WWW.JAMK.FI

Järviradion levystö- ja soittojenraportointiohjelmisto

Samuli Lager

Teemu Myller

Opinnäytetyö

Toukokuu 2013

Ohjelmistotekniikka

Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN KUVAILULEHTI

Tekijät LAGER, Samuli MYLLER, Teemu	Julkaisun laji Opinnäytetyö	Päivämäärä 13.5.2013
	Sivumäärä 56	Julkaisun kieli suomi
		Verkkojulkaisulupa myönnetty (X)
Työn nimi JÄRVIRADION LEVYSTÖ- JA SOITTOJENRAPORTOINTIOHJELMISTO		
Koulutusohjelma Ohjelmistotekniikan koulutusohjelma		
Työn ohjaajat PIETIKÄINEN, Kalevi SALMIKANGAS, Esa		
Toimeksiantaja Järviradio		

Tiivistelmä

Opinnäytetyönä tehtiin radioasema Järviradiolle ohjelmisto, jolla he voivat hallita ja käyttää levytietokantaansa sekä toimittaa sähköiset raportit soitettujen kappaleiden tiedoista tekijänoikeusjärjestöille. Raportti tuli toimittaa muodossa, jonka Gramex ja Teosto olivat tarkasti määritelleet.

Ohjelmisto toteutettiin WWW-sovelluksena mahdollistamaan laite- ja käyttöjärjestelmäriippumattoman etäkäytön. Palvelinsovellus kirjoitettiin Python-ohjelmointikielellä käyttäen Django-kehysympäristöä, joka tarjoaa mm. rajapinnan tietokannan käyttöön ja HTTP-pyyntöjen käsittelyyn. Internetselaimella käytettävä käyttöliittymä kirjoitettiin HTML-merkintäkielellä ohjelmoiden toiminnallisuus JavaScript-tekniikoita käyttäen.

Vanhan järjestelmän levytietokantaa analysoitiin ja sen tietojen siirtämiseksi ohjelmiston käyttöön toteutettiin erilaisia skriptejä Pythonilla ja Perlillä. Entisestä järjestelmästä tuotiin yli 31 000 albumin ja hieman alle 200 000 kappaleen tiedot uuden ohjelmiston tietokantaan.

Toteutettu ohjelmisto on jo asiakkaan käytössä. Mahdollisuuksia jatkokehitykseen jäi runsaasti, kuten työkaluja vanhan levystön tiedoissa olevien virheiden korjaamiseen.

Avainsanat (asiasanat)
ohjelmointi, Python, tietojärjestelmät, tietokannat, WWW-sivustot

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Authors	Type of publication	Date
LAGER, Samuli MYLLER, Teemu	Bachelor's thesis	13.5.2013
	Pages 56	Language finnish
		Permission for web publication (X)

Title
AUDIOLIBRARY SOFTWARE WITH ROYALTY REPORTING FOR JÄRVIRADIO

Degree programme
Software Engineering

Tutors
PIETIKÄINEN, Kalevi
SALMIKANGAS, Esa

Assigned by
Järviradio

Abstract
Radio station Järviradio requested software for managing and using their audio database and reporting played songs to copyright organisations for royalties. Such application was designed and implemented as a bachelor's thesis. The form of the report was defined by Teosto and Gramex.

The software was implemented as WWW application to allow device and operating system independent remote access. The server-side of the software was written in Python, using Django-framework, which provides i.a. interfaces for database access and handling of HTTP-requests. The user interface was written in HTML to be used in any web browser and its functionality was programmed using JavaScript technologies.

The database of Järviradio's old system was analyzed and various scripts were written with Python and Perl for parsing the data for the use of the developed software. The information of totally over 31 000 albums and slightly less than 200 000 tracks was successfully brought to the database.

At the time of returning the thesis, the implemented software was already in active customer use. There were plenty of possibilities left for further development in the future, such as tools for repairing errors in the data from the old database.

Keywords
databases, information systems, programming, Python, www-sites

Miscellaneous

SISÄLTÖ

KUVIOT

TERMIT JA LYHENTEET

Access

Microsoftin toimisto-ohjelmistopakettiin kuuluva tietokantaohjelmisto.

Ajax

Asynchronous Javascript And Xml on kokoelma tekniikoita, joiden avulla JavaScriptillä voidaan vaihtaa palvelimen kanssa tietoa ja päivittää osaa sivun HTML-sisällöstä lataamatta koko sivua uudelleen. Nimestään huolimatta tiedonvälitykseen käytetään nykyisin enemmän JSON-muotoa kuin XML:ää

Autentikointi

Käyttäjän henkilöllisyyden varmistus eli todennus

CoffeeScript

Syntaksiltaan selkeä ja yksinkertainen JavaScriptiksi käännettävä ohjelmointikieli.

CSS

Cascading Style Sheets on HTML-sivujen ulkonäköön ja tyyliin liittyvien määrityksien kieli, jonka tarkoitus on erottaa rakenne (HTML) ja ulkoasu (CSS) omiin tiedostoihinsa.

Dekoraattori

Pythonin dekoraattoreilla mahdollistetaan funktioiden ja metodien toiminnan kätevä muuttaminen. Pythonissa dekoraattorit ovat niin sanottua syntaktista sokeria, sillä ne tuovat uutta toiminnallisuutta kieleen, mutta tekevät kielen kirjoittamisen mukavammaksi. Dekoraattori vaikuttaa siihen funktioon, jonka eteen se on kirjoitettu.

Django

Pythonilla toteutettu MVC-arkkitehtuuria toteuttava Web-kehysympäristö.

FTP

File Transfer Protocol on yleinen yhteyskäytäntö tiedostojen siirtoon.

Git

Git on avoimen lähdekoodin hajautettu versiohallintajärjestelmä, jonka kehityksen aloitti Linus Torvalds Linuxin kehitystä varten.

Gramex

Gramex on rekisteröity yhdistys, joka valvoo äänitteillä esiintyvien taiteilijoiden ja äänitteiden tuottajien tekijänoikeuslaissa säädettyjä oikeuksia ja kerää heille korvauksia. Gramex edustaa yli 45 000 kotimaista ja lukemattomia ulkomaista musiikkiteollisuuden toimijaa. Se on perustettu vuonna 1967.

HTML

HyperText Markup Language on merkkauskieli, jolla määritellään dokumentteja. HTML-dokumentti sisältää yleensä tekstiä, rakenteen osoittavaa merkkausta sekä eri tyyppisiä viittauksia muihin dokumentteihin, kuten kuva- tai äänidataan.

HTTP

Hypertext Transfer Protocol on selainten ja WWW-palvelinten tiedonsiirtoon käyttämä yhteyskäytäntö.

JavaScript

JavaScript on enimmäkseen Web-ympäristössä käyttöliittymälogiikan luontiin käytetty ohjelmointikieli.

JSON

JavaScript Object Notation on tiedon vaihtoon tehty tietomuoto, jota on helppo kirjoittaa sekä lukea koneellisesti ympäristöstä ja ohjelmointikielestä riippumatta.

Kehysympäristö

Ohjelmistojen kehityksessä käytettävä ohjelmistoalusta, joka sisältää ohjelmakirjastoja ja geneeristä toiminnallisuutta.

Komento

Komentotulkissa suoritettava ohjelma.

Kontrolleri

Kontrollerilla tarkoitetaan MVC-arkkitehtuurin osaa, joka tarkkailee tapahtumia ja reagoi niihin.

Lokalisointi

Sisällön esittäminen käyttäjän omalla kielellä ja kulttuurisilla merkintätavoilla.

MySQL

MySQL on relaatiotietokantaohjelmisto, joka on saatavissa vapaalla GNU GPL -lisenssillä. Mikäli käyttäjä ei halua sitoutua GPL-lisenssiin, hän voi hankkia kaupallisen lisenssin.

ORM

Object-relational mapper on tekniikka, jolla tietomallit voidaan määritellä ohjelmointikielen luokkina ja näistä luokista tehtyjen olioiden kautta voidaan lisätä, muokata ja poistaa tietokannan sisältöä.

