja miksi usein suomessakin tyydytään käyttämään englanninkielisiä termejä.

Esimerkiksi Twitter tarjoaa www-sovelluspalvelun, jonka ohjelmointirajapinnoilla voi esimerkiksi noutaa twiittejä ja tietoja rekisteröityneistä käyttäjätileistä (Twitter Inc. 2020, viitattu 13.10.2020). Twitter käyttää nykyään yleistä www-sovelluspalveluiden arkkitehtuurimallia nimeltä REST eli

Representational State Transfer. Tätä arkkitehtuurimallia käyttävistä www-sovelluspalveluista käytetään englanninkielistä termiä RESTful Web Services tai RESTful API. Keskeisessä osassa REST-mallissa on kuusi korkean tason suunnittelun periaatetta, joita palvelun tulee noudattaa voidakseen käyttää itsestään termiä RESTful API. (restfulapi.net 2020, viitattu 14.10.2020.)

Nämä periaatteet tuovat REST-mallia noudattaviin palveluihin kaikille yhteistä logiikkaa, kuitenkaan määrittelemättä palveluille tiukkoja matalan tason toteuttamisen ohjeita. RESTful API -periaatteita noudattavat palvelut mahdollistavat haluttujen tietojen hakemisen tarkasti rajatuissa, optimaalisissa tietokokonaisuuksissa ja muodossa, joka on helposti ja luotettavasti jatkokäsiteltävissä ja tallennettavissa paikallisessa ohjelmassa. Täten on helppo ymmärtää, että APIn ollessa käytettävissä sen hyödyntäminen on paras vaihtoehto erilaisten tietojen noutamiselle.

Internetistä löytyy kuitenkin myös paljon sisältöä, jota ei ole saatavilla ohjelmistorajapinnan kautta. Näissä tapauksissa, tietojen ollessa vain ihmisille suunnatun verkkopalvelun muodossa joudutaan turvautumaan jälkimmäiseen alussa mainituista keinoista ja suorittamaan enemmän toimenpiteitä tiedon saattamiseksi hyödylliseen muotoon. Tätä toimenpiteiden kokonaisuutta kutsutaan englanninkielisellä termillä ”scraping”, suomennettuna *kaapiminen* tai *haravointi*, mikä kuvastaakin hyvin sitä, että kyseessä on APIn hyödyntämistä työläämpi tapa tiedon hankkimiseen.

Myös ”scraping” on englanninkielinen termi, jota käytetään usein suomenkielisen, vähemmän vakiintuneen termin sijaan. Sanastokeskus TSK ry:n suositus suomenkieliseksi käännökseksi on ”haravointi” eli yleisesti ”tiedonharavointi” ja, kun tiedot haetaan verkkosivulta, ”verkkosivujen haravointi” (engl. web scraping). Samassa lähteessä tarjotaan myös tiedonharavoinnille seuraavanlainen, ansiokkaan tiivistetty määritelmä: ”tietojen automaattinen kokoaminen ihmisen luettavassa muodossa olevasta aineistosta”. (Termipankki 2013, viitattu 13.10.2020.)

Tiedonharavointi koostuu kahdesta erillisestä vaiheesta: tarjolla olevan tietokokonaisuuden, kuten esimerkiksi verkkosivun, noutamisesta, ja tämän jälkeen haluttujen tietojen poimimisesta haetusta materiaalista. Verkkosivun noutaminen on melko suoraviivainen toimenpide käytettäessä hyväksi vaikkapa Pythonin lisäosia Requests tai AIOHTTP. Ensin on kuitenkin selvitettävä, minkälaisissa rakenteissa halutut tiedot verkkosivuilla sijaitsevat. Tämän perusteella voidaan sitten päättää, mitä työkalua ja tietojen poimimistapaa niihin kannattaa käyttää.

3 VERKKOSIVUJEN HARAVOINTIIN PERUSTUVA OHJELMOINTIPROJEKTI

Kuten johdannossa todettiin, tässä lopputyössä ohjelmoitiin Python-kielellä verkkosivujen haravointiin perustuva vaihtoehtoinen tapa Bandcamp-musiikkipalvelun käyttämiseen. Tarkemmin sanottuna tehtiin Python-ohjelma, joka mahdollistaa yksittäisten musiikkijulkaisujen tietojen katsomisen ja ilmaiseksi kuunneltavissa olevien, musiikkiin tutustumisen mahdollistavien rajoitetun bittinopeuden äänitiedostojen toistamisen.

Tässä luvussa esitellään projektin eri osa-alueet, aloittaen kohteena olevasta Bandcamp-musiikkipalvelusta. Sen jälkeen siirrytään kohdelaitteiston esittelyyn, ja lopuksi listataan projektissa hyödynnetyt ohjelmakomponentit.

3.1 Bandcamp-musiikkipalvelu

Bandcamp on vuonna 2007 perustettu, erityisesti digitaalisen musiikin myymiseen erikoistunut musiikkipalvelu, joka tarjoaa myös mahdollisuuden fyysisten musiikkiformaattien ja fanituotteiden kauppaamiseen. Palvelua voi käyttää maksutta oman musiikin myymiseen, jolloin Bandcamp kuittaa palvelun pääasiallisena tulon lähteenä toimivan prosenttiprovision tapahtuneista kaupoista: 15 % digitaalisesta musiikista ja 10 % fyysisistä hyödykkeistä. Kun huomioon otetaan vielä erilliset maksuliikenteen kulut, artistille jää yleensä noin 80–85 % ostohinnasta. (Bandcamp 2020a, viitattu 11.10.2020.)

Monesta digitaalista musiikkia myyvästä palvelusta poiketen Bandcamp tarjoaa musiikkiin tutustumiseksi mahdollisuuden kokonaisten kappaleiden kuunteluun 128 kb/s bittinopeuden mp3-formaatissa, rajoittamatta siis ilmaista tutustumista pelkästään lyhyeen katkelmaan (Bandcamp 2020b, viitattu 11.10.2020). Palvelu tarjoaa kuitenkin musiikintekijöille 10 dollarin kuukausimaksusta lisäominaisuuksia tuovan Pro-jäsenyyden, joka sisältää myös mahdollisuuden jättää kappaleita ilmaisen toiston ulkopuolelle. Vaikka Bandcamp sivuillaan toteaakin uskovansa, että parhain tapa saada ihmiset kiinnostumaan musiikista on antaa heidän kuulla sitä *enemmän*, ilmaisen toiston rajaaminen tarjotaan yleisesti tunnettuja tai yksinkertaisesti Bandcampin toimintaperiaatteita epäileviä artisteja silmällä pitäen. (Bandcamp 2020c, viitattu 11.10.2020.) Tämän lisäksi on olemassa myös kuukausimaksullinen julkaisijoille suunnattu ”Bandcamp label

account”, joka erinäisten julkaisijatoiminnallisuuksien lisäksi tuo myös Pro-ominaisuudet kaikille kyseisen julkaisijan alaisille musiikkijulkaisuille (Bandcamp 2020d, viitattu 11.10.2020).

Useimmissa tapauksissa palvelussa oleviin musiikkijulkaisuihin voi kuitenkin tutustua kuuntelemalla kaikki kappaleet, joskin ilmaiset toistokerrat voivat olla kappalekohtaisesti rajattu. Tietyn kappaleen kohdalla täyttyneen toistorajoituksen voi kuitenkin kiertää yksinkertaisesti vaikkapa vierailemalla sivulla yksityisessä selainistunnossa, jolloin toistokertojen laskuri nollautuu.

Bandcamp tarjoaa mobiilisovelluksen sekä iOS- että Android -käyttöjärjestelmille. Tietokoneella käytössä on kuitenkin vain web-selaimessa toimiva Internet-palvelu. Musiikkipalvelun tarpeisiin pelkkä selainkäyttö on mielestäni riittämätön, ja näistä puutteista tämän lopputyön ohjelmointiprojekti sai alkunsa. Kyseiset Bandcampin Internet-palvelun puutteet ja niihin kaavaillut parannukset käsitellään omassa alaluvussaan 5.1.

3.2 Projektin kohdelaitteisto: Raspberry Pi.

Raspberry Pi (Rpi) on Raspberry Pi Foundation -hyväntekeväisyysjärjestön kehittämä yhden piirilevyn minitietokoneiden linja, jonka ensimmäinen malli, Raspberry Pi Model B, julkaistiin vuonna 2012. Perimmäinen ajatus oli luoda edullinen tietokone, joka tarjoaisi tietotekniikan mahdollisuuksia opetukseen ja kehittyviin maihin. Tuotteesta tuli paljon ennakoitua suositumpi erilaisten tietotekniikan projektien alustana ympäri maailman, ja ensimmäisenä julkaistusta, vain 256 megatavun muistilla varustetusta Rpi1 Model B:stä on kuljettu matka vuonna 2020 ilmestyneeseen Raspberry Pi 4 Model B -varianttiin 8 gigatavun keskusmuistilla. (Wikipedia 2020a, viitattu 13.10.2020.)

Kuvio 1. Raspberry Pi 3B, tulitikkuaski kuvassa laitteen koon havainnollistamiseksi.

Tässä projektissa oli käytössä Raspberry Pi 3 Model B, joka on vuonna 2016 julkaistu malli varustettuna yhden gigatavun keskusmuistilla. Kaikille Raspberry Pi -malleille on yhteistä se, ettei niissä ole valmiina minkäänlaista aktiivista jäähdytystä järjestelmäpiirille tai edes jäähdytysripaa. Tuulettimen puuttuminen tekee laitteesta täysin äänettömän, vaikkakin järeämpään käyttöön uusissa malleissa suositellaan jonkinlaisen lisäjäähdytyksen lisäämistä, joko passiivisena pelkän jäähdytysrivan muodossa tai aktiivisena pienen tuulettimen kanssa. Äänettömyys on erityisen hyödyllinen ominaisuus käytettäessä laitetta äänentoistoon, koska kaikenlaiset toimintaäänet häiritsevät etenkin hiljaisemmalla äänenvoimakkuudella kuunneltaessa.

Vaikka Raspberry Pi:n menestyksen siivittämänä markkinoille on ilmestynyt myös muita kilpailevia, pieniä ja edullisia yhden piirilevyn minitietokoneita, Raspberry Pi:lle löytyy kuitenkin vielä parhaiten ohjeita ja neuvoja, sekä eniten yhteensopivia ohjelmia ja ohjelmistoja. Virallinen käyttöjärjestelmä Rpi:lle on Debian Linux -jakelupakettiin pohjautuva Raspberry Pi OS, aiemmalta nimeltään Raspbian (Wikipedia 2020b, viitattu 13.10.2020). Tässä projektissa oli käytössä 32-bittinen, Debian Buster versioon perustuva Raspberry Pi OS.

3.3 Projektissa hyödynnetyt ohjelmakomponentit

Soitinkokonaisuuden ohjelmointiin käytettiin Python-ohjelmointikieltä, koska sille on saatavilla suuri määrä lisätoiminnallisuuksia tarjoavia kirjastoja ja lisäosia, ja ohjelmointikieli on suunnattu erityisesti nopeiden ohjelmointiprojektien toteuttamiseen. Python-koodia on myös mahdollista kirjoittaa ja suorittaa interaktiivisessa komentotulkissa, mikä nopeuttaa ohjelmien kehittämistä ja ohjelmointikielen opiskelemista.

Tässä osiossa listataan lyhyesti ohjelmakomponentteja, joita projektissa hyödynnettiin. Suurin osa niistä on Python-ohjelmointikielelle saatavilla olevia lisäosia tai kirjastoja, mutta myös muutama projektille tärkeä erillisohjelma mainitaan. Tämän osion esittelyt ovat lyhyitä, koska pääpainossa olevista verkkosivujen haravoinnissa käytettävistä Python-kirjastoista annetaan käyttöesimerkkejä projektin toteutuksesta kertovassa osiossa.

Beaufiful Soup on Python-kirjasto tietojen hakemiseen HTML- ja XML-tiedostoista. Beautiful Soup muuntaa HTML-koodin Python-objekteista koostuvaksi puurakenteeksi, josta voi sitten hakea haluamiaan tietoja HTML-tageja ja niiden ominaisuuksia määrittelemällä. (Richardson 2020, viitattu 13.10.2020.) BS:n hyödyntäminen projektissa jäi lopulta melko vähäiseksi, koska halutut tiedot löytyivät pikemmin Javascript-upotuksista kuin HTML-koodin sisältä.

JS2XML on Python-lisäosa, joka muuttaa Javascript-koodin XML-dokumentiksi. Tämän konvertoinnin jälkeen lisäosalla on mahdollista hakea tietoja käyttäen XPathia (XML Path Language), joka on helppokäyttöisempi vaihtoehto verrattuna esimerkiksi tietojen hakemiseen pelkkiä säännöllisiä lausekkeita käyttäen. (GitHub 2020, viitattu 13.10.2020.)