Perl

Practical Extraction and Report Language on skriptimäinen ohjelmointikieli, joka alunperin suunniteltiin erilaisten raporttien tuottamiseen.

PHP

PHP: Hypertext Preprocessor on erittäin suosittu Web-palveluiden toteuttamisessa käytetty Perlin kaltainen ohjelmointikieli.

PostgreSQL

PostgreSQL on Relaatiotietokantaohjelmisto, joka on saatavissa joustavalla BSD-tyyppisellä lisenssillä.

Python

Python on monipuolinen ja tulkattava ohjelmointikieli. Sen yksinkertainen syntaksi ja korkean tason tietorakenteet tekevät siitä hyvin ilmaisuvoimaisen ja helppokäyttöisen.

SQL

Structured Query Language on IBM:n kehittämä standardoitu kyselykieli, jolla voidaan tehdä hakuja, muutoksia ja lisäyksiä relaatiotietokantaan.

Teosto

Suomalaisten säveltäjien ja musiikintekijöiden edunvalvontajärjestö, joka jakaa lupia musiikin esittämiseen ja tallentamiseen sekä kerää ja tilittää niistä korvaukset musiikintekijöille ja kustantajille.

Skripti

Skripti on komentosarja, jolla automatisoidaan joitain vaihtoehtoisesti käsin tehtäviä toimenpiteitä.

Säännölliset lausekkeet

Säännölliset lausekkeet (engl. *regular expressions, regexp*) on kielioppi, jossa määritellään tiettyjen sääntöjen avulla merkkijonon rakenne.

URL

Uniform Resource Locator on merkkijono, jota käytetään internetsivun osoitteena.

Versiohallinta

Tiedostoihin tapahtuvat muutokset ja niiden tekijän tallentava järjestelmä.

WSGI

Web Server Gateway Interface on Web-palvelimen ja -sovelluksen välistä kommunikointia määrittelevä Python-standardi.

XML

Extensible Markup Language on merkintäkieli, joka sisältää tiedon määrityksen lisäksi myös itse tiedon samassa dokumentissa.

Äänite

Mitä tahansa ääntä sisältävä tallenne tai tiedosto kuten cd-levy, minidisc, c-kasetti, lp-levy, ääninauha tai esimerkiksi tietokoneen muistissa tai USB-tikulla oleva äänitiedosto.

1 OPINNÄYTETYÖN LÄHTÖKOHDAT

1.1 Asiakas

Järviradio on suomalainen paikallisradioasema, jolla on noin 95 000 viikoittaista kuuntelijaa (Radioiden mediakortit 2013). Radion päästudio sijaitsee Alajärvellä ja se työllistää noin 20 henkilöä.

Alun perin Etelä-Pohjanmaan Järviseudun paikallisradiona aloittaneen radioaseman lähetykset kuuluvat nykyään suuressa osassa Pohjanmaata ja Keski-Suomea. Järviradiota voi kuunnella suorana myös internetradion kautta ympäri maailman.

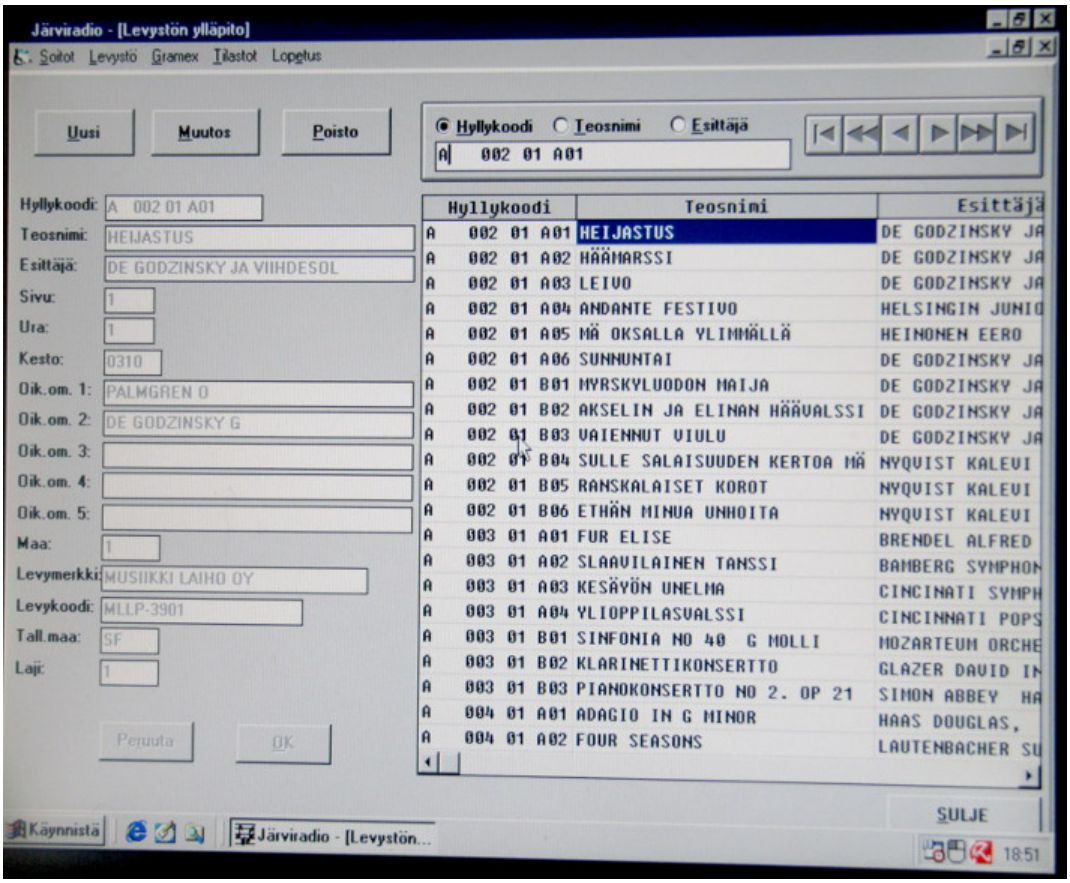
Järviradio on aktiivisesti mukana kuuluvuusalueensa tapahtumissa. Järviradion musiikkitarjonta keskittyy pääosin kotimaiseen iskelmämusiikkiin, mutta myös muille musiikkityyleille on omia erikoisohjelmia. Ohjelmistoon kuuluu myös esimerkiksi urheilua ja hengellisiä lähetyksiä.

1.2 Tehtävät ja tavoitteet

Järviradio tarvitsi levystönsä käyttöön ja hallintaan ohjelmiston, joka myös suorittaisi automaattiset soittojen raportoinnit Teostolle ja Gramexille. Tekijänoikeusjärjestöihin kuuluvien artistien musiikin julkisesta esittämisestä täytyy maksaa korvauksia ja kaupallisten radioiden tuleekin toimittaa kuukausittainen listaus soitetuista kappaleista sähköisesti erillisen ohjeen mukaisesti (Kaupallisten radioiden musiikin esitystietojen sähköisen raportoinnin ohje 2009). Studiolle lähetetään päivittäin uusia äänitteitä, joiden tiedot täytyy kirjata levytietokantaan. Vanhasta tietokannasta oli lisäksi tuotava kaikki levyihin ja kappaleisiin liittyvä tieto toteutettavan ohjelmiston käyttöön.

Vanha levytietokanta oli Microsoft Access -ratkaisu, jonka levystöhallinnasta ruutukaappaus kuviossa 1. Vanhaa järjestelmää ajettiin Microsoft Windows 98 -käyttöjärjestelmällä, joka oli toteutettu noin viisitoista vuotta sitten, eikä enää vastannut nykypäivän tarpeita tai tekijänoikeusjärjestöjen vaatimuksia. Levytietokannan päivitys Järviradion kotisivujen levystöhakuun vaati erillisen toimenpiteen, mikä on

nykypäivänä täysin automatisoitavissa. Ohjelma ei myöskään tarkistanut tai automatisoinut syötteitä, minkä vuoksi kaikki tiedot piti syöttää käsin. Tämä lisäsi sekä työntekijöiden työtä että tietokannassa olevan tiedon virheellisyyttä.



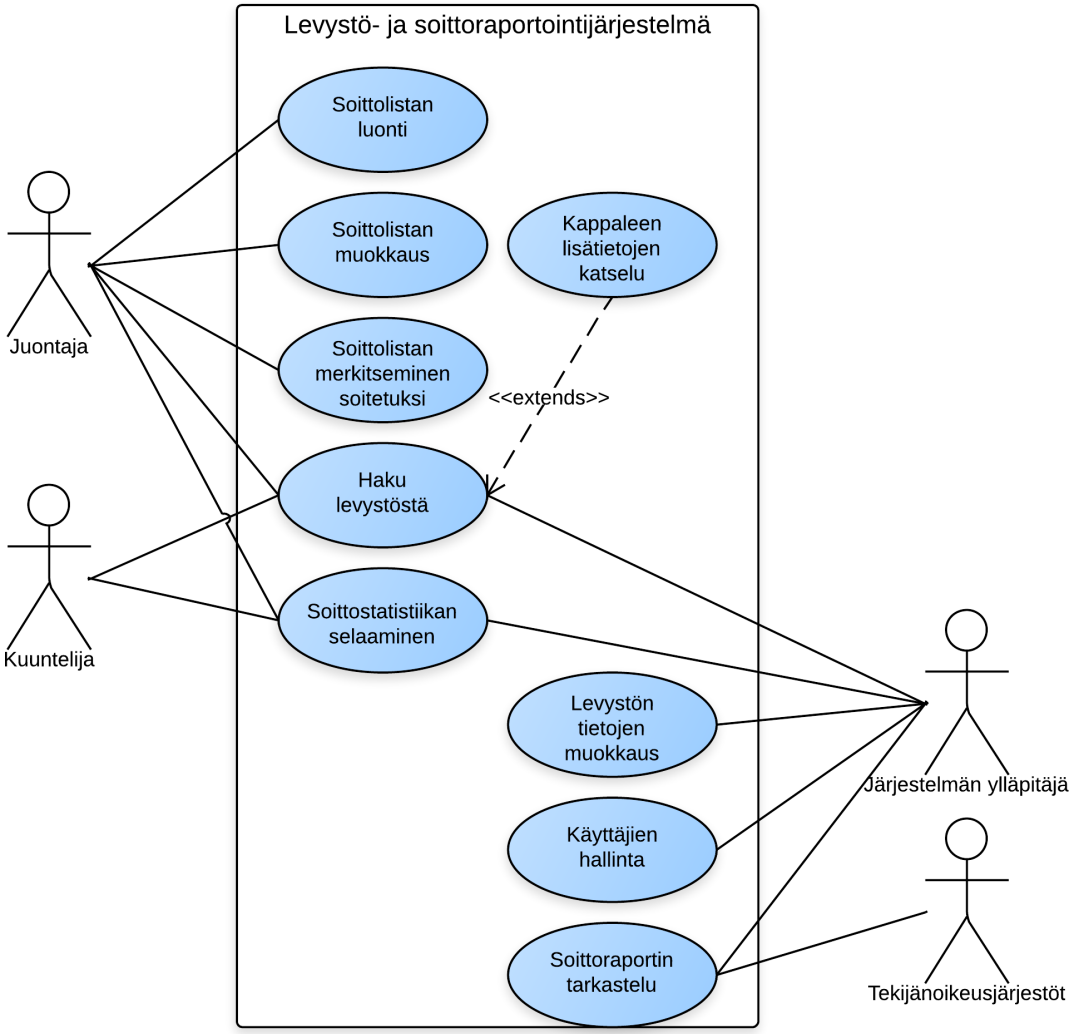
Kuvio 1. Ruutukaappaus Access-ratkaisun levystönhallinnasta

Järjestelmään haluttiin samalla kuuntelijoille käyttöliittymä, jolla he voisivat selata levytietokantaa ja hakea soitettavissa olevia kappaleita. Kappaleista voi tehdä soittopyyntöjä ilmoittamalla studioon kappaleen hyllykoodi joko puhelimitse tai tekstiviestillä.

Ohjelmistoon haluttiin myös näkyviin statistiikkaa järjestelmän tiedoista kuten soitetuimmista kappaleista ja artisteista. Statistiikan arveltiin kiinnostavan sekä Järviradion kuuntelijoita että henkilökuntaa.

Juontajia varten ohjelmistoon toteutettiin soittolistojen luontia ja muokkausta varten toiminnallisuus, jolla tulevien lähetysten ohjelman voi tehdä etukäteen tai vaikkapa lennossa lähetyksen aikana. Kappaleiden merkitseminen soitetuiksi tekijänoikeusjärjestöjen raporttia varten tuli myös olla mahdollisimman helppokäyttöistä. Ennen

soitot oli merkitty papereille, joista levystön ylläpitäjä kirjasi ne soitetuiksi Access-järjestelmään. Kuviossa 2 on kokonaisuuden hahmottamista varten esitetty käyttötapauskaavio, josta selviää järjestelmän toiminnot ja niiden käyttäjät.



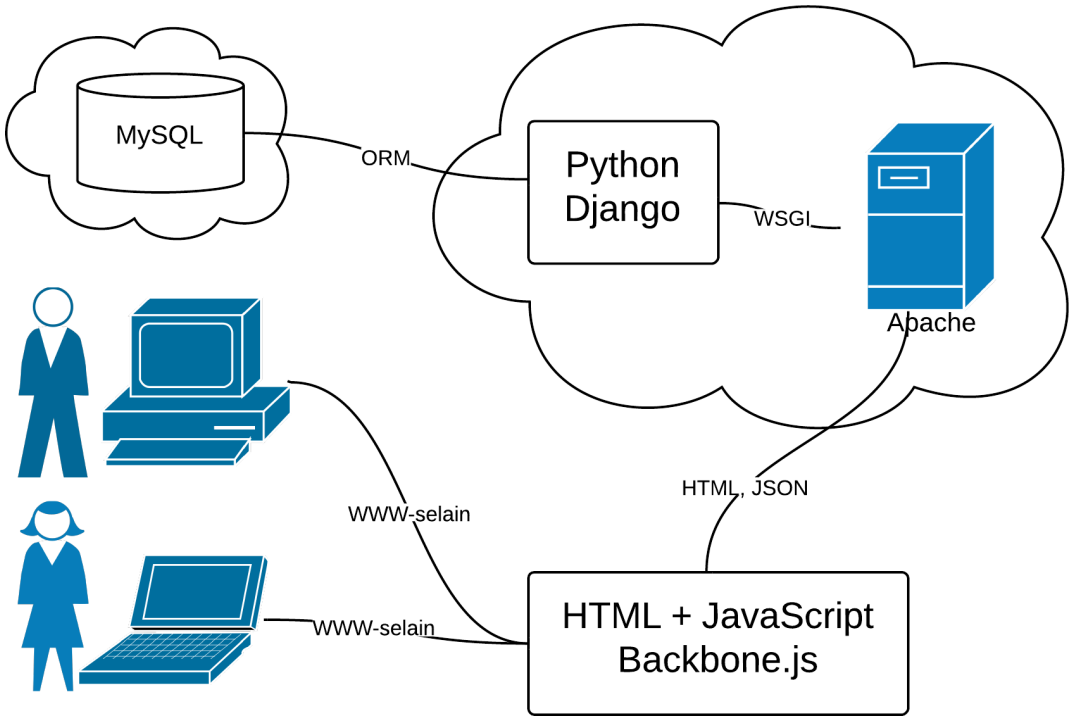
Kuvio 2. Ohjelmiston käyttötapauskaavio

2 MENETELMÄT JA TYÖKALUT

2.1 Web-ympäristö

Asiakkaan vaatimukset ja toiveet huomioon ottaen ohjelmisto päätettiin toteuttaa HTTP-palvelimella toimivana WWW-sovelluksena. Ratkaisussa on etuna se, että ohjelmistoa pääsee internetyhteydellä vaivattomasti käyttämään lähes mistä tahansa. Internetyhteyden päässä ohjelmisto tarjoaa samalla kuuntelijoille selaus- ja hakumahdollisuuden levystöön. Ratkaisun etuna on myös se, että järjestelmään lisätyt albumit ja kappaleet tulevat reaaliaikaisesti näkyviin.