Nykyään on olemassa jo neljä eri versiota Xpath-kielestä: vuonna 1999 esitelty XPath 1.0, XPath 2.0, XPath 3.0 ja viimeisimpänä vuonna 2017 esitelty XPath 3.1. Eri versioiden välillä on huomattavia eroja, sillä varsinkin XPath 2.0 laajensi kieltä ja muutti sen perusteita mittavasti. Tämän vuoksi on syytä tarkentaa, että tässä työssä XPathilla viitataan XPathin 1.0 -versioon, jota JS2XML käyttää. Tietojen hakeminen XPath 1.0:aa käyttäen perustuu XML-dokumentin puumuotoisen rakenteen navigointiin polkulausekkeita käyttäen. (W3C 2020, viitattu 25.11.2020.) Koska suurin osa musiikkisoittimen tarvitsemista tiedoista oli saatavilla HTML-lähdekoodissa olevista Javascript-upotuksista, päädyttiin tässä projektissa käyttämään eniten juuri JS2XML:ää tarvittavien tietojen poimimiseen.

Pythonin re-moduuli, eli säännölliset lausekkeet ovat kirjoitusmerkeistä muodostettu hakumääritelmä, jonka avulla voidaan hakea tietyt ehdot täyttäviä kohtia tekstikokonaisuudesta. Säännöllisten lausekkeiden hyödyntäminen Pythonissa onnistuu sisäänrakennetun re-moduulin avulla. Säännölliset lausekkeet ovat käytettävissä useimmissa ohjelmointikielissä, mutta eri kielten välillä voi olla pieniä eroavaisuuksia toiminnoissa. Monimutkaisempien tehtävien suorittaminen säännöllisillä lausekkeilla voi myös johtaa hyvin vaikeaselkoisiin lausekkeisiin. (Kuchling 2020, viitattu 14.10.2020.)

Mpv (virallinen kirjoitusasu on kokonaan pienillä kirjaimilla, mpv) on ilmainen, avoimen lähdekoodin komentorivipohjainen mediantoisto-ohjelma, joka on saatavilla useille eri laitteisto- ja käyttöjärjestelmäalustoille. Se on forkattu mplayer2-ohjelmasta, joka taas oli forkki MPlayerista. Vaikka mpv:llä voi toistaa myös videoita, tässä projektissa sitä käytettiin vain äänitiedostojen toistamiseen. (Wikipedia 2020c, viitattu 14.10.2020.) Mpv on oma erillinen ohjelmansa, ja Python-ohjelmasta sitä ohjattiin python-mpv lisäosan avulla.

Arrow on Python-kirjasto päivämäärien ja kellonaikojen käsittelyyn. Arrow-kirjastoa käytettiin lähinnä päivämääräaikojen muuntamiseen ja vertailuun Unix-aikoina. (Smith 2020, viitattu 14.10.2020.) Unix-aika on useissa ohjelmissa käytetty aikaformaatti, joka ilmoittaa ajan kuluneiden sekuntien muodostamana kokonaislukuna, alkaen ajanhetkestä ensimmäinen tammikuuta 1970 kello 0.00.00 UTC (Wikipedia 2020d, viitattu 25.11.2020).

Tmux (Terminal MultipleXer) saattaa ehkä olla useille tuntematon ohjelma, mutta moni Linuxia käyttävä tuntee ainakin Screen-ohjelman, jonka avulla yhdestä fyysisestä pääteikkunasta voi käynnistää ja käyttää useampaa komentoriviohjelmaa jokaiselle ohjelmalle luodussa omassa virtuaalipääteikkunassa. Myös screen-ohjelma on englanninkielisesti ilmaistuna ”terminal multiplexer”, joka on jälleen yksi tietotekniikan termi, jolle ei tunnu löytyvän laajassa käytössä olevaa suomennosta. Tällaisten ohjelmien toimintaa voisi kuvailla vaikkapa nimityksellä ”virtuaalipäätehallintaohjelma”, mutta tässä työssä käytetään vakiintuneen suomennoksen puuttuessa englanninkielisen termin raakakäännöstä terminaalimultiplekseri.

Alussa mainitun toiminnallisuuden jatkoksi terminaalimultiplekseri tarjoaa myös mahdollisuuden avata ohjelmia sen hallinnoimaan istuntoon ja sitten irrottautua niistä fyysisessä pääteikkunassa. Tällöin avatut ohjelmat säilyvät terminaalimultiplekserin istunnossa, johon voi myöhemmin

kiinnittyä uudestaan ja jatkaa työskentelyä, tarvittaessa jopa useasta eri tietokoneesta, pääte-emulaattoria ja SSH-yhteyttä käyttäen. Tässä projektissa käytettiin tmux-terminaalimultiplekseriä, koska se sisältää vielä pieniä lisäominaisuuksia tutumpaan screeniin verrattuna. (McKay 2020, viitattu 14.10.2020.)

AIOHTTP on asynkroninen HTTP-asiakas sekä HTTP-palvelin Python-ohjelmointikielelle. AIOHTTP hyödyntää Pythonin rinnakkaiseen (engl. concurrent) ohjelmointiin käytettävää asyncio-kirjastoa. AIOHTTP pystyy tekemään HTTP-pyyntöjä samoin kuin seuraavana esittelyvuorossa oleva Requests, mutta siitä poiketen se tukee myös rinnakkaisuutta toiminnoissaan, joten sillä voi tehdä useamman pyynnön samanaikaisesti. Requests sen sijaan etenee pyyntö kerrallaan, odottaen niiden valmistumisen ennen seuraavan pyynnön tekemistä. Tämän vuoksi musiikkijulkaisusivujen noutamisessa käytettiin AIOHTTP:tä. Koska projektiohjelma noutaa useita kymmeniä musiikkijulkaisusivuja peräkkäin, tuo AIOHTTP:n rinnakkaisuus merkittävää ajansäästöä Requests-kirjaston käyttämiseen verrattuna. (Uriegas 2017, viitattu 14.10.2020.)

Requests on Python-ohjelmointikielen HTTP-kirjasto, jolla voi helposti tehdä HTTP/1.1-pyyntöjä (Reitz 2020, viitattu 14.10.2020). Projektissa sitä käytettiin esimerkiksi MP3-tiedostojen lataamiseen, koska projektiohjelma noutaa äänitiedostoja vain yksi kerrallaan, ja Requests on AIOHTTP:tä yksinkertaisempi käyttää juuri rinnakkaisuuden puuttumisen vuoksi.

4 PROJEKTIN EETTINEN NÄKÖKULMA

Siinä missä jo pelkkä WWW-sovelluspalvelun olemassaolo viestii oikeudesta sillä saatavien tietojen käyttämiseen, verkkosivujen haravointiin perustuvat sovellukset lepäävät huomattavasti huterammalla pohjalla eettisten ja lainsäädännöllisten kysymysten suhteen. Projektin kohteena oleva Bandcamp rahoittaa toimintansa myymällä palvelussa olevaa musiikkia, joten sivujen toiminnallisuus on suunniteltu tätä tarkoitusperää palvelevaksi. Oma soittimeni lataa sivujen kautta ilmaiseksi kuunneltavissa olevat musiikkitiedostot ja mahdollistaa niiden soittamisen normaalissa äänentoisto-ohjelmassa. Vaikka kyseiset musiikkitiedostot ovatkin sivuilla ilmaiseksi kuunneltavissa, voi niiden kuunteleminen palvelun sivujen kautta olla rajoitettu tiettyihin soittokertoihin, minkä jälkeen niiden toistaminen pysähtyy pop-up-ikkunaan, jossa kehotetaan tukemaan artistia ja ostamaan kyseessä oleva albumi tai kappale.

Tämän rajoituksen voi kuitenkin helposti kiertää menemällä kyseiselle sivulle vaikkapa Internet-selaimen yksityinen selaus -toimintatilassa, jossa selain ei tallenna eikä käytä mitään vieraillulle sivulle kuuluvia väliaikaistiedostoja. Täten myöskään soittokertojen rajoitus ei toimi. Uusia selausistuntoja avaamalla voit siis käytännössä kuunnella samaakin kappaletta loputtomia kertoja.

Joten vaikka tässä projektissa toteutetulla musiikkisoittimella voi soittaa sivustolta ladattuja musiikkitiedostoja rajoituksetta, niin samoin voi tehdä myös edellä mainittua tapaa käyttäen. Lisäksi selaimen yksityistä tilaa käyttäessä palvelu joutuu toimittamaan musiikkitiedoston jokaiselle istunnolle uudelleen, joten tähän verrattuna projektin ohjelma itse asiassa rasittaa palvelua vähemmän. Samoin myöskään palvelun sivuja ei ladata kuin yhden kerran ohjelmaa käyttäessä, liikkuipa eri musiikkijulkaisujen välillä kuinka paljon tahansa. Ilmaiseksi kuunneltavissa olevat musiikkitiedostot ovat myös ainoastaan 128 Kb/s bittinopeuden mp3-tiedostoja, joten ne eivät ole yhdenvertaisia ostettaessa saataviin korkeamman laadun musiikkitiedostoihin. Korkein saatavilla oleva äänenlaatu ostamiasi musiikkitiedostoja ladatessa on häviötön, esimerkiksi FLAC-audiotiedosto, mutta muitakin vastaavia häviöttömiä audioformaatteja on tarjolla (Bandcamp 2020f, viitattu 14.10.2020).

Asiaa sivuaa myös Bandcampin Audio FAQ -osuudesta löytyvä vastaus kysymykseen, joka vapaasti suomennettuna kuuluu ”Kuulin että Bandcampin sivuilta voi helposti ’varastaa’ musiikkia useitakin keinoja käyttäen, mitä aiotte tehdä tämän asian suhteen”, johon todetaan ”etteivät he aio

tehdä asian suhteen mitään, ja että muutamat näin tekevät ’halpamaiset’, mutta innokkaat, ja mahdollisesti musiikkia ’puskaradiossa’ mainostavat kuuntelijat kannattaa pikemminkin nähdä mahdollisuutena, kuin uhkana” (Bandcamp 2020e, viitattu 14.10.2020).

Tärkeimpänä seikkana tämän projektin eettisyyttä pohtiessa halutaan kuitenkin vielä painottaa sitä, ettei ohjelmaa ole tehty minkäänlaista laajamittaista taikka kaupallista käyttöä varten. Sen toteuttaminen palveli ainoastaan uusien asioiden opiskelemista tarjoamalla mielenkiintoisen reaalimaailman kehyksen erilaisten Python-työkalujen ja tiedonharavoinnin hyödyntämiseen. Projektilla ei myöskään haluta edistää minkäänlaista tekijänoikeuksien loukkaamista, ja projektiin liittyvät eettiset näkökulmat on tiedostettu alusta lähtien.

5 OHJELMOINTIPROJEKTIN ETENEMISKERTOMUS

Tässä osiossa kerron ohjelmointiprojektin toteutuksesta ja etenemisestä. Koska kyseessä oli kokonaan oma projektini, tunsin luontevaksi ratkaisuksi käyttää tässä osiossa ensimmäistä persoonaa. En nähnyt mitään syytä häivyttää asioiden alkuperää epämääräisemmällä muodolla, koska olen kuitenkin itse vastuussa kaikista projektin päämääristä ja päätöksistä.

Alaluku 5.2 tarjoaa tiedonharavointiin keskittyvän yleiskatsauksen ohjelmointiprojektin etenemisestä, jotta lukija voi paremmin sijoittaa kyseistä osiota seuraavat tarkemmat tiedonharavointikomponenttien käytöstä kertovat esimerkit projektikokonaisuuteen. Tiedonharavoinnin ulkopuolelle jääviä ohjelmakomponentteja, kuten esimerkiksi mediantoisto-ohjelman mpv:n käyttöä, ei siis käsitellä tässä yleiskatsauksessa eikä sitä seuraavissa esimerkeissäkään.

5.1 Bandcamp-palvelun käyttökokemuksen puutteet ja suunnitellut parannukset

Kuten tietoperustassa jo todettiin, Bandcamp tarjoaa mobiilisovelluksen Android- ja iOS-käyttöjärjestelmille. Tietokoneella käytössä on kuitenkin vain selainpohjainen Internet-palvelu, joka on mielestäni riittämätön musiikkipalvelun sujuvaan käyttämiseen. Tässä osiossa listaan omasta mielestäni pahimpia puutteita kyseisessä käyttöliittymässä sekä Python-pohjaiseen projektiohjelmaan kaavaillut parannukset.

Ensiksi haluan kuitenkin selventää sanan ”albumi” käyttämistä tässä työssä. Bandcampissa olevat kappaleet ovat useimmissa tapauksissa osa jotain suurempaa musiikkikokonaisuutta, jota kutsutaan yleisesti ottaen albumiksi. Myös rajoitetumpia EP-julkaisuja kutsutaan palvelun sisällä sanalla ”album”. Tämän lisäksi on kuitenkin olemassa myös yksittäisiä kappaleita, jotka voivat olla listattuna erillisinä musiikkijulkaisuina julkaisijan Music-sivulla. Tässä työssä puhuttaessa albumisivusta tai albumista viitataan siis oman sivunsa alta löytyvään musiikkijulkaisukokonaisuuteen, olipa se sitten albumi tai yksittäinen kappale. Projektiohjelma ei tee niiden välillä eroa, ja jokaisella kerralla termin ”musiikkijulkaisukokonaisuus” tai ”musiikkijulkaisu” käyttäminen kuulostaisi turhan kömpelöltä.