Kuviossa 3 on kuvattu toteutettua järjestelmää suurpiirteisesti. Palvelinpuolen ohjelmisto toteutettiin Python-ohjelmointikielellä käyttäen Django-kehysympäristöä, asiakasohjelma taas HTML-sivustona, johon tehtiin varsinainen toiminnallisuus JavaScript-tekniikoita käyttäen.



Kuvio 3. Järjestelmäkaavio

Hankitulla vuokrapalvelimella pyörii Web-palvelinohjelmistona Apache ja tietokantapalvelimena MySQL. Toteutettu ohjelmisto toimii kuitenkin minkä tahansa WSGI:a

tukevan Web-palvelinohjelmiston kanssa ja valinnat ovat halutessa vaihdettavissa. Tämä tuskin kuitenkaan on tarpeen, sillä molemmat ovat laajalti käytössä olevia ja varmatoimisia valintoja. Ohjelmistoa ei myöskään tarvitse ajaa julkisella internetpalvelimella, vaan sitä voi yhtä hyvin ajaa myös intranetissä.

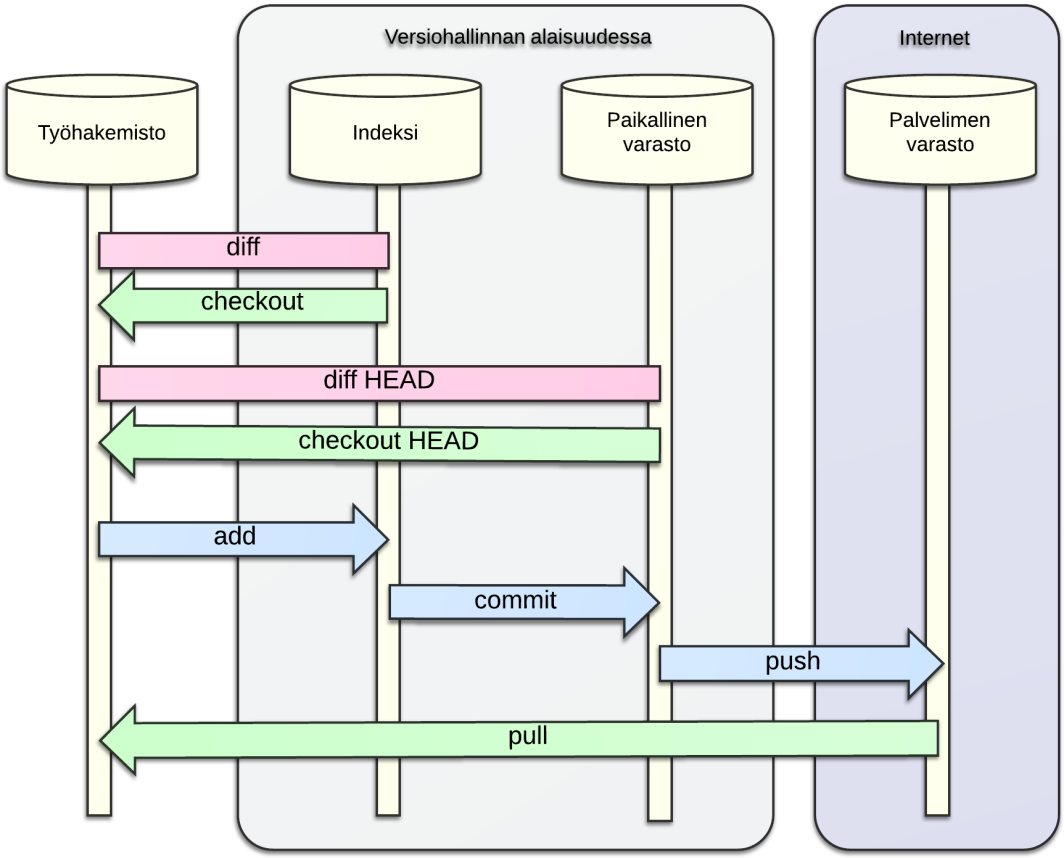
2.2 Versionhallinta ja Git

Versionhallinnan käyttö on käytännössä välttämättömyys ohjelmistoprojektissa. Lähdekoodiin tehtyjen muutosten tallentamisen lisäksi se vähentää päällekkäisten muutosten mahdollisuutta ja paikallisia versionhallintajärjestelmiä lukuun ottamatta, mahdollistaa usean kehittäjän käsiksipääsyn ajan tasalla olevaan lähdekoodistoon. Ohjelmiston versioiden tallentaminen on tärkeää, jotta mahdollisten ongelmien myötä voi palata aiempaan tilaan.

Ennen varsinaisia versionhallintajärjestelmiä yleinen tapa oli tehdä työhakemistosta kopio, mikä on erittäin virhealtis ja pidemmän päälle hyvin epäkäytännöllinen tapa. Ensimmäiset ratkaisut olivat *paikalliset versionhallintajärjestelmät*, jotka ainoastaan sisälsivät yksinkertaisen tietokannan muutoksille. Ohjelmistojen koon suurentuessa tuli tarve tehdä yhteistyötä muiden kehittäjien kanssa, minkä vuoksi luotiin *keskitetyt versionhallintajärjestelmät*. Ne toimivat ja säilyttävät versiohistorian palvelimella, josta asiakasohjelmat hakevat tiedostot. (Alkusanat - versionhallinnasta 2013) Haittana palvelimella toimivassa versionhallintajärjestelmässä on esimerkiksi, että yhteyden ollessa poikki tai palvelimen ollessa nurin ei pääsyä tiedostoihin eikä versiontallennusmahdollisuutta paikallisiin muutoksiin ole ollenkaan. Palvelimen kiintolevyrikon myötä menettää koko muutoshistorian ilman hajautettua varmuuskopiota. Tähän auttavat *hajautetut versionhallintajärjestelmät*, joissa asiakasohjelmat hakevat viimeisimmän tilannekuvan sijaan täyden versiohistorian, joka toimii myös varmuuskopioina. Tässä ohjelmistoprojektissa käytetty avoimen lähdekoodin Git on juuri tällainen hajautettu versionhallintajärjestelmä.

Kuviossa 4 on esitetty Git-versionhallinnan rakennetta. Työhakemisto on paikallisella koneella oleva hakemisto, jonka alla olevia tiedostoja halutaan tallentaa Gitin versionhallintaan. Komennolla *git add* lisätään tiedostoja indeksiin, jota voisi kuvailla puskurina tallentamattomien muutosten ja tallennetun versiohistorian välillä. Varsi-

nainen versiotallennus tehdään indeksissä olevista tiedostoista komennolla *git commit*. Tämä luo paikalliseen varastoon niin sanotun *commitin*. Sanalle ei ole vakiintunutta suomenkielistä vastinetta tai edes kandidaattia sellaiseksi. Commit sisältää yksilöllisen tunnisteen, muutokset tiedostoihin sekä käyttäjän syöttämän viestin, johon hyvän tavan mukaisesti selitetään tehdyt muutokset selkokielellä.



Kuvio 4. Git-versionhallinnan toiminta

Komennolla *git diff* saadaan nähtäväksi erot työhakemiston tiedoston ja indeksiin viedyn välillä. Mikäli halutaan vertailla vanhempiin versioihin voidaan joko antaa parametrina halutun commitin tunniste tai käyttää apuna viittausta HEAD, joka osoittaa viimeisimpään commitiin. Aiempaan historiaan viitatessa erotetaan HEAD:sta tildellä järjestysnumero, esimerkiksi HEAD~3, joka viittaa kolmanteen commitiin viimeisimmästä taaksepäin. Komennolla *git checkout* saadaan palautettua tiedosto sellaiseksi kuin se oli indeksiin lisättäessä. Versiohistoriasta saadaan haluttuun commitiin viittamalla vanha versio vastaavalla tavalla kuin *diff*:n kanssa. Komennoilla *git push* ja *git pull* siirretään versiotietoa paikallisen varaston ja etävaraston välillä.