Ongelma 1: Vaivalloinen liikkuminen albumisivujen välillä

Jokainen albumi on Bandcampissa erillinen verkkosivu, ja niiden välillä siirtyminen tapahtuu normaalista Internet-selaamisesta tutulla tavalla, napsauttamalla internet-linkkiä. Jos ajatellaan vaikkapa tietyn julkaisijan alla olevia albumeita, niin niiden läpikäymiseksi kuuntelijan on ladattava kaikki julkaisijan albumit listaava Internet-sivu, ja sitten yksitellen napsautettava ja ladattava jokaisen albumin omasta linkistä avautuva erillinen albumisivu. Tämä myös tarkoittaa, että edestakainen liikkuminen vaatii käytännössä aina uuden sivulatauksen tai uuden auki jätettävän välilehden avaamisen, joka taas vie tietokoneen resursseja ja levittää käyttöliittymäkokonaisuuden usean erillisen selainsivun alle. Tämä tekee julkaisijan diskografiaan tutustumisesta uutta musiikkia etsiessä työlästä ja hidasta.

Ongelma 1: Projektiin suunniteltu parannus

Musiikintoisto siirretään oikean mediantoisto-ohjelman tehtäväksi (mpv), ja julkaisijan albumisivuilta haravoidut tiedot tallennetaan pysyvään dictionary-tietorakenteeseen. Tämän ansiosta eri albumien välillä liikkuminen tapahtuu välittömästi, ilman toistuvaa ja turhaa muuttumattomien verkkosivujen lataamista jokaisen siirtymän yhteydessä. Julkaisijan kaikkien albumien välillä voi myös siirtyä edestakaisin yksinkertaisella komentorivikäskyllä.

Ongelma 2: Tarve tietokoneen päällä pitämiseen musiikkia kuunnellessa

Jos aikeissasi on pelkkä musiikin kuunteleminen, ilman että samalla työskentelet tietokoneella, tuntuu tietokoneen käynnissä pitäminen vain musiikin toistamiseksi ylimitoitetulta. On todennäköistä, että tietokone pitää ainakin jonkinlaista ääntä päällä ollessaan, ja tämä tietysti haittaa musiikin kuuntelua erityisesti hiljaisemmilla äänenvoimakkuuksilla. Lisäksi tietokoneiden, varsinkin järeämpien pöytäkoneiden, sähkönkulutus vaihtelee suuresti, ja etenkin pelkkänä musiikkisoittimena toimiessa se voi tehtävän vaativuuteen suhteutettuna olla hyvinkin korkea.

Ongelma 2: Projektiin suunniteltu parannus

Python-ohjelman kohdelaitteeksi valittiin Raspberry Pi 3B -minitietokone, laitteen pienen sähkönkulutuksen ja aktiivisen jäähdytyksen puuttumisesta johtuvan täydellisen äänettömyyden vuoksi. Enemmän tietoa Raspberry Pi:stä löytyy laitteen omasta alaluvusta 3.2.

Ongelma 3: Musiikin hallinta on sidottu äänentoistosta vastaavaan laitteeseen

Koska musiikkipalvelun käyttö tapahtuu Internet-sivujen kautta, musiikintoistoa ei voi ohjata esimerkiksi kannettavasta tai mistään muustakaan ulkopuolisesta laitteesta. Hallinta onnistuu ainoastaan samalta laitteelta johon kaiuttimet tai vastaavat äänentoistolaitteet on liitetty. Tässä projektissa toistolaitteina toimivat ulkoiseen USB-äänikorttiin liitetyt aktiivikaiuttimet.

Ongelma 3: Projektiin suunniteltu parannus

Raspberry Pi:n Linux-käyttöjärjestelmässä suoritettavan Python-ohjelman ohjaaminen onnistuu mistä tahansa SSH-yhteyksiä tukevasta pääte-emulaattorista, kuten vaikkapa Windows-käyttöjärjestelmän PuTTy-ohjelmasta. Kun minitietokoneessa suoritettava ohjelma käynnistetään vielä terminaalimultiplekserin (tmux) sisällä, voidaan sitä jopa hallita useammasta laitteesta yhtä aikaa, sekä vapaasti katkaista ja palauttaa hallintayhteys.

Halutut lisäominaisuudet ovat siis lähinnä asioita, joiden koen tekevän uuden musiikin löytämisestä helpompaa ja suoraviivaisempaa poistamalla musiikinkuuntelua häiritseviä manuaalisia toimenpiteitä läpikäytäessä suurta määrää albumeita tietyn julkaisijan diskografiasta. Tärkeää oli myös siirtää musiikintoisto äänettömän ja vähemmän sähköä kuluttavan, jatkuvasti päällä pidettäväksi soveltuvan ratkaisun piiriin.

5.2 Ohjelmointiprojektin yleinen etenemiskertomus

Ensimmäinen askel tiedonharavoinnissa on tutustuminen halutut tiedot sisältävään aineistoon, tässä tapauksessa Bandcamp-musiikkipalvelun muodostaviin verkkosivuihin ja niiden rakenteen selvittämiseen haluttujen tietojen poimimisen automatisoimista varten. Tätä tarkoitusta varten tutustuin ensimmäiseksi Bandcampin yksittäisen albumisivun rakenteeseen. Käytin ensimmäiseksi nykyään miltei kaikkien web-selaimien kehitystyökaluista löytyvää Inspector-toimintoa, jolla voidaan siirtyä tiettyä selainnäkymän kohtaa napsauttamalla vastaavaan kohtaan HTML-lähdetekstissä. Tämä on hyvin käytännöllinen työkalu, kun esimerkiksi halutaan selvittää, millaisten eri HTML-elementtien alla tarvittava teksti sijaitsee.

Heti ensimmäiseksi halusin varmistua siitä, että Bandcampin musiikkijulkaisusivuilla veloituksetta kuunneltavissa olevat musiikkitiedostot löytyvät jossain muodossa, jota erillinen mediantoisto-ohjelma, tämän projektin tapauksessa mpv, pystyy toistamaan verkkosivuston ulkopuolella. Olin

alustavasti päätellyt tämän olevan mahdollista, mikä johtui Bandcampin antamasta vastauksesta luvussa 4. mainittuun ”Audio FAQ”-kysymykseen.

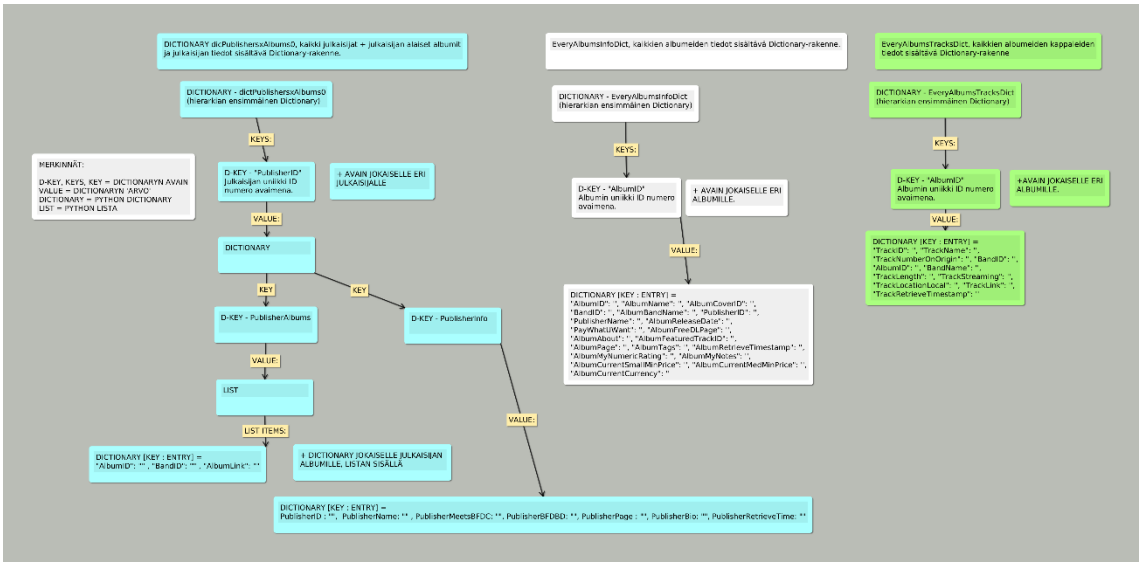
Tutkittuani albumisivun lähdekoodia jonkin aikaa, huomasin että siitä löytyy Javascript-upotus, jossa on myös linkki kyseiseen vastaukseen. Kyseistä koodia tarkemmin tutkimalla kävi ilmi, että siitä löytyisi myös suurin osa kehitettävän musiikkisoittimen tarvitsemista musiikkitiedoista, ja tärkeimpänä generoidut linkit kyseisen musiikkijulkaisun 128 Kb/s bittinopeuden mp3-tiedostoihin. Ajattelin, että minun kannattaisi poimia kaikki mahdolliset tiedot tästä Javascript-koodista, koska se olisi todennäköisesti kaikkialla samanlainen ja sisältäisi varmimmin kaikki halutut tiedot jokaisen sivun tapauksessa. Lisäksi tiedot olisivat yhdessä paikassa eikä niitä tarvitsisi metsästää monesta kohdasta ympäri sivun HTML-lähdekoodia.

Tähän asti olin ajatellut, että suurin osa tiedoista tulisi poimittua Beautiful Soup -lisäosalla, mutta en oikein osannut hyödyntää sitä tietojen poimimiseen Javascript-koodista. Ryhdyin etsimään lisätietoa hakusanoilla ”scraping Javascript python” ja löysin Python-lisäosan nimeltä JS2XML. Kyseinen lisäosa mahdollistaa Javascriptin muuttamisen XML-dokumentiksi ja haluttujen tietojen noutamisen dokumentista XPath-polkulausekkeita käyttäen. En kuitenkaan ollut ikinä aiemmin käyttänyt XPathia, joten vuorossa oli toimivien sijaintipolkujen selvittäminen yrityksen ja erehdyksen kautta. Tallensin JS2XML:n XML-muotoon muuttaman Javascript-koodin tekstitiedostoon, tutustuin sitten sen rakenteeseen ja hain internetistä lisätietoja. Tarkempaa tietoa JS2XML:n hyödyntämisestä löytyy alaluvusta 6.5.

Siirryttyäni poimimaan musiikkitiedot Javascript-koodista JS2XML:ää käyttäen, jäi Beautiful Soup -kirjaston ainoaksi tehtäväksi tällä saralla Javascript-koodiupotusten hakeminen. Se ei enää tuntunut optimaaliselta työkalulta tähän tehtävään, koska koko albumisivun lähdekoodin Beautiful Soup -parsinta vei kuitenkin aikaa muutaman sekunnin jokaista sivua kohden. Yksinkertaiseen toimenpiteeseen kuluisi siis suhteettoman paljon aikaa käsiteltäessä kaikki julkaisijan alaiset musiikkijulkaisusivut. Niinpä siirryin tapaan, jossa ensiksi haetaan kaikki Javascript-upotukset säännöllisellä lausekkeella ja sitten paikallistetaan niistä oikea hakemalla tiettyä osamerkkijonoa, joka ei esiinny muualla kuin halutussa Javascript-upotuksessa. Tämä menettely tuntui huomattavasti nopeammalta kuin aikaisempi Beautiful Soup -parsinnan sisältänyt toimenpide, vaikken tehnytkään tarkkaa vertailua näiden kahden tavan vaatiman ajan välillä.

Beautiful Soup ei kuitenkaan jäänyt kokonaan pois projektista, koska sillä poimittiin musiikkijulkaisujen listaustiedot julkaisijoiden Music-sivuilta. Näistä Music-sivulta poimittavista tiedoista tärkein on tietysti jokaisen musiikkijulkaisun oma Internet-osoite, jota AIOHTTP käyttää musiikkijulkaisusivun noutamiseen. Lopulta Beautiful Soupin tehtäväksi jäi myös genre-avainsanojen poimiminen näiltä musiikkijulkaisusivuilta. Luonnollisesti jo aiemmin mainittu musiikkijulkaisusivujen hidas prosessointi haluttiin välttää myös tässä yhteydessä, joten tämän toiminnon vaatima Beautiful Soup parsinta suoritettiin säännöllisillä lausekkeilla eristetylle, tarkasti rajatulle lähdetekstiosiolle. Lisää sekä AIOHTTP:n että Beautiful Soupin käyttämisestä kerrotaan molempien omissa käyttöesimerkkiluvuissa.

Koska poimitut tiedot oli tietysti myös tallennettava johonkin, ryhdyin miettimään niille sopivaa tietorakennetta, ja päädyin lopulta kolmen erillisen Python dictionaryn muodostamaan tietokokonaisuuteen. Päädyin tähän osittain siitä syystä, että suunnitelmissani oli jossain kehitysvaiheessa lisätä soittimen ominaisuuksiin myös tietojen tallentaminen SQLite-tietokantaan, ja tätä varten nämä kolme dictionarya sisältäisivät myös dictionaryt, joiden rakenne peilautuisi suoraan kolmen identtisen tietokantataulun kanssa. Täten tietojen tallentaminen ohjelmasta tietokantaan, kuten myös vastaavasti tietojen hakeminen tietokannasta ohjelman käyttöön, olisi mahdollisimman yksinkertainen toimenpide. Alla oleva kuva esittää projektiohjelman käyttämän tietorakenteen, ja siitä voi nähdä myös kaikki ohjelman käsittelemät musiikkitiedot.



Kuvio 2. Projektiohjelman käyttämä, kolmen Python dictionaryn muodostama tietorakenne.

6 TIEDONHARAVOINTIIN LIITTYVIEN PYTHON-KIRJASTOJEN KÄYTTÖ PROJEKTISSA

Tässä osiossa esittelen projektissa käytettyjen Python-kirjastojen toimintaa omien havaintojeni ja käyttöesimerkkien muodossa. Käytin projektiohjelmassa monenlaisia Pythonille saatavilla olevia lisäosia ja kirjastoja, jopa puhesyntetisaattorista lähtien (soitettavan kappaleen tietojen kuuluttamiseen ennen toiston alkamista), mutta järkevän kokonaisuuden saavuttamiseksi esimerkit rajataan pelkästään tiedonharavointiin liittyviin ohjelmakomponentteihin. Esimerkit ovat siis joko sisällön hakemiseen käytettävistä kirjastoista, kuten Requests tai AIOHTTP; tai tiedon poimimiseen käytettävistä komponenteista, kuten re (säännölliset lausekkeet), Beautiful Soup ja JS2XML.

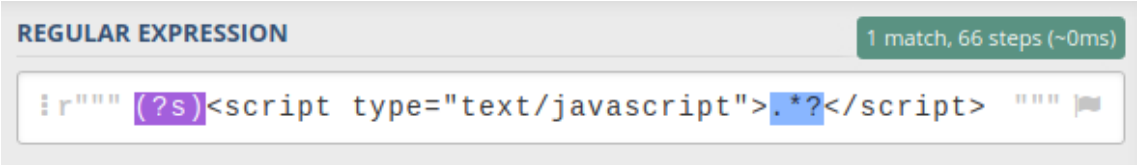
Esimerkeissä esiintyvät koodikatkelmat on merkitty lainaamalla ne (´ **)** merkkien sisälle, tyyliin *´teksti = ”tekstisisältö”´*. Näin on tehty jotta niissä mahdollisesti esiintyvät normaalit (’ ”) lainausmerkit eivät aiheuttaisi sekaannusta. Esimerkeissä ei myöskään käsitellä käytettyjen Python-kirjastojen asentamista, mutta se oli jokaisen kohdalla helppo toimenpide Pythonin PIP-paketinhallintaohjelmaa käyttäen.

6.1 Re-moduuli (Pythonin säännölliset lausekkeet)

Tulin käyttäneeksi projektissa säännöllisiä lausekkeita melko vähän, ja enimmäkseen suuripiirteisempään lähdeaineiston esikäsittelyyn, jonka jälkeen varsinainen yksittäisten tietojen poimiminen tapahtui muita keinoja käyttäen. Jätin säännölliset lausekkeet myös tarkoituksella mahdollisimman yksinkertaiseksi, jotta niiden hahmottaminen ja muodostaminen ei vaatisi liikaa aikaa. Vianetsimisen kannalta en myöskään halunnut kasata yhteen toimintoon liiallista virhepotentiaalia.

Säännöllisten lausekkeiden käyttöesimerkki, Javascript-upotusten etsiminen.
AIOHTTP:n noutamilta musiikkijulkaisusivuilta poimin tietoja Javascript-upotuksesta JS2XML:ää käyttäen, ja ensimmäisenä vaiheena tätä prosessia käytin säännöllisiä lausekkeita kaikkien musiikkijulkaisusivulla esiintyvien Javascript-koodien löytämiseen. Tämä tehtävä onnistui seuraavalla säännöllisellä lausekkeella:

(?s)<script type="text/javascript">.?</script>*



Kuvio 3. Käyttöesimerkin säännöllinen lauseke online-testaajassa.

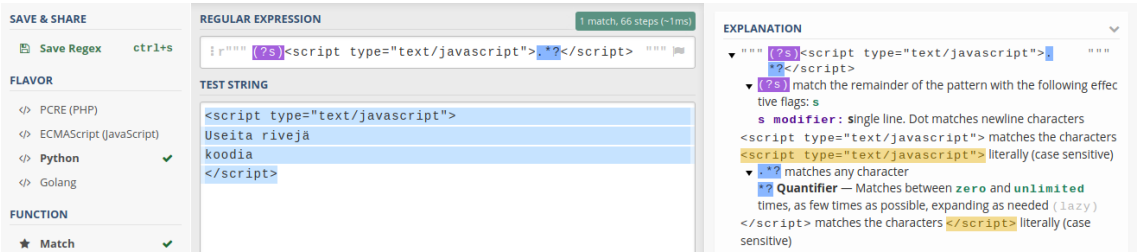
´(?s)´: Tämä osa säännöllistä lauseketta muuttaa kyseisessä lausekkeessa esiintyvien ´**.**´ erikoismerkkien toimintaa niin että ne tuottavat osuman jokaisen merkin, normaalitoiminnasta poiketen myös rivinvaihdon kohdalla. Se tarvitaan koska Javascript-koodi saattaa pitää sisällään myös rivinvaihtoja.

´<script type="text/javascript">´: Tämä on sitä mitä sen voi ehkä arvatakin olevan, tekstiä joka tuottaa osuman identtisen osamerkkijonon kohdalla. Haluttu Javascript-upotus alkaa aina tällä tekstillä, joten sen alkutagi voidaan yksinkertaisesti etsiä tällaisella ”staattisella” osamerkkijonolla.

´.?´*: ´**.**´ mikä tahansa kirjoitusmerkki, ´***´ edellisen kirjoitusmerkin osumia niin monta kertaa kuin mahdollista, ja ´**?**´ eli ”Non-greedy” tai ”minimal matching”, mikä tarkoittaa että ehdot täytetään mahdollisimman vähillä merkeillä.

´</script>´: Sama mitä alkutaginkin kohdalla, eli kyseisen osamerkkijonon kohdalla täyttyvä sääntö, kaikissa upotuksissa samanlaisena esiintyvän lopputagin löytämiseksi.

Eli kokonaisuutena käsitettynä, kyseessä oleva säännöllinen lauseke palauttaa osuman (osamerkkijonon), joka täyttää seuraavat ehdot: alkaa osamerkkijonolla *´<script type="text/javascript">´*, jonka jälkeen voi olla mitä tahansa merkkejä (mukaan lukien rivinvaihtoja) niin monta, mutta kuitenkin myös niin vähän kuin mahdollista (non-greedy), jotta kokonaisuus täyttää myös viimeisen ehdon, eli päättyy osamerkkijonoon ´</script>´.



Kuvio 4. Säännöllisen lausekkeen tuottama osuma online-testaajassa https://regex101.com/.

Tämän lausekkeen tulokseksi tulee sivun lähdekoodiin upotettu yksittäinen Javascript-koodiosio, aloittavan ja lopettavan HTML-tagin kanssa.

Pythonin re-moduulissa tätä säännöllistä lauseketta voi käyttää seuraavalla tavalla:

```
import re
hakulause = re.compile(""""(?s)<script type="text/javascript">.*?</script>"""")
tulokset = hakulause.findall(ownList[0])
```

Ensiksi tuodaan re-moduuli Pythonin käyttöön (´*import re*´), ja sitten tehdään säännöllinen lauseke -objekti ´hakulause´ haluamallamme säännöllisellä lausekkeella (´*hakulause = re.compile(""""(?s)<script type="text/javascript">.*?</script>"""")*´). Tällä objektilla voi sitten suorittaa useita kyseistä hakulausetta käyttäviä toimintoja, tässä tapauksessa haetaan kaikki ei-päällekkäiset säännöllisen lausekkeen osumat ´tulokset´ listaan (´*tulokset = hakulause.findall(ownList[0])*´). ´*ownList*´ on Python-lista, ja se on jatkoa AIOHTTP esimerkin puolelta, missä siihen haetaan musiikkijulkaisusivujen lähdetekstit. Jokaisessa listaindeksissä on siis eri musiikkijulkaisusivun lähdeteksti.

Haettuani näin yhden musiikkijulkaisusivun osumat listaan, valitsin niistä oikean, albumitiedot sisältävän osuman yksinkertaisesti hakemalla kyseisistä teksteistä ainoastaan halutussa koodiosumassa esiintyvää osamerkkijonoa "var BandData". JS2XML parsintaan kelpaavan puhtaan Javascript-koodin saamiseksi tekstistä piti vielä poistaa HTML-lähdetekstiin kuuluva aloittava ja lopettava tagi, eli ´<*script type="text/javascript"*>´ ja ´</script>´. Tämän koko toimenpiteen olisi voinut tehdä myös pelkällä yksittäisellä säännöllisellä lausekkeella, mutta koska pyrin pitämään säännölliset lausekkeeni mahdollisimman yksinkertaisina, tein tämän osumien jatkokäsittelyn useammassa askeleessa, Pythonin string-metodeja käyttäen. Näiden toimenpiteiden jälkeen lopputulokseksi saadun Javascript-tekstin käsittely JS2XML:llä käydään puolestaan läpi JS2XML esimerkkinä.

Säännölliset lausekkeet ladattujen sivujen varmentamisessa

Käytin säännöllisiä lausekkeita myös ladattujen sivujen varmentamiseen, koska yhtäaikainen sivujen noutaminen AIOHTTP:llä aiheutti välillä pyyntöjen epäonnistumisen 503-virhesivuun.

Ryhdyttyäni etsimään tapaa erottaa oikein noudettu sivu mahdollisista 503-virhesivuista, huomasin että musiikkijulkaisusivujen lopusta löytyi aina seuraavanmuotoinen kommenttiteksti:

<!-- bender25-6 Sat Aug 22 14:04:42 UTC 2020 -->

<!-- album id 2779615956 -->

Koska album-ID on tärkeimpiä (ja aikaisimpia) projektiohjelman käyttämiä tietoja, sen tarkastaminen albumisivujen lähdekoodista löytyvästä kommentista kertoisi nopeasti, jos kyseessä olisi oikein vastaanotettu sivu, eikä aiemmin mainittu 503-virhesivu. Lisäksi tällä menettelyllä sai samalla tiedon siitä minkä albumin (tai kappaleen) sivu oli kyseessä. Tämä oli tarpeellista, koska AIOHTTP:n hakemien sivujen lähdeteksti yksinkertaisesti tallennettiin Python-listaan sitä mukaa kun ne vastaanotettiin, ja yhtäaikaisuudesta johtuen niiden järjestys määräytyy siitä kuinka nopeasti erilliset lähetetyt pyynnöt valmistuvat. Koska album-ID:n sisältävä kommentti on myös aina viimeinen sivun lähdekoodissa oleva osamerkkijono, voitiin sen etsiminen rajata vain 100 viimeisen merkin joukkoon, toiminnon nopeuttamiseksi, ja mahdollisimman tarkaksi kohdentamiseksi.

Säännölliset lausekkeet optimoimiskäytössä

Tietojen poimintaan käyttämäni Javascript-upotus sisälsi myös valuuttaosion, mikä listaa kyseisen albumin taikka kappaleen hinta- ja valuuttatietoja kaikissa mahdollisissa Bandcampin tukemissa valuutoissa. Itseäni kiinnosti tästä vain ”Current”-osan alla olevat, vierailijan geografisesta sijainnista riippuvaiset, hänelle käytössä olevat hinta- ja valuuttatiedot. Valuuttaosio kokonaisuudessaan oli Javascript-koodin suurin osa, joten päädyin poistamaan sen säännöllisten lausekkeiden avulla, jättäen jäljelle vain mainitun ”Current”-osan, ennen koodin luovuttamista JS2XML:n käsiteltäväksi.

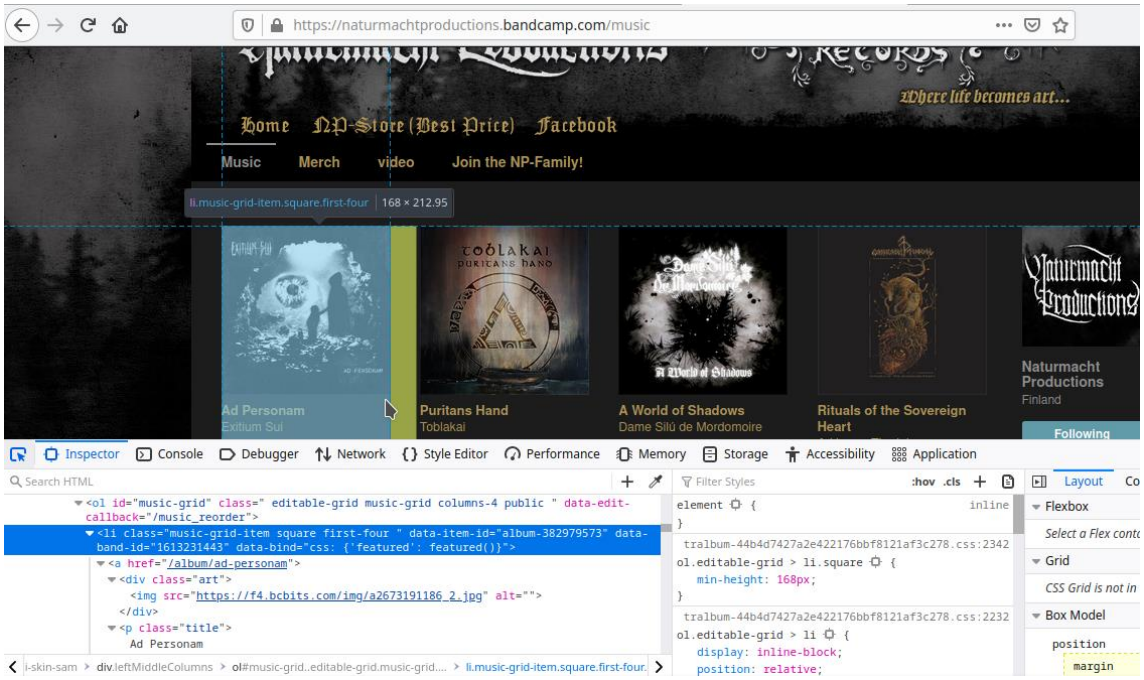
Käyttäen satunnaista albumisivua vertailukohtana, Javascript-upotuksen tiedostokoko käsittelemättömän valuuttaosion kanssa oli tekstitiedostoksi tallennettuna 49.5KB, kun taas pelkästään Current-osion sisältävälle optimoidulle tiedostolle kertyi kokoa vain 10.1KB. Tämän optimoinnin myötä JS2XML parsinnan vaatima aika puolestaan laski ~2.5 sekunnista ~0.5 sekuntiin, vain viidesosaan alkuperäisestä.