Gitin ominaisuudet eivät rajoitu tiedostojen muutosten tallentamiseen. Hyödyllisimpiä komentoja ovat muun muassa *git status*, joka näyttää työhakemiston muutokset eriteltynä sen mukaan, onko tiedosto versionhallinnan ulkopuolella. Jos ei, niin komento näyttää onko tiedostoon tullut indeksiin viemättömiä muutoksia. Versionhallinnan historiaa pääsee tutkimaan käskyllä *git log*, ja minkä tahansa commitin voi nimetä komennolla *git tag*, jolloin kyseiseen commitiin voidaan viitata sille annetulla nimellä. Esimerkiksi ensimmäisen valmiin version valmistuessa voi ajaa käskyn *git tag v1.0*, jolloin jatkossa voidaan esimerkiksi tarkistaa, kuinka tiedosto on muuttunut ensimmäisestä versiosta komennolla *git diff v1.0*.

2.3 Python ja Django

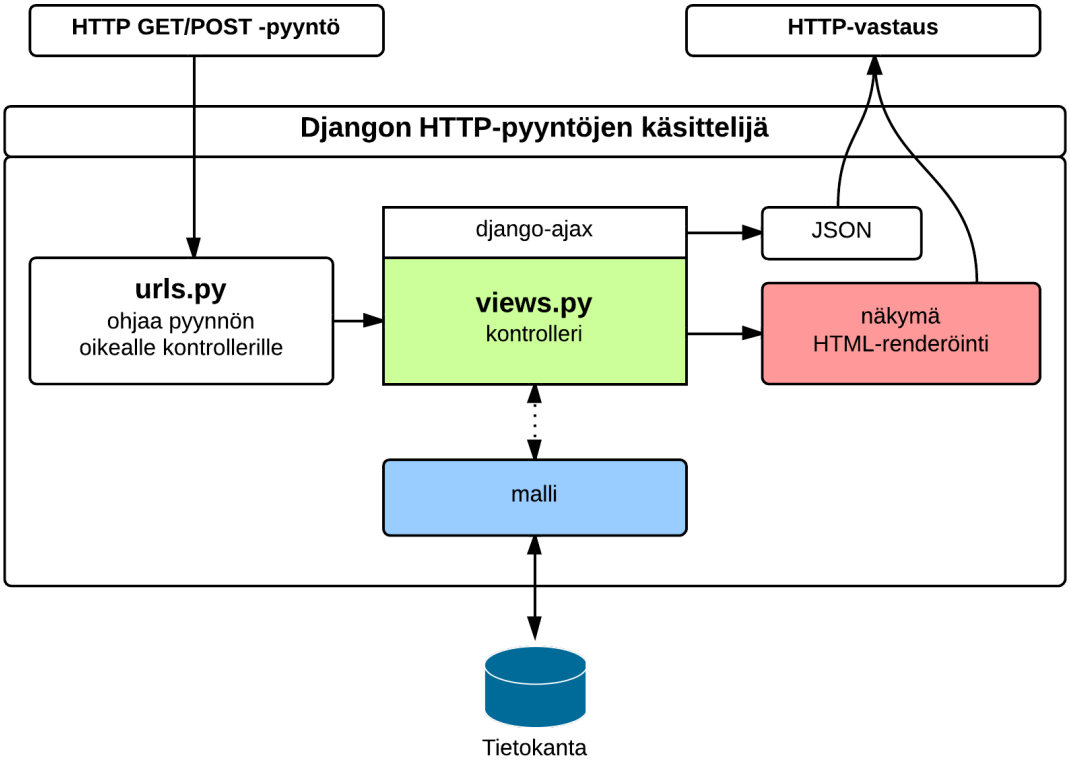
2.3.1 Yleistä

Python on alustariippumaton, erittäin ilmaisuvoimaisena pidetty ohjelmointikieli. Yleensä Pythonilla kirjoitettu ohjelma sisältää vähemmän koodirivejä kuin esimerkiksi C++:lla tai Javalla kirjoitettu vastaava ohjelma ilman, että kielen syntaksi olisi suppeaa tai epäselvää. Python on alustariippumatonta: yleensä samoja ohjelmia voidaan ajaa Microsoft Windowsissa ja Unixin kaltaisissa järjestelmissä kuten Linuxissa tai Mac OS X:ssä. Python on pohjimmiltaan oliopohjainen ohjelmointikieli, mutta sillä voi toteuttaa myös funktionaalista proseduuria. (Summerfield 2008, s. 1)

Django on avoimen lähdekoodin Python-ohjelmointikielellä toteutettu MVC-arkkitehtuuria toteuttava Web-kehysympäristö. MVC tulee sanoista model-view-controller eli malli-näkymä-kontrolleri ja suunnittelumallina sen ajatus on käyttöliittymän ja toimintalogiikan pitäminen erillään. MVC -mallissa arkkitehtuuri jaetaan kolmeen osaan. Malli sisältää toimintalogiikan ja on ainoana osana yhteydessä tietovarastoon, näkymä hoitaa tietojen esittämisen ja kontrolleri ottaa vastaan sovelluksen pyynnöt sekä ohjaa niiden perusteella ohjelmiston toimintaa. (Vahtolammi 2010)

Kuviossa 5 on esitetty kaavio Djangon MVC-arkkitehtuurin toteutuksesta. Vaikka MVC-malli vaikuttaa selkeältä, ei toiminnallisuuden sijoittaminen ole läheskään aina täysin yksiselitteistä. Silti mallin ja näkymän välinen työnjako on selvä, eli näkymä

ei sisällä tietovaraston käsittelyä eikä malli esimerkiksi HTML-merkintää. Django-ajax-lisäosa laajentaa Djangoa tarjoamalla yhteyden malleihin suoraan asiakaspuolen (engl. client-side) JavaScriptistä.



Kuvio 5. Djangon arkkitehtuurikaavio

MVC-arkkitehtuurin lisäksi Djangosta löytyy runsaasti valmiita sekä erikseen lisättäviä kolmannen osapuolen moduuleita helpottamaan Web-sovelluksessa tarvittavien asioiden, kuten autentikoinnin, monikielisyyden, HTML-lomakkeiden tai Ajax-tuen, toteuttamista.

Django sisältää ohjelmiston hallinta- ja ylläpitotehtäviä varten komentorivityökalun *django-admin.py* ja sitä käyttävän välikerroksen (engl. *wrapper*) *manage.py*, joka huolehtii projektin ja sen asetuksien saatavuudesta ennen syötetyn komennon ohjaamista *django-admin.py*-työkalulle. (Django-admin.py and manage.py 2013) Kyseisillä työkaluilla suoritettavia komentoja on esimerkiksi *startproject*, jolla luodaan hakemistorakenne uutta Django-projektia varten tai *syncdb*, jolla alustetaan ohjelmiston käyttämän tietokannan rakenne. Komentoja voi tehdä tarpeen mukaan myös itse.

2.3.2 ORM

Object-relational mapper on tekniikka, jolla tietomallit voidaan määritellä ohjelmointikielen luokkina ja näistä luokista tehtyjen olioiden kautta voidaan lisätä, muokata ja poistaa tietokannan sisältöä. Sen avulla voidaan jopa luoda koko tietokanta luokkien pohjalta. ORM mahdollistaa myös taustalla käytetyn tietokannan vaihtamisen ilman koodimuutoksia.

Perinteisesti suoraan tietokantaa käsiteltäessä on täytynyt käyttää tietokantaspesifistä kirjastoa, jonka avulla tietokantayhteys on avattu ja suljettu. Tietokannan tietoa kyseltäessä, lisättäessä tai muokatessa on sitten tarjottu SQL-kielisiä komentosarjoja. SQL eli *Structured Query Language* on IBM:än kehittämä standardoitu kyselykieli (Kansainvälinen standardoimisjärjestö ISO 2011), mutta eri tietokantatoteutuksilla on siihenkin omia murteita. Sen vuoksi vaihdettaessa tietokantaa vaikkapa MySQL:stä PostgreSQL:ään joudutaan yleensä muuttamaan koodiin kirjoitettuja SQL -kyselylauseitakin.