6.2 Beautiful Soup

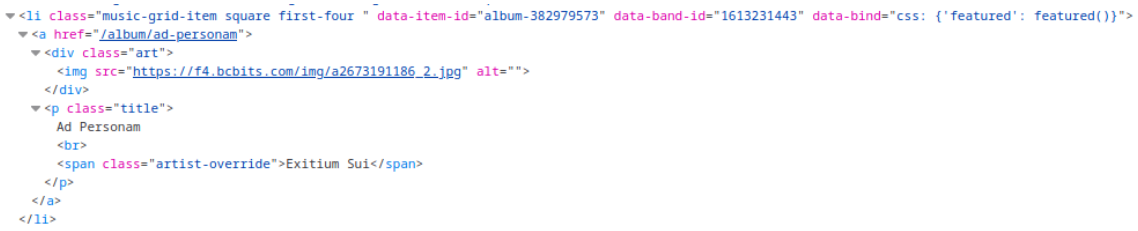
Projektin alkuvaiheessa ajattelin, että Beautiful Soup tulee olemaan eniten käyttämäni Python-kirjasto tietojen poimimiseen. Sen käyttö jäi kuitenkin paljon alussa arvioitua pienemmäksi, koska lopulta käytännössä kaikki tiedot poimittiin Javascript-upotuksesta, eikä HTML-elementtien sisältä. Beautiful Soup -kirjaston tärkeimmäksi tehtäväksi jäi julkaisijoiden Music-sivuilla listattavien musiikkijulkaisusivujen tietojen poimiminen (tärkeimpänä internet-osoite), ja tämä toimenpide käydään käyttöesimerkkinä läpi seuraavaksi.

Beautiful Soup käyttöesimerkki, Music-sivun tietojen poiminta.

Music-sivulta projektiohjelman jokaisen julkaisun kohdalta tarvitsemat tiedot olivat musiikkijulkaisun internet-osoite, ja Bandcampin käyttämät ”AlbumID” ja ”BandID” numerotunnisteet. AIOHTTP käyttäisi täältä saatavia osoitteita sivujen lataamiseen, ja id-numerot olisivat myös omalle projektiohjelmalle tärkeitä identifiointitapoja musiikkijulkaisujen käsittelyssä.



Kuvio 5. Julkaisijan musiikkijulkaisulistauksen sisältävä Bandcamp-sivu (eli Music-sivu), Firefoxin sisäänrakennetut ”Web-työkalut” avattuna.



Kuvio 6. Ensimmäisen albumin listaustiedot sisältävän li-elementin lähdeteksti Firefoxin Inspector-työkalulla tarkasteltuna. Sama albumi mitä edeltävässä kuvassa.

Kuten kuvassa 5. olevasta Firefox Inspector -näkymästä voidaan nähdä, yksittäisen albumin listaustiedot löytyvät html-tagin ”li” alta. Kyseisen li-tagin määreitä (engl. attribute) ovat luokka (engl. class) ´music-grid-item´, sekä ´data-item-id´ ja ´data-band-id´. Tutkimalla sivun lähdetekstiä laajemmin, voidaan varmistua että kaikki li-elementit luokkaa ´music-grid-item´ ovat todellakin erillisiä julkaisijan sivulla listattuja musiikkijulkaisuja. Hakemalla täältä löydettyjä ´data-band-id´ ja ´data-item-id´ määreiden arvoja *kyseessä olevan musiikkijulkaisun* verkkosivun lähdetekstistä, puolestaan selviää että ´data-band-id´:n arvo ´1613231443´ on tarvitsemamme ”Band ID”, ja ´data-item-id´:n arvo sisältää ”Album ID” -numerosarjan. Joskin tarkasti ottaen tässä esimerkissä ´data-band-id´ on itse asiassa julkaisijan identifioiva ”Publisher ID” numerosarja. Se onko tämä arvo ”Band ID”, vaiko ”Publisher ID”, riippuu kyseessä olevasta julkaisijasta; ja asianlaita voidaan selvittää tarkistamalla, onko numero sama jokaisen albumin kohdalla.

Koska jokainen li-tagi luokkaa ´music-grid-item´ sisältää tarvittavia tietoja, voidaan tiedonpoiminta aloittaa hakemalla ne kaikki Python-listaan. Tämä voidaan tehdä, kunhan ensiksi tuodaan ”BeautifulSoup” moduuli BeautifulSoup4 kirjastosta komennolla: *´from bs4 import BeautifulSoup´*, ja parsitaan Requestsin hakema julkaisijan verkkosivu komennolla *´parsittuHTMLsisältö_Music = BeautifulSoup(requestsnoudettusivuMusic.content, ’lxml’)´*. Edellä olevassa esimerkissä käytetään erikseen asennettavaa lxml-parseria, joka on nopeampi mitä Pythonin sisäänrakennettu HTML-parseri.

Näiden alkutoimien jälkeen voidaan käyttää Beautiful Soupin Select-metodia, joka mahdollistaa CSS-valitsimien (engl. CSS Selector) käyttämisen halutun HTML-elementin valitsemiseen. Komento, joka hakee listaan haluamamme ´music-grid-item´ luokan li-tagit (sekä niiden alaisen sisällön) Beautiful Soupin parsimasta HTML-koodista ´parsittuHTMLsisältö_Music´, näyttää seuraavalta:

lista = parsittuHTMLsisältö_Music.select("li.music-grid-item")

Näin saadaan lista Beautiful Soupin Tag-objekteja, joista jokainen pitää sisällään yhden musiikkijulkaisun listaustiedot, ja joiden järjestys on identtinen lähdeaineistona toimineen Internet-sivun kanssa. Koska Beautiful Soup mahdollistaa määreiden käsittelyn Python Dictionaryn tyyliin, aiemmin mainittujen määreiden ´data-band-id´ ja ´data-item-id´ arvot voidaan nyt hakea yksinkertaisesti komennolla *´lista[0]["data-band-id"]´* ja *´lista[0]["data-item-id"]´*. Albumeiden kokonaismäärän voi tässä vaiheessa todeta komennolla *´len(lista)´*, ja ensimmäisenä sivulla listatun albumin tiedot sijaitsevat siis listaindeksin nolla (*´lista[0]´*) Tag-objektissa.

Albumisivun internet-osoite voidaan myös poimia samasta lähteestä komennolla *´lista[0].a['href']´*, jolla valitaan CSS-valitsimella *a*-tagin määreen *href*-arvo. Kuvassa 6. olevassa esimerkissä linkin *href*-arvo on pelkästään halutun internet-osoitteen loppuosa, joten projektiohjelmassa joutuu vielä lisäämään osoitteen alkuosan (joka on sama kaikille kyseisen julkaisijan alla) saadakseen toimivan internet-osoitteen myöhemmässä vaiheessa tapahtuvaa AIOHTTP:n suorittamaa verkkosivunoutoa varten. Toki myös *´data-item-id´*:n arvosta on poistettava alussa oleva osamerkkijono ”album-” (joka voi olla myös ”track-”, koska julkaisijasivun listaus voi sisältää myös yksittäisiä kappaleita) käyttökelpoisen id-numerosarjan saamiseksi.

```
In [8]: from bs4 import BeautifulSoup

In [9]: publisherSiteSoup = BeautifulSoup(publisherSite.content, 'lxml')

In [10]: albumsList = publisherSiteSoup.select("li.music-grid-item")

In [11]: len(albumsList)
Out[11]: 171

In [12]: albumsList[0]["data-band-id"]
Out[12]: '1613231443'

In [13]: albumsList[0]["data-item-id"]
Out[13]: 'album-382979573'

In [14]: albumsList[0].a['href']
Out[14]: '/album/ad-personam'

In [15]: albumsList[0]
Out[15]:
<li class="music-grid-item square first-four" data-band-id="1613231443" data-bind="css: {'featured': featured()}" data-item-id="album-382979573">
<a href="/album/ad-personam">
<div class="art">
<img alt="" src="https://f4.bcbits.com/img/a2673191186_2.jpg"/>
</div>
<p class="title">
            Ad Personam

                <br/><span class="artist-override">
                Exitium Sui
                </span>
</p>
</a>
</li>

In [16]: albumsList[170].a['href']
Out[16]: '/track/archonsbane-in-bloom-2'
```

Kuvio 7. Esimerkkikomentoja Beautiful Soupin käyttämisestä Pythonin interaktiivisessa komento-tulkissa.

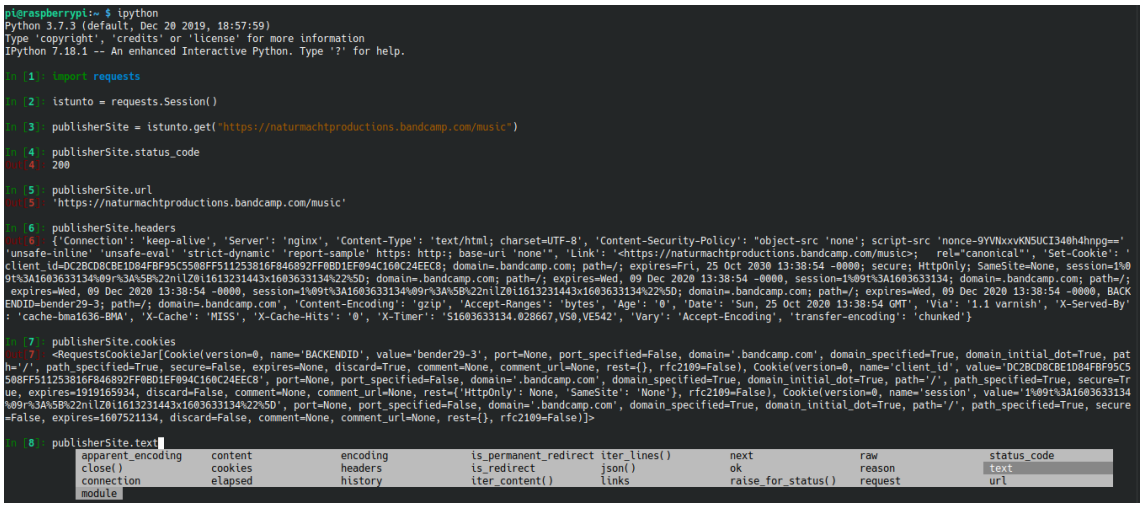
6.3 Requests

Johtuen Requests-kirjaston synkronisesta, tehtävä kerrallaan etenevästä toiminnasta, Internet-sisällön noutaminen on sillä huomattavasti yksinkertaisempaa verrattuna asynkronisen AIOHTTP-kirjaston käyttämiseen. Tämän vuoksi projektissa käytettiin Requests-kirjastoa niissä tapauksissa, joissa ohjelman on tarpeen ladata vain yksittäisiä tiedostoja, kuten julkaisijan Music-sivun hakemiseen, sekä albumi- ja kappalesivuilta löytyvien 128 kB/s mp3-tiedostojen lataamiseen.

Requests-kirjaston käyttöesimerkki, julkaisijan Music-sivun noutaminen.

Requests-kirjaston käyttäminen kannattaa aloittaa luomalla tehtävien pyyntöjen välillä tiettyjä parametreja (esimerkiksi cookiet) säilyttävä Session-objekti komennolla *´istunto = requests.Session()´*. Istunnon luominen ei ole välttämätöntä, mutta sen pitäisi myös hieman nopeuttaa samalle palvelimelle tehtäviä peräkkäisiä pyyntöjä. Istunnon luomisen jälkeen pyyntöjen tekeminen onnistuu yksinkertaisesti komennolla *´nimiSisällölle = istunto.get("internetosoite")´*.

Tämän jälkeen ´nimisisällölle´ on Requests-kirjaston Response-objekti, joka sisältää suoritetun pyynnön tietoja, kuten vaikkapa HTTP-otsikkotiedot (engl. HTTP headers), HTTP-tilakoodin (engl. HTTP status code) ja tärkeimpänä tietysti varsinaisen pyynnössä vastaanotetun sisällön, esimerkiksi HTML-sivun lähdetekstin. Haetun musiikkilistaus-sivun lähdekoodi voidaan nyt luovuttaa Beautiful Soupin käsiteltäväksi. Requests-kirjasto on todellakin hyvin helppokäyttöinen, joten en käsittele sitä tämän enempää.



Kuvio 8. Pythonin IPython-komentotulkissa suoritettuja esimerkkikomentoja Requests-kirjaston käyttämisestä.

6.4 AIOHTTP

AIOHTTP:n kohdalta on ensiksi mainittava, että kyseisen kirjaston pääominaisuus, eli rinnakkaisuutta (engl. concurrent) tukevat toiminnot käyttäen hyväksi Asyncio-kirjastoa, olivat (ja ovat vieläkin) itselleni erityisen tuntematon osa-alue Pythonin parissa. Tämän vuoksi muodostin AIOHTTP-koodini tekemällä vain pieniä muokkauksia internetistä löytyvään koodiesimerkkiin. Vaikka haluankin tutustua Asyncio-kirjaston käyttämiseen syvällisemmin, se vaatii enemmän aikaa mitä olin tämän projektin tarpeisiin valmis käyttämään.

AIOHTTP otettiin projektiin mukaan halusta nopeuttaa ohjelman käyttämistä. Alun pitäen ohjelma haki myös julkaisijoiden Music-sivun musiikkijulkaisulistauksessa olevat verkkosivut Requests-kirjastoa käyttäen. Yhtenä projektiohjelman toimintatavoitteena oli kuitenkin viiveetön siirtyminen julkaisijan diskografiaan kuuluvien musiikkijulkaisujen välillä. Jotta tämä pystyttäisiin toteuttamaan, jokainen musiikkijulkaisulistauksesta löytyvä, yksittäisen musiikkijulkaisun sisältävä verkkosivu olisi noudettava, ja kyseessä olevasta julkaisijasta riippuen näitä sivuja on kymmeniä, tai jopa satoja kappaleita. Esimerkiksi alaluvussa 6.2 sijaitsevassa 5. kuvassa esitetyssä musiikkijulkaisulistauksessa oli kirjoitushetkellä 171 kappaletta erillisiä musiikkijulkaisuja (albumeja tai yksittäisiä kappaleita).

Haettaessa tällaista määrää sisältöä yhtäjaksoisena vaiheena, AIOHTTP-kirjaston rinnakkaisuus tuo huomattavaa ajansäästöä verrattuna Requests-kirjaston yksittäinen pyyntö kerrallaan valmistuvaan toimintaan. Riippuu kuitenkin sivustosta, kuinka monta yhtäaikaista pyyntöä palvelimelle kannattaa lähettää, eikä liiallisuuteen meneminen ole suositeltavaa tiedonharavoinnin eettisestäkään näkökulmasta tarkasteltuna. AIOHTTP:n oletusasetus on 100 yhtäaikaista yhteyttä, joka omassa käyttötapauksessani oli aivan liian monta. Tällä oletusasetuksella toimiminen kuormittaa palvelua suhteettomasti, eivätkä pyynnöt tule valmistumaan onnistuneesti. Useimmat näin tehdyistä pyynnöistä tuottivat tulokseksi pelkän 503 HTTP-tilakoodin virhesivun (palvelu ei saatavissa, engl. service unavailable).

Satunnaisia 503-virheitä tuli useampia sivuja yhtäaikaisesti ladattaessa, vaikka laskin yhtäaikaisuuksien määrää AIOHTTP:n oletusarvosta 100 lopulta ainoastaan viiteen yhteyteen. Viisi yhtäaikaista yhteyttä vaikutti kuitenkin toimivalta kompromissilta latausvirheiden minimoimisen ja sivujen lataamisen nopeuttamisessa, verrattuna sivujen hakemiseen rinnakkaisuutta tukematonta Requests-kirjastoa käyttämällä.

AIOHTTP-kirjaston käyttöesimerkki, musiikkijulkaisusivujen noutaminen.

AIOHTTP-koodini pohjautuu blogikirjoitusten sarjaan, jossa alkuperäisen ”Making 1 million requests with python-aiohttp” -nimisen kirjoituksen innoittamana on tehty jatkokirjoitukset ”Making 100 million requests with Python aiohttp”, ja ”Making an Unlimited Number of Requests with Python aiohttp + pypeln” (Garcia 2018, viitattu 25.11.2020). Koska esimerkeissä haettiin usean websivun sisältö, ne soveltuivat ongelmani ratkaisemiseen pienellä muokkauksella, ja niiden perusteella tein AIOHTTP:n käyttämiseksi oman alkeellisen, mahdollisimman vähän toimintoja sisältävän asyncio-koodin. En voi millään tavalla kutsua tätä koodia optimaaliseksi tai esimerkilliseksi, mutta tärkeintä on että se kuitenkin suoritti haluamani toimenpiteen. Toki projektiohjelman ei tarvitsekaan tehdä kuin korkeintaan muutama sata pyyntöä yhdeltä istumalta.

```
2  throttle = aiohttp.TCPConnector(limit=5)
3  async def download_site(session, url):
4      global ownList
5      async with session.get(url) as response:
6          await ownList.append(str(await response.text()))
7
8  async def download_all_sites(sites):
9      global throttle
10     async with aiohttp.ClientSession(connector=throttle) as session:
11         tasks = []
12         for url in sites:
13             task = asyncio.ensure_future(download_site(session, url))
14             tasks.append(task)
15         await asyncio.gather(*tasks, return_exceptions=True)
16
17 def startPubAProcessing(siteList):
18     asyncio.get_event_loop().run_until_complete(download_all_sites(siteList))
```

Kuvio 9. AIOHTTP pyynnöt tekevä koodi projektiohjelmasta.

Puutteellisen asyncio ja AIOHTTP tietämykseni vuoksi en käy yllä olevaa koodia läpi yksityiskohtaisesti, vaan tyydyn pelkästään toteamaan muutaman seikan sen toiminnasta. *´startPubAProcessing´* funktiolle annetaan parametrina *´siteList´*, joka on kaikkien haettavien Internet-sivujen osoitteet sisältävä lista. Kohta *´throttle = aiohttp.TCPConnector(limit=5)´* rajoittaa yhtäaikaisten pyyntöjen määrän 5 kappaleeseen. Rivi *´await ownList.append(str(await response.text()))´* lisää jokaisen pyyntövastauksen sisältämän varsinaisen sisällön, eli kyseisen Internet-sivun lähdetekstin, listaan nimeltä *´ownList´*. Lopputuloksena syntyvä Python-lista *´ownList´* sisältää jokaisen sivulistassa annetun Internet-sivun lähdekoodin, ennalta arvaamattomassa, pyyntöjen valmistumisajankohdan muodostamassa järjestyksessä. Tämän järjestyksen selvittäminen, ja sivujen onnistuneen vastaanottamisen varmistaminen tehtiin säännöllisten lausekkeiden avulla, ja siitä puhutaan säännöllisten lausekkeiden osiossa.

6.5 JS2XML ja XPath 1.0

JS2XML osoittautui erittäin hyödylliseksi työkaluksi poimittaessa tietoja Javascript-upotuksista; Beautiful Soup ei siihen oman käsitykseni mukaan pystynyt, ja säännöllisten lausekkeiden käyttäminen olisi ollut huomattavasti työläämpää ja hitaampaa. JS2XML:n tarjoamat XPath 1.0 polkulausekkeet tarjosivat toimintaperiaatteen sisäistämisen jälkeen hyvin nopean ja käytännöllisen tavan haluttujen tietojen poimimiseen XML-dokumentiksi muutetun Javascriptin sisältä. Siirryn seuraavaksi XPath 1.0 käyttöesimerkkiin, jonka yhteydessä avaan myös XPathin käyttämisen perusteita.

JS2XML ja XPath 1.0 käyttöesimerkki, tietojen poimiminen musiikkijulkaisusivulta

Ensimmäinen askel JS2XML:n käyttämisessä on Javascript-koodin parsinta XML-dokumentiksi; halutun Javascript-upotuksen toimittaminen JS2XML:lle on käsitelty säännöllisten lausekkeiden esimerkissä. Seuraava koodi tuo JS2XML lisäosan Pythonin käytettäväksi, ja suorittaa parsinnan.

Import JS2XML

XML_koodi = js2xml.parse(javascript_koodi)

```
var SiteData = {
    supportEmail: "support@lumnos.bandcamp.com",
    is_custom_domain: null,
    env: "prod"
};

var BandData = {
    id: 1292771405,
    name: "Lumnos",
    fan_email: "mocopfvr@gmail.com",
    account_id: 3210813892,
    has_discounts: null,
    image_id: 6544210
};

var EmbedData = {
    tralbum_param: { name: "album", value: 264424610 },
    album_title: "Shining Mirros",
    linkback: "http://lumnos.bandcamp.com" + "/album/shining-mirros",
    art_id: 56878656,
    artist: "Lumnos",
    swf_base_url: "https://bandcamp.com",
    show_campaign: null,
    embed_info:
{"no_track_preorder":false,"exclusive_embeddable":null,"item_public":true,"public_embeddable":"22 Jun 2015
18:49:51 GMT"}
};

var FanData = {
    logged_in: false,
    name: null,
    image_id: null
};

var TralbumData = {
    // For the curious:
    // http://bandcamp.com/help/audio_basics#steal
    // http://bandcamp.com/terms_of_use
```

```
<program>
  <var name="SiteData">
    <object>
      <property name="supportEmail">
        <string>support@lumnos.bandcamp.com</string>
      </property>
      <property name="is_custom_domain">
        <null/>
      </property>
      <property name="env">
        <string>prod</string>
      </property>
    </object>
  </var>
  <var name="BandData">
    <object>
      <property name="id">
        <number value="1292771405"/>
      </property>
      <property name="name">
        <string>Lumnos</string>
      </property>
      <property name="fan_email">
        <string>mocopfvr@gmail.com</string>
      </property>
      <property name="account_id">
        <number value="3210813892"/>
      </property>
      <property name="has_discounts">
        <null/>
      </property>
      <property name="image_id">
        <number value="6544210"/>
      </property>
    </object>
  </var>
  <var name="EmbedData">
    <object>
```

Kuvio 10. Musiikkijulkaisusivun tietojen poimimiseen käytetyn Javascript-upotuksen alku, vasemmalla puolella Javascript-koodi valmiina JS2XML:n parsittavaksi, ja oikealla puolella koodi operaation jälkeen, XML-dokumentiksi muutettuna.

Näiden toimenpiteiden jälkeen voidaan siirtyä hakemaan haluamiamme tietoja XPath lausekkeita käyttäen. Seuraava XPath lauseke on sijaintipolku musiikkijulkaisusivun albumi-nimi arvoon, ja käymällä sen läpi tutustumme XPathin toimintaan tarkemmin.

//property[@name='current']//property[@name='title']/string/text()

XPathin sijaintipolku (engl. location path) koostuu useista sijaintiaskeleista (engl. location steps), ja jokainen sijaintiaskel taas muodostuu akselimäärittelystä (engl. axis specifier), solmutestistä (engl. node test), ja mahdollisesta vapaaehtoisesta, yhdestä tai useammasta lisäehtoja asettavasta predikaatista (engl. predicate). Sijaintipolun voi kirjoittaa joko lyhennetyllä syntaksilla, tai lyhentämättömällä syntaksilla. Lyhennetty syntaksi on helpompaa lukea ja kirjoittaa, kun taas lyhentämätön syntaksi tarjoaa, pitempien lausekkeiden hinnalla, enemmän määrittelyjä, ja avaa toimintaansa sitä ajatuksella lukevalle.

XPathin yhteydessä käytettävä ”solmu” termi kuulostaa aluksi hieman oudolta, alkuperäisen englanninkielisen ”node” sanan voisi ilmeisesti kääntää myös soluksi. Kyseessä on jälleen kerran tietotekniikan termi, jolle ei löydy kovin helpolla virallista suomennosta. Näyttää kuitenkin siltä että ”solmu” käännöstä käytetään enemmän kuin solua, vaikka jälkimmäinen kuulostaisi mielestäni järkevämmältä.

On tärkeää ymmärtää miten XPathin sijaintipolut hakevat osumia XML-dokumentista. Tämän määrittelyssä keskeinen käsite on kontekstisolmu (engl. context node). Kontekstisolmulla tarkoitetaan paikkaa, jonka sijaintiin perustuen sijaintiaskel määrittelee muiden ehtojensa, eli solmutestin ja mahdollisten predikaattien vaatimukset täyttävät tulokset. Jokaisen sijaintiaskeleen jälkeen, sitä seuraavan sijaintiaskeleen käyttämä kontekstisolmu määrittyy uudelleen, edeltävän sijaintipolkukokonaisuuden osoittamaan paikkaan. Tällä tavoin sijaintipolun osoittama kohde tarkentuu askel askeleelta.

Akselimäärittely määrittelee sijaintiaskeleen valitsemien solmujen ”sukulaissuhteen” XML-dokumentin puurakenteessa, kontekstisolmun näkökulmasta tarkasteltuna. Akselimäärittely voi olla joko etenevä (esim. lapsi, jälkeläinen) tai käänteinen (vanhempi, esi-isä). Esimerkkini sijaintipolku alkaa lyhennetyn syntaksin etenevällä akselimäärittelyllä *´//´*. Kyseinen akselimäärittely *´//´* tarkoittaa nykyisen puun juurta ja kaikkia sen jälkeläisiä, lyhentämättömällä syntaksilla kirjoitettuna *´/descendant-or-self::node()/´*. Tämä siis tarkoittaa, että esimerkin ensimmäinen sijaintiaskel voi sijaita missä kohtaa dokumenttia tahansa, kunhan se vain täyttää solmutestinsä (*´property´*) ja predikaattinsa (*´[@name='current']´*) vaatimukset.