Kuvion 6 koodilistauksessa on havainnollistettu tietokannan käyttämistä ilman ORM:ää. Aluksi riveillä 1 ja 2 luodaan tietokantayhteys. Riveillä 5 ja 9 muodostetaan SQL -kyselylauseet ja ne suoritetaan riveillä 7 ja 13. Esimerkistä puuttuu kyselyn validointi ja virheiden käsittely kokonaan.

```
1  db = MySQLdb.connect(...)
2  cursor = db.cursor()
3
4  # Insert track into database
5  sql = 'INSERT INTO track' +
6        'SET title="Kappaleen nimi", duration="120"'
7  cursor.execute(sql)
8
9  # Query tracks from artist CMX
10 sql = 'SELECT * FROM track, artist' +
11       'WHERE track.artist=artist.id' +
12       'AND artist.name="CMX"'
13 cursor.execute_query(sql)
14
15 for track in cur.fetchall():
16   print track['title']
```

Kuvio 6. Tietokannan käyttö ilman ORM:ää

Kuvion 7 koodilistauksessa havainnollistetaan, kuinka tietokantaa käytetään ORM:llä. Tässä tapauksessa *Track*-luokka on periytetty ORM:n tarjoamasta pohjaluokasta, joka tarjoaa SQL -kyselyt piilottavan, kattavan ja dynaamisen rajapinnan tietokantakyselyihin.

```
1  # Insert track into database
2  track = Track(title='Kappaleen nimi', duration=120)
3  track.save()
4
5  # Query tracks from artist CMX
6  tracks = Track.objects.filter(artist__name='CMX')
7  for track in tracks:
8    print track.title
```

Kuvio 7. Tietokannan käyttö ORM:n avulla.

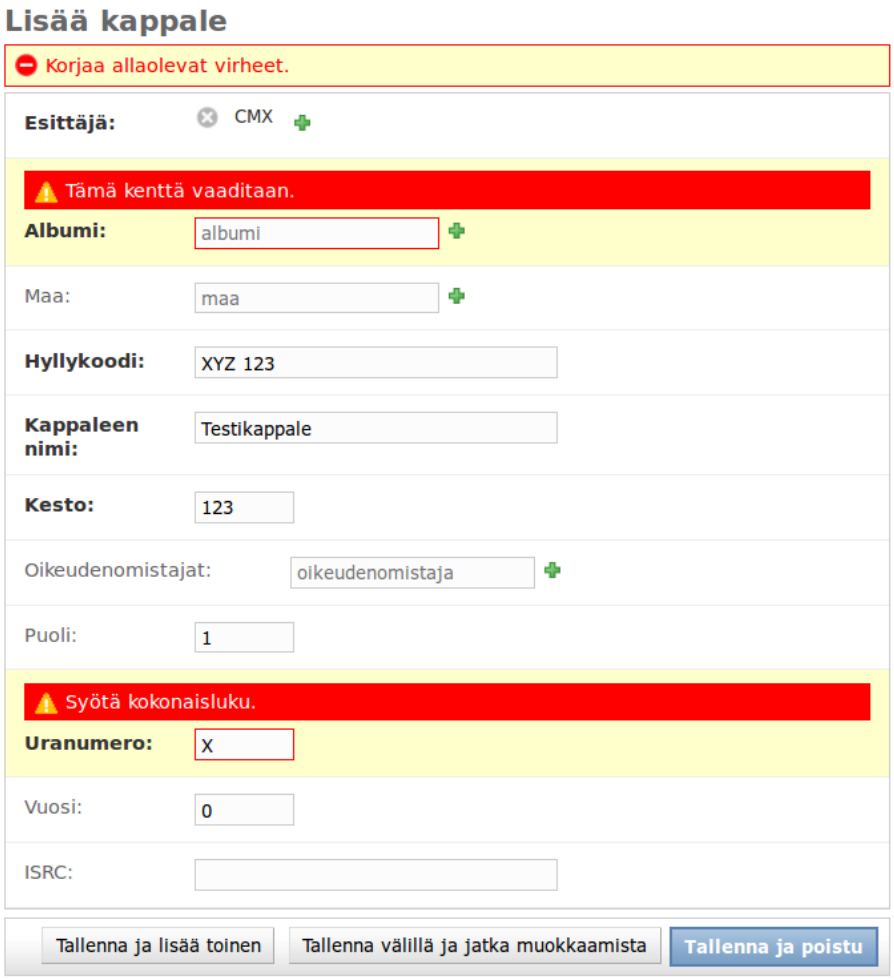
Kuviossa 8 on esitetty toteutetun *Track*-luokan avulla, kuinka Djangon ORM:n *Model*-luokasta periytetään halutunkaltainen tietomalli. Django tarjoaa tietokannan taulun kenttätyyppejä vastaavia luokkia runsaasti paketissa *django.db.models*. *PositiveIntegerField*:iin voi tallentaa positiivisen luvun tai *CharField*:iin merkkijonon. Kentät vastaavat esimerkiksi tässä yhteydessä alun perin tietokantaratkaisuksi valitussa PostgreSQL:sä *Integer*- ja *VarChar*-tyyppisiä kenttiä. *ForeignKey* ja *ManyToManyField* ovat yhteyksiä muihin tauluihin, ja niille annetaan parametreiksi vastaavat Djangon *Model*-luokasta perityt luokat.

```
1  import django.db.models
2
3  class Track(models.Model):
4    artist = models.ForeignKey(Artist)
5    album = models.ForeignKey(Album)
6    country = models.ForeignKey(Country, null=True,
7                                blank=True)
8    title = models.CharField(max_length=255)
9    duration = models.PositiveIntegerField()
10   proprietaries = models.ManyToManyField(Proprietary,
11                                          blank=True)
```

Kuvio 8. Riisuttu versio toteutetusta Track-luokasta

2.3.3 Hallintapaneeli

Django tarjoaa myös helposti mukautettavan ja laajennettavan hallintapaneelin tietokannan hallintaa varten. Perusliittymän saa tehtyä, kun luo ilmentymän Djangon *AdminSite*-luokasta ja rekisteröi siihen tietomallit, joiden taulujen tietoon haluaa päästä käsiksi hallintapaneelin kautta. Django lukee tietomallien metadataa ja muodostaa niiden pohjalta lomakkeet. Kuviossa 9 on esitetty Djangon *Track*-tietomallista automaattisesti luoma kappaleen lisäyslomake ja sen syötetietojen validointi käytössä.



Kuvio 9. Kappaleen lisäyslomake käytössä

Lomakkeen kentät luodaan tietomallissa määriteltyjen attribuuttien mukaan. Kyseinen lomake on luotu *Track*-luokasta, josta riisuttu versio esitettiin kuviossa 8. Kaikki kentät ovat oletuksena vaadittuja, mutta vapaaehtoiseksi sellaisen voi määritellä antamalla konstruktorille parametrin *blank=True*, kuten riveillä 6–7 sekä 10–11 on maan ja oikeudenomistajien kanssa menetelty.

Lomakkeita ja kenttiä voi tehdä hallintapaneeliin myös itse periyttämällä ne Djangon tarjoamista pohjaluokista. Lomakkeen pohjaluokasta löytyy toiminnallisuus esimerkiksi automaattiseen HTML-koodin generointiin ja syötetietojen oikeellisuuden tarkistukseen. Lomakkeet ja kentät eivät ole kuitenkaan sidottuja hallintapaneeliin, vaan ne ovat käytettävissä koko sovelluksen laajuisesti.

Djangon hallintapaneeliin kuuluu myös haku- ja suodatusominaisuuksilla varustettu selausnäkymä varsinaiseen tietosisältöön. Rekisteröimällä tietomallin ModelAdmin-luokan kautta voidaan toteuttaa erilaisia mukautuksia, esimerkiksi asettaa muokkauslomake tai muokattavat kentät itse, säätää selausnäkymässä näytetyt kentät ja järjestelyperusteet tai ohjelmoida toiminnallisuutta erilaisiin tapahtumiin.