Kuvio 11. XML-koodia XPath lausekkeiden testaajassa. Syötettynä esimerkin XPath sijaintipolku, joka valitsee albumin nimen

Ensimmäinen sijaintiaskel kokonaisuutena valitsee siis mistä kohtaa dokumenttia tahansa ´//´ (akselimäärittely), ´*property*´ elementin (solmutesti), joka predikaatin antaman lisämäärittelyn mukaisesti sisältää ´*name*´ attribuutin (lyhennetyssä syntaksissa attribuutti on ´@´) arvolla '*current*'.

Seuraavassa sijaintiaskeleessa ´*//property[@name='title']*´ määritellään sijaintipolkua lisää edellä selitettyyn tyyliin, eli tässä askeleessa haluttu ´*property*´ elementti voi olla saavutetun kontekstisolmun juuressa tai sen jälkeläisissä, ja tällä kertaa predikaatti määrittelee että sen ´*name*´ attribuutin arvon pitää olla '*title*'. Kolmas sijaintiaskel ´*/string*´ jatkaa sijaintipolkua nimen tarkistavalla solmutestillä, joka on tosi kontekstisolmuun lapsisuhteisen string-elementin kohdalla; lyhennetyssä syntaksissa "child" määrittely voidaan näet jättää pois, koska child-suhde on akselimäärittelyn oletus. Yksittäinen ´/´ on pelkkä erotin sijaintiaskelten välillä, sijaintipolut kun muistuttavat koostumukseltaan hakemistopolkuja. Viimeinen sijaintiaskel ´/text()´ on solmutesti, joka on tosi jokaisen kontekstisolmun kanssa lapsisuhteisen tekstisolmun kohdalla.

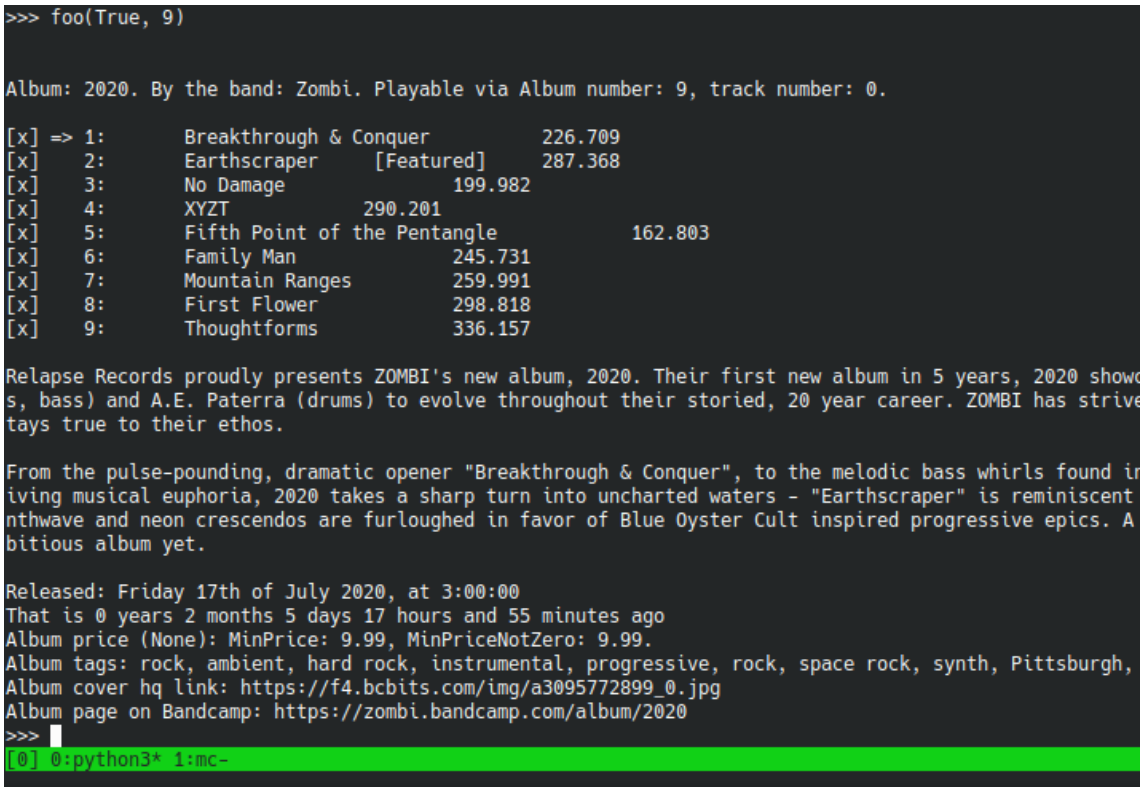
Lopuksi, nyt kun sijaintipolku on läpikäyty, JS2XML:ssä sijaintipolulla hakeminen tapahtuu seuraavalla komennolla:

> *tulokset_Lista=XML_koodi.xpath("//property[@name='current']//property[@name='title']/string/text()")*

Eli sijaintipolku tulee lainausmerkkien sisään, ja tulokseksi saadaan lista (vaikka osumia olisikin vain yksi) sijaintipolun tuottamista tuloksista.

7 VALMIIN PROJEKTIOHJELMAN ESITTELY

Koska tein projektia Pythonin interaktiivisessa komentotulkissa, ja tärkein asia oli tiedonharavointiin käytettävien ohjelmakomponenttien käytön opetteleminen, jäi tämän lopputyön piirissä ohjelmoitu ohjelmakokonaisuus käytettävyydeltään ja ulkonäöltään prototyyppi-asteelle. Sen erilaisia toimintoja käytetään kutsumalla Python-funktioita suoraan komentotulkista, josta ohjelmakokonaisuus ei poistunut missään vaiheessa kehitystä. Tästä huolimatta se kuitenkin tarjoaa osiossa 5.1 mainitut parannukset Bandcampin käyttämiseen, sitä ei vain ole viimeistelty käyttökokemuksen ja visuaalisen ilmeen puolesta ”valmiin” ohjelman vaatimalle tasolle. Loppuun asti viimeistelty ohjelma ei tietenkään missään vaiheessa ollutkaan tämän oppimisprojektin tavoitteena.



Kuvio 12. Albuminäkymä projektiohjelmassa, osa About-tekstin pitkistä riveistä rajattu ulkopuolelle jotta kuva pysyisi selkeämpänä.

Esittelen lyhyesti projektiohjelman yllä olevassa albuminäkymässä tarjoamat tiedot. Ensimmäiseksi löytyy musiikkijulkaisun nimi (2020), artisti (Zombi), musiikkijulkaisun järjestysnumero ohjelmassa (9), ja tämänhetkisen toistettavan kappaleen numero (0, koska listojen indeksinumerot alkavat nollasta). Tämän jälkeen listattuna ovat musiikkijulkaisun kappaleet ja niiden kesto sekunneissa.

Ilmaiseksi kuunneltavissa olevat kappaleet on ilmaistu merkinnällä ´[x]´, kun taas ilmaisen suoratoiston ulkopuolelle jätettyjen kohdalla olisi merkintä ´[ ]´. Parhaillaan toistettava kappale osoitetaan merkinnällä ´=>´. Kyseisessä kuvankaappauksessa ei tosin toisteta ensimmäistä kappaletta, vaan merkintä on sen kohdalla koska albumi on tuotu näkyviin pelkästään tiedot näyttävällä ohjelmakomennolla, jonka käyttämä koodi on miltei täysin yhteinen toistokomennon kanssa. Bandcampin sivuilla oletuksena ensimmäisenä soitettava kappale ilmenee ´[Featured]´ merkinnästä. Kappaleiden jälkeen seuraa About-teksti, jossa kerrotaan kyseisestä musiikkijulkaisusta. Sitten on julkaisupäivä (Friday 17th of July 2020, at 3:00:00), ja julkaisupäivän ”inhimillistetty” ajallinen etäisyys nykyhetkestä, joka voi myös olla tulevaisuudessa (aikatoiminnot Arrow-kirjaston avulla). Sen jälkeen löytyy musiikkijulkaisun hintatietoja, sille asetetut genre-avainsanat, linkki suurikokoiseen ”albumikansikuvaan” ja lopuksi linkki kyseisen musiikkijulkaisun internet-osoitteeseen Bandcamp-palvelussa.

Vaikka näkymä onkin askeettinen, siitä kuitenkin löytyy tärkeimmät asiat pelkistetyssä muodossa, jotta voi oikeasti keskittyä musiikin kuuntelemiseen, ilman häiritseviä visuaalisia elementtejä. Koska ohjelmaa käytetään suoraan interaktiivisessa komentotulkissa, visuaalinen näyttävyys voisi parhaimmillaankin tarkoittaa vain erilaisia tekstivärejä. Tekstiväriä itse asiassa löytyy siinä että jo ladatut kappaleet on merkitty eri värillä, kyseisessä kuvankaappauksessa olevasta albumista ei vain ollut ladattuna ainoatakaan musiikkitiedostoa. Vertailun vuoksi alkuperäinen selain-näkymä kyseisen albumin Bandcamp-sivusta löytyy liitteenä (Liite 1); vaikka olisinkin halunnut kuvan siitä projektiohjelman vierelle, oli kyseessä aivan liian suuri kuva tekstin seassa esitettäväksi.

Projektiohjelma keskittyy julkaisijapohjaisen musiikinkuuntelun parantamiseen. Ensimmäiseksi sille syötetään halutun julkaisijan Music-sivun osoite, jonka ohjelma sitten hakee ja selvittää siitä jokaisen listatun musiikkijulkaisusivun osoitteen. Tämän jälkeen se noutaa jokaisen näistä sivuista, ja poimii niistä tarvitsemansa tiedot. Tämän jälkeen ohjelmalla on mahdollista siirtyä musiikkijulkaisujen välillä järjestysnumeron avulla; kyseinen järjestys on identtinen sen järjestyksen kanssa, missä musiikkijulkaisut on julkaisijasivulla listattu. Eri musiikkijulkaisujen välillä liikkuminen tapahtuu välittömästi, ja musiikkitiedosto noudetaan siinä vaiheessa kun tietyn kappaleen soittamisen vuoro koittaa. Jos kyseinen kappale on jo noudettu aiemmin, sitä ei ladata uudelleen, vaan aiemmin tallennetun musiikkitiedoston toistaminen alkaa välittömästi.

Projektiohjelmaa ohjataan komentotulkkiin kirjoitettavilla tekstikomennoilla. Kuvassa 11. näkyy ylälaidassa syötetty komento *´foo(true, 9)´*, jolla saa näkyviin kyseisessä järjestysnumerossa

olevan albumin tiedot. Kappaleen toistamisen voi aloittaa komennolla *´playTrack(9, 1)*', jossa annetaan ensiksi musiikkijulkaisun järjestysnumero, ja kappaleen numero. Jos musiikintoistoa ei keskeytetä, se etenee automaattisesti kappaleesta seuraavaan, ja myös järjestysnumerossa seuraavan musiikkijulkaisuun siinä vaiheessa kun kaikki tämän hetkisen musiikkijulkaisun kappaleet on soitettu. ´playTrack(”n”)´ komennolla taas voi vaihtaa suoraan sen hetkisestä kappaleesta seuraavan musiikkijulkaisun ensimmäisen ilmaiseksi suoratoistettavissa olevan kappaleen soittamiseen. Vaikka ohjelmaa käytetäänkin suoraan interaktiivisesta komentotulkista, python-mpv:llä ohjattu mpv-musiikintoisto ei estä samanaikaista komentotulkkiin kirjoittamista ja siinä työskentelemistä.

Näiden tärkeimpien komentofunktioiden lisäksi projektiohjelmassa on myös komennot esimerkiksi toiston tauottamiseen ja siihen palaamiseen, kappaleen jäljellä olevan ajan näyttämiseen, äänenvoimakkuuden säätämiseen, kappaletietojen puhesyntetisaattorilla kuuluttamisen päälle ja pois laittamiseen, ja kappaleen toistossa sekuntimääräiseen eteenpäin siirtymiseen. Suurin osa näistä komennoista ei tietenkään ole perimmiltään muuta kuin omien lyhyempien ja vähemmän kirjoitusta vaativien funktioiden alle koottuja mpv-python komentojen kokonaisuuksia.

8 TULOKSET

Koska projektiohjelma esiteltiin jo edellisessä osiossa, käsittelen täällä vain sen, miten alaluvussa 5.1 mainittuihin tavoitteisiin päästiin.

Parannus kohtaan 1: Kankea liikkuminen albumisivujen välillä

Projektiohjelmassa liikkuminen musiikkijulkaisujen välillä tapahtuu viiveettä, koska niiden tietoja ei tarpeettomasti ladata useita kertoja kuuntelemisen aikana. Musiikkitiedostot ladataan sitä mukaa, kun niiden soittaminen aloitetaan, eikä samaa äänitiedostoa noudeta kuin yhden kerran riippumatta toistokertojen määrästä. Liikkuminen kappaleesta ja musiikkijulkaisusta seuraavaan voidaan asettaa automaattiseksi.

Parannus kohtaan 2: Tarve tietokoneen päällä pitämiseen musiikkia kuunnellessa

Raspberry Pi ei pidä ääntä päällä ollessaan, ja se kuluttaa sähköä vähemmän kuin normaali tietokone. Se soveltuu jopa jatkuvasti päällä pidettäväksi ollen normaalia tietokonetta huomattavasti paremmin musiikintoiston tarpeisiin mitoitettu laite.

Parannus kohtaan 3: Musiikin hallinta on sidottu äänentoistosta vastaavaan laitteeseen.

Tmux-terminaalimultiplekserissä suoritettavan projektiohjelman hallinta onnistuu miltä tahansa samaan verkkoon liittyneeltä pääte-emulaattoriohjelmaa pyörittävältä laitteelta, ja projektiohjelmaan voi olla yhteydessä jopa useammalta laitteelta yhtäaikaisesti. Projektiohjelma ei myöskään sulkeudu, vaikkei mikään laite olisi siihen hallintayhteydessä tai vaikka yhteys siihen katkeaisi jostain suunnittelemattomasta syystä.

9 JOHTOPÄÄTÖKSET

Tämän opinnäytetyön tavoitteena oli rakentaa tiedonharavointia hyödyntäen vaihtoehtoinen, alkuperäiseen selainpohjaiseen käyttöliittymään verrattuna tiettyjä parannuksia tarjoava tapa käyttää Bandcamp-musiikkipalvelua. Kiteytettynä tärkeimmät halutut parannukset olivat musiikinkuuntelun irrottaminen rajoittavaksi ja kömpelöksi koetusta selainkäytöstä, ja julkaisijan diskografioihin tutustumisen helpottaminen. Nämä tavoitteet täyttävän projektiohjelman koodaaminen toimisi mielenkiintoisena oppimiskehyksenä tiedonharavoinnin eri osa-alueisiin ja työkaluihin.

Projekti toteutettiin Python-ohjelmointikielellä, ja tätä päätöstä voi tulosten valossa pitää oikeana. Pythonille saatavilla olevat kirjastot ja lisäosat olivat tarpeeksi helppokäyttöisiä ja kykeneviä, jotta tällainen projekti oli toteutettavissa huolimatta siitä, etten ollut missään vaiheessa perusteellisesti opiskellut Python-ohjelmointia. Ehdottoman tärkeää oli myös Pythonin interaktiivisen komentotulkin tarjoama mahdollisuus välittömään ohjelmakoodin suorittamiseen ja testaamiseen ilman tarvetta kirjoittaa koodia erilliseen tiedostoon tai erillisen ohjelmointikielen kääntäjän suorittamisesta koituvaa kehittämistä hidastavaa lisäaskelta.

Säännöllisten lausekkeiden ja XPathin opettelemisessa ja rakentamisessa kannattaa ehdottomasti hyödyntää erilaisia online-testaajia. Etenkin monimutkaisempien säännöllisten lausekkeiden laatimisessa jonkinlainen testaajaohjelma on arvokas, olipa se sitten erillinen ohjelmansa tai online-työkalu. Tutustuessani enemmälti online-testaajiin esimerkeistä kirjoittaessani huomasin, että niiden epäröimättömällä käyttämisellä olisin saanut tiivistettyä joitain säännöllisiä lausekkeita sekä Pythonin string-metodeja hyödyntäviä työvaiheitani pelkästään yhdellä säännöllisellä lausekkeella suoritettaviksi tehtäviksi.

Lopputuloksena syntynyt projektiohjelma toi halutut parannukset, mutta kokonaisuutena sen ominaisuudet jäivät tietysti paljon selainliittymää rajoittuneemmiksi. Se helpotti etusijalla ollutta julkaisijapohjaista musiikkijulkaisujen kuuntelua, mutta jotta se olisi tarjonnut muilta osin edes suurin piirtein vastaavan määrän ominaisuuksia kuin virallinen selainkäyttö, olisi sen ohjelmoimista pitänyt jatkaa huomattavasti pitempään.

10 POHDINTA

Tekemäni tiedonharavointiin perustuva ohjelmointiprojekti oli omasta mielestäni hyvin mielenkiintoinen, koska ohjelmaan haluamieni ominaisuuksien toteuttaminen tarjosi motivaation selvittää ja tutustua hyvin moneen uuteen Python-lisäosaan, erillisiin ohjelmiin ja internet-palveluiden toimintaan. Tulin tutustuneeksi myös moniin ohjelmistoihin, joita tässä lopputyössä ei tuotu esille lainkaan tai mainittiin vain ohimennen, koska pääpaino oli tiedonharavoinnissa, ja jonkinlainen rajaus oli suoritettava. Esimerkkinä voi mainita vaikkapa puhesyntetisaation. Koska toteutin projektiohjelmassa myös puhesyntetisaattoria käyttävän kappaleen tietojen kuuluttamisen ennen toiston aloittamista, tutustuin Pythonille saatavilla oleviin puhesyntetisaattoreihin. Tästä opin, että nykyään puhesyntetisaation tuotteet ja kehitys ovat lähes yksinomaan www-sovelluspalveluiden alaa, eikä ilman internet-yhteyttä toimivia nykyaikaisia ratkaisuja juurikaan löydy. Halusin juuri paikallisesti toimivan ratkaisun, mutta se tarkoitti suurta pudotusta syntetisoidun äänen laadussa verrattuna nykyaikaisiin isojen tietotekniikkayritysten tarjoamiin, internet-yhteyttä vaativiin, sekä neuroverkko-tekniikoita hyödyntäviin puhesynteeseihin.

Tämän projektin tiimoilta tutustumieni uusien asioiden määrästä johtuen en voi väittää, että olisin voinut perehtyä mihinkään osa-alueeseen kovinkaan perusteellisesti. Itse asiassa projektin ohjelmointivaiheessa selvitin kutakin osa-aluetta vain sen verran, että sain aikaiseksi haluamani lopputuloksen tuottavan ratkaisun. On siis todennäköistä, etteivät projektiohjelmassa käytetyt ratkaisut ole kaikkein optimaalisimpia.

Jouduin tutustumaan käyttämiini asioihin perusteellisemmin opinnäytetyön kirjoitusvaiheessa ja erityisesti kirjoittaessani käyttöesimerkkejä hyödyntämistäni ohjelmakomponenteista. Jälkeenpäin tarkasteltuna olisikin ehkä kannattanut kirjoittaa käyttöesimerkit ensimmäiseksi, koska niitä valmistellessa mieleen tulee myös tärkeitä teorian puolelle kuuluvia tietoja. Omassa tapauksessani järjestys, jossa tehdään ensiksi jotain ohjelmistolla ja vasta sen jälkeen tutustutaan dokumentaatioon tarkemmin, on parempi motivaation kannalta. Näin menetellessä dokumentaatioon syventymistä ei ole haittaamassa malttamaton odotus päästä toteuttamaan ohjelmalla jotain konkreettista, ja omat alustavat käyttökokemukset tuovat esille omakohtaisesti tarpeellisia, tulevaisuuden monipuolisempaa käyttöä varten tärkeitä opiskelemisen kohteita. Vielä tätäkin parempi keino asioiden perinpohjaiseen ymmärtämiseen on yrittää tehdä niistä paikkansa

pitävää ja ytimekästä dokumentaatiota. Mitä tiiviimpää tästä tekstistä haluaa, sitä perusteellisemmin käsiteltävään asiaan joutuu tutustumaan.

Entäpä onko mielestäni suositeltavaa toteuttaa tiedonharavointiin perustuvia sovelluksia tästä projektista saamieni kokemuksien perusteella? Kuten alussa todettiin, jos vain suinkin on mahdollista käyttää www-sovelluspalvelun APIa tarvittavien tietojen hankkimiseen, niin se on huomattavasti parempi ratkaisu, mikä johtuu sekä selkeämmästä lainsäädännöllisesti perustasta että API-kanavan pysyvämmästä rakenteesta. Tiedonharavointia käyttäessä havahtuu vääjäämättä jonain päivänä siihen, että sovellus on lopettanut toimintansa verkkopalvelun sivujen rakenteen muututtua jollain tapaa. Näin kävi myös omalle ohjelmalleni eräässä vaiheessa syksyä. Se kuinka työlästä ohjelman palauttaminen toimivaksi on, riippuu täysin siitä kuinka paljon sen käyttämä lähderakenne muuttuu. Toki myös tietojen hakeminen www-sovelluspalvelusta on huomattavasti helpompaa ja tehokkaampaa kuin tiedonharavoinnin käyttäminen.

Jos kuitenkin tiedonharavointi on ainoa vaihtoehto, niin Python-ohjelmointikieli tarjoaa siihen hyviä työkaluja, joiden käyttäminen ei ole ylivoimaisen vaikeaa. Työkalujen käytön opettelemista paljon suurempaa ajanhukkaa ja huolta tulevat todennäköisesti aiheuttamaan palvelujen muuttumisen myötä tulevat ongelmat ja epävarmuus.

LÄHTEET

Bandcamp 2020a. Bandcamp Fair Trade Music Policy. Hakupäivä 11.10.2020. https://bandcamp.com/fair_trade_music_policy.

Bandcamp 2020b. Does Bandcamp support 30-second snippets?. Hakupäivä 11.10.2020. https://get.bandcamp.help/hc/en-us/articles/360007802374-Does-Bandcamp-support-30-second-snippets-.

Bandcamp 2020c. Seven reasons to go Pro. Hakupäivä 11.10.2020. https://bandcamp.com/pro.

Bandcamp 2020d. Bandcamp for Labels. Hakupäivä 11.10.2020. https://bandcamp.com/labels.

Bandcamp 2020e. I heard you can steal music on Bandcamp. What are you doing about this?. Hakupäivä 14.10.2020. https://get.bandcamp.help/hc/en-us/articles/360007902173-I-heard-you-can-steal-music-on-Bandcamp-What-are-you-doing-about-this-.

Bandcamp 2020f. In which formats can I download my purchases?. Hakupäivä 14.10.2020. https://get.bandcamp.help/hc/en-us/articles/360009319033-In-which-formats-can-I-download-my-purchases-.

Garcia, Christian 2018. Making an Unlimited Number of Requests with Python aiohttp + pypeln. Hakupäivä 25.11.2020. https://medium.com/@cgarciae/making-an-infinite-number-of-requests-with-python-aiohttp-pypeln-3a552b97dc95

GitHub 2020. js2xml. Hakupäivä 13.10.2020. https://github.com/scrapinghub/js2xml.

Kuchling, A.M. 2020. Regular Expression HOWTO. Hakupäivä 14.10.2020. https://docs.python.org/3/howto/regex.html#regex-howto.

Marriott, Nicholas 2020. Welcome to tmux!. Hakupäivä 14.10.2020. https://github.com/tmux/tmux/wiki.

McKay, Dave 2020. How to Use tmux on Linux (and Why It's Better Than Screen). Hakupäivä 14.10.2020. https://www.howtogeek.com/671422/how-to-use-tmux-on-linux-and-why-its-better-than-screen/.

Reitz, Kenneth 2020. Requests: HTTP for Humans. Hakupäivä 14.10.2020. https://requests.readthedocs.io/en/master/.

restfulapi.net 2020. REST Architectural Constraints. Hakupäivä 14.10.2020. https://restfulapi.net/rest-architectural-constraints/.

Richardson, Leonard 2020. Beautiful Soup Documentation. Hakupäivä 13.10.2020. https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

Smith, Chris 2020. Arrow: Better dates & times for Python. Hakupäivä 14.10.2020. https://arrow.readthedocs.io/en/stable/.

Termipankki 2012. Www-sovelluspalvelu. Hakupäivä 6.10.2020. https://termipankki.fi/tepa/fi/haku/www-sovelluspalvelu.

Termipankki 2013. Tiedonharavointi. Hakupäivä 6.10.2020. https://termipankki.fi/tepa/fi/haku/tiedonharavointi.

Twitter Inc. 2011. One Million Registered Twitter Apps. Hakupäivä 6.10.2020. https://blog.twitter.com/official/en_us/a/2011/one-million-registered-twitter-apps.html.

Twitter Inc. 2020. Understand the public conversation. Hakupäivä 6.10.2020. https://developer.twitter.com/en/use-cases/listen-and-analyze.

Uriegas, Eli 2017. Asynchronous HTTP Requests in Python. Hakupäivä 14.10.2020. https://www.terriblecode.com/blog/asynchronous-http-requests-in-python/.

W3C 2020. Xpath cover page. Hakupäivä 25.11.2020. https://www.w3.org/TR/xpath/.

Wikipedia 2020a. Raspberry Pi. Hakupäivä 13.10.2020. https://en.wikipedia.org/wiki/Raspberry_Pi.

Wikipedia 2020b. Raspberry Pi OS. Hakupäivä 13.10.2020. https://en.wikipedia.org/wiki/Raspberry_Pi_OS.

Wikipedia 2020c. mpv (media player). Hakupäivä 14.10.2020. https://en.wikipedia.org/wiki/Mpv_(media_player).

Wikipedia 2020d. Unix time. Hakupäivä 25.11.2020. https://en.wikipedia.org/wiki/Unix_time.

ALKUPERÄINEN SELAINNÄKYMÄ KYSEISEN ALBUMIN BANDCAMP-SIVUSTA LIITE 1