2.3.4 URL-ohjain

Django mahdollistaa siistin ja elegantin URL-suunnittelun. Perinteinen URL:n rakenne on `http://palvelin/polku?muuttujat`-tyyppinen. Esimerkiksi URL:a `http://palvelin.fi/index.php?muuttuja=arvo&toinen=muuta` haettaessa verkkopalvelin ohjaa pyynnön PHP-moottorille index.php-sivun prosessoimiseksi. Kyseisen tiedoston PHP-koodista pääsee muuttujiin käsiksi *$_GET*-taulukon kautta, esimerkiksi

```
$_GET['muuttuja'] == 'arvo' // true
```

Djangossa URL:n käsittelyä varten luodaan kuviossa 10 esitetyn kaltainen muunnostaulukko, jossa URL-rakenteet liitetään Pythonin funktioihin. Rakenteet ovat yksinkertaisia säännöllisiä lausekkeita (engl. *regular expression regexp*), ja funktion sijasta voidaan viitata myös toiseen vastaavaan muunnostaulukkoon. Django käy listan läpi järjestyksessä ja ohjaa HTTP-pyynnön ensimmäiselle sellaiselle vastaantulevalle käsittelijälle, jonka edessä oleva säännöllinen lauseke vastaa pyynnön URL:a (URL dispatcher 2013).

```
1  urlpatterns = patterns('',
2    (r'^$', 'app.views.audiolibrary'),
3    (r'^users/(?P<user>\w+)/$', 'app.views.profile'),
4    (r'^admin/', include(admin.site.urls))
5  )
```

Kuvio 10. URL:n muunnostaulukko

Varsinaiset URL-rakenteet ovat esimerkkilistauksen riveillä 2, 3 ja 4. Ennen merkkijonoja oleva *r*-kirjain on Pythonin syntaksi, jonka avulla tietyt erikoismerkit, kuten säännöllisissä lausekkeissa paljon käytetty kenoviiva, voidaan kirjoittaa merkkijonoon sellaisenaan. URL:sta saa kerättyä muuttujien arvoja säännölliseen lausekkeeseen kirjoitettujen tavallisten sulkujen avulla ja Django ohjaa nämä arvot parametreina pyynnön käsittelijälle.

Esimerkin säännölliset lausekkeet alkavat sirkumfleksillä, joka tarkoittaa säännöllisen lausekkeen ensimmäisenä merkkinä, että kyseinen kohta täytyy löytyä rivin alusta. Vastaavasti dollarilla tarkoitetaan rivin loppua. Näin ollen esimerkkilistauksen toisen rivin lauseke *'^$'* vastaa osoitetta, jonka loppu seuraa välittömästi sen alkua eli käytännössä osoittaa sivuston juureen. Kolmannella rivillä kerätään käyttäjätunnus osoitteesta ja välitetään se *app.views.profile*-funktiolle *user*-nimisenä parametrina. Kuvion 11 listauksessa on esimerkkitoiminnallisuus kyseiselle funktiolle.

```
1  def profile(request, user):
2    return render(request, 'profile.html',
3                  user=User.get(name=user))
```

Kuvio 11. Esimerkki HTTP-pyynnön käsittelijäfunktiosta

Esimerkkitapauksessa Djangon saadessa HTTP-pyynnön osoitteeseen *http://sivusto.fi/users/johndoe/* saa *user*-parametri arvokseen merkkijonon *'johndoe'*. Rivillä 3 haetaan näkymän käyttöön johndoe-nimisen käyttäjän ORM-malli, jolloin HTTP-vastauksena voidaan käyttäjälle palauttaa profiilisivu johndoen tiedoilla.

2.3.5 Autentikointi

Djangon tarjoama autentikointimoduuli sisältää toiminnallisuuden sekä käyttäjän tunnistukseen että hänen käyttöoikeuksiensa määrittelyyn. Sen tietomalleihin sisältyy käyttäjät (*User*), ryhmät (*Group*) ja oikeudet (*Permission*). Djangon autentikointijärjestelmä tarjoaa myös lomakkeet kirjautumista ja rekisteröitymistä varten sekä kattavan rajapinnan aiheeseen liittyvään toiminnallisuuteen kuten käyttäjien luontiin ja tunnistukseen tai heidän salasanansa vaihtoon tai käyttöoikeuksiensa tarkistamiseen.

Autentikointijärjestelmä asentuu Djangoon niin tiiviisti, että kontrollereista pääsee kirjautuneeseen käyttäjään suoraan käsiksi parametrina saamansa HTTP-pyyntöolion *user*-muuttujan kautta, joka saa arvon *None*, kun ei olla kirjauduttu. Django tarjoaa myös asiaan liittyviä dekoraattoreita kuten *login_required* ohjaamaan kirjautumattomat kirjautumissivulle, *permission_required* estämään oikeudettomien käyttäjien toimia ja *user_passes_test* antamaan mahdollisuuden liittää oma funktio päättämään käyttäjän mahdollisuudesta tehdä tiettyä sivunlatausta tai muuta toimenpidettä.

2.3.6 Lokalisointi

Djangossa on täysi tuki tekstien lokalisointiin ja päiväyksien, aikojen sekä lukujen esitystavan määrittelemiseen. Tekstien lokalisoimiseksi kaikki käännettävät tekstit tulee saattaa Djangon tietoon. Koodissa ja mallinteissa olevista käännettävistä merkkijonoista tehdään käännöstiedosto *django-admin.py makemessages* -komennolla, joka käy tiedostot läpi ja muodostaa niistä tekstitiedoston käännettäväksi. Tiedostoon tehtävän käännöksen kieli annetaan komennolle parametrina.

Merkkijonot määritellään käännettäväksi Djangon mallinteessa komennolla *trans* ja Python-koodissa käyttäen funktiota *ugettext()*, joka on kätevää ladata käyttöön lyhyemmällä nimellä esimerkiksi *_()*. Käännettävissä merkkijonoissa voi käyttää myös parametreja ja käännöstiedostoon saa käännettävien tekstien yhteyteen kommentteja tekstin kääntäjiä varten. Kuvioissa 12 ja 13 on esitetty, kuinka lokalisointijärjestelmää käytetään sekä Python-koodista että mallinteesta.

```
1  def foo(bar, date):
2    # Translators: take into account that xxx...
3    viesti = _('Today is %(month)s/%(day)s/%(year)s.') %
4             {'month': date.month,
5              'day': date.day,
6              'year': date.year}
```

Kuvio 12. Esimerkki Python-tiedoston lokalisoinnista

```
1  {# Translators: this is for yyy... #}
2  {% trans "Something to translate." %}
```

Kuvio 13. Esimerkki Djangon mallinteen lokalisoinnista

Näistä esimerkkitiedostoista Django kokoaisi käännettäväksi kuviossa 14 esitetyn tiedoston. Tuon tiedoston *msgstr*-merkkijonoihin merkitään edeltävän *msgid*:n käännös, kuten on havainnollistettu kuvion 15 listauksessa. Merkillepantavaa on kolmannella ja neljännellä rivillä esitetty parametrien käyttö. Tästä tiedostosta täytyy vielä kääntää optimoidut binääritiedostot komennolla *django-admin.py compilemessages*.

```
1  #. Translator: take into account that xxx...
2  # path/to/file.py:10
3  msgid "Today is %(month)s/%(day)s/%(year)s."
4  msgstr ""
5
6  #. Translator: this is for yyy...
7  # path/to/template.html:25
8  msgid "Something to translate."
9  msgstr ""
```

Kuvio 14. Esimerkki Djangon kokoamasta käännöstiedostosta

```
#. Translator: take into account that xxx...
# path/to/file.py:10
msgid "Today is %(month)s/%(day)s/%(year)s."
msgstr "Tänään on %(day)s.%(month)s.%(year)s."

#. Translator: this is for yyy...
# path/to/template.html:25
msgid "Something to translate."
msgstr "Jotain käänneltävää."
```

Kuvio 15. Tekstien kääntäminen käännöstiedostoon

Djangon lokalisointijärjestelmä mahdollistaa käännösten tekemisen myös JavaScript-lähdekoodista sekä URL-rakenteesta. Django valitsee käytetyn kielen niin, että ensin tutkitaan, onko kieltä määritelty URL:ssa. Mikäli ei, katsotaan onko kieliasetus määritelty istunnossa, evästeessä tai selaimen lähettämässä HTTP-otsikossa *Accept-Language*. Jos kieli ei selvinnyt, käytetään Django-sovelluksen asetuksissa määriteltyä globaalia *LANGUAGE_CODE*-asetusta.

2.3.7 Mallinteet

Djangon mallinnemoottori (engl. *template engine*) tarjoaa tehokkaan kielen esitys- ja toimintalogiikan erottamiseksi. Mallinteita voi kirjoittaa kuka hyvänsä HTML-kuvauskieltä taitava ilman ymmärrystä Pythonista. (Templates 2013) Mallinteesta saadaan käännettyä HTML-sivu *render*-funktiolla, jolle annetaan mallinteen lisäksi käytetty konteksti (engl. *context*). Konteksti on hajautustaulu mallinteelle käytettävissä olevista muuttujista.

Djangon mallinnekielestä on esitetty esimerkki kuviossa 16. Mallinnekielen ohjauskomennot kirjoitetaan aaltosulkujen sisään. Kaikki tekstit näiden ulkopuolelta tulostuu sellaisenaan. Kieli sisältää yleisten *if-else* -rakenteiden lisäksi komentoja ja suodattimia tiedon esittämiseen. Esimerkissä käytetty *trans*-komento lokalisoi tekstin ja putkimerkin jälkeinen *capfirst*-suodatin kapitalisoi ensimmäisen kirjaimen.

```
1  {% if user.is_active %}
2    {% trans 'welcome'|capfirst %},
3       <strong>
4         {% user.get_username %}
5       </strong>.
6
7       {% block userlinks %}
8         {% if user.has_usable_password %}
9         <a href="{% url 'admin:password_change' %}">
10          {% trans 'change password'|capfirst %}
11        </a>
12        {% endif %}
13
14        <a href="/accounts/logout">
15          {% trans 'logout'|capfirst %}
16        </a>
17      {% endblock %}
18
19 {% else %}
20   <a href="/accounts/login">
21     {% trans 'login'|capfirst %}
22   </a>
23 {% endif %}
```

Kuvio 16. Esimerkkimallinne

Kun kyseinen esimerkkimallinne käännetään *user*-muuttujana aktiivinen käyttäjä *Matti Meikäläinen*, saadaan kuviossa 17 esitetty tulos. Kirjautumattoman käyttäjän kohdalla mallinnemoottori kääntäisi vain riveillä 20–22 olevan sisäänkirjautumislinkin havaittuaan rivillä 1 olevan testin epätodeksi.

```
1    Tervetuloa, <strong>Matti Meikäläinen</strong>.
2
3        <a href="/admin/change_password">
4          Vaihda salasanaa
5        </a>
6
7        <a href="/accounts/logout">
8          Kirjaudu ulos
9        </a>
```

Kuvio 17. Esimerkki mallinteesta käännetystä sivusta

3 TOTEUTUS

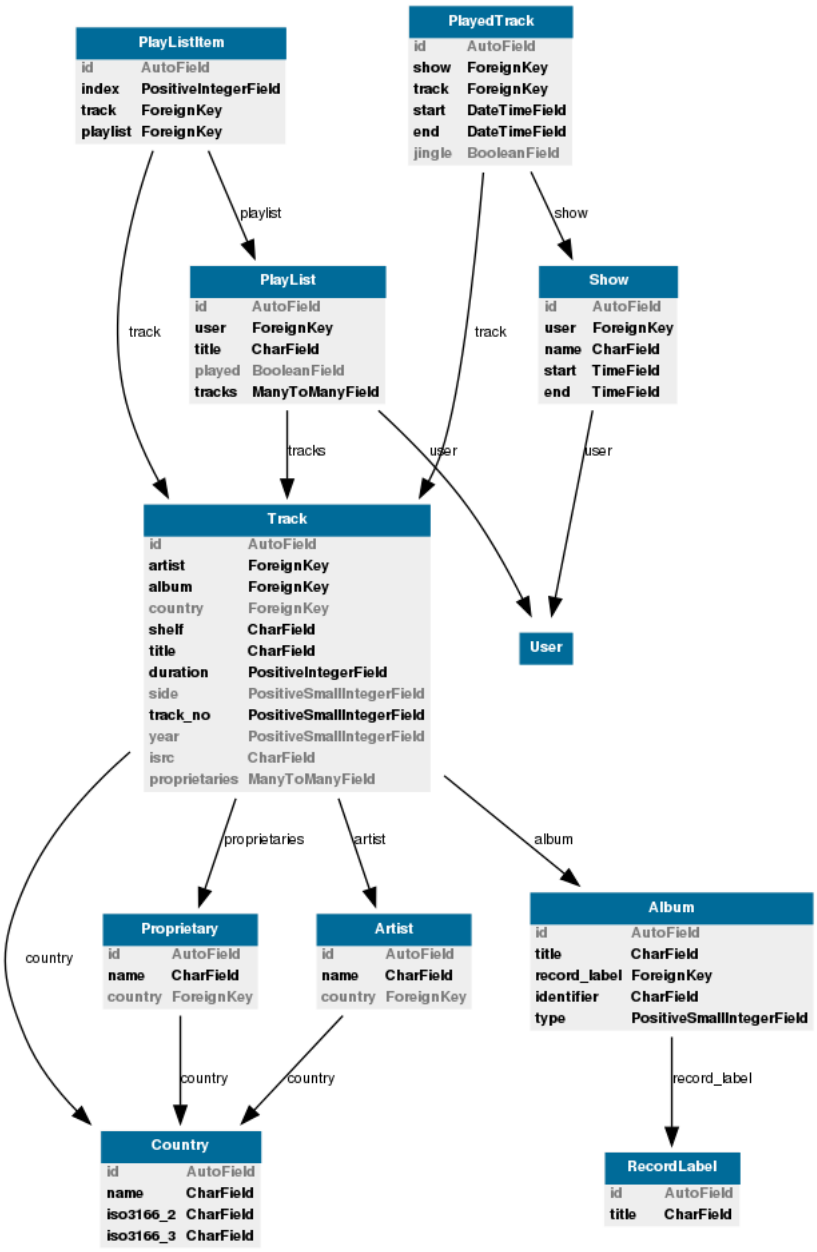
3.1 Tietokanta

Tietokannan suunnittelussa määrääviä ja rajoittavia tekijöitä olivat vanhan tietokannan sisältö sekä kaupallisten radioiden raportointiohje. Vaikka raportointiohjeessa eivät kaikki kentät ole pakollisia, ne päätettiin silti tallentaa ja ilmoittaa raportissa.

Vanha levystö oli Access-tietokannan yhdessä taulussa, johon oli manuaalisesti kirjoitettu jokaisen levyn kaikki tiedot erikseen. Tämän vuoksi kantaan on päässyt jonkin verran virheitä ja siitä johtuen tietojen oikeellisuuteen ei voinut luottaa.

Kappaleen hyllykoodi muodostuu albumin hyllykoodista, raidan järjestysnumerosta sekä tiedosta, onko kappale esimerkiksi albumin ensimmäisellä vai toisella levyllä tai kasetin A- vai B-puolella. Kappaleen hyllykoodi oli kuitenkin laitettava tietokannan kappaletauluun, sillä kirjoitusvirheistä johtuen ei ollut olemassa luotettavaa tietoa albumin hyllykoodista. Esimerkiksi jos levykoodilla *XYZ123* on kolme kappaletta, joiden hyllykoodit ovat *Xx A 1*, *Xy A 2* ja *Xz A 3*, niin annetuista tiedoista ei voi päätellä, onko albumin hyllykoodi *Xx*, *Xy* vai *Xz*. Tämä oli otettava huomioon tietokantaa suunnitellessa, sillä vanha tietokanta oli niin mittava, että sen tuominen automaattisesti oli välttämättömyys.

Kuviossa 18 on esitetty ER-kaavio toteutetun tietokannan rakenteesta. Yleiskäsityksen muodostamiseksi tarkastellaan tietokantatauluja ja niiden välisiä yhteyksiä. Taulussa *User* on ohjelmiston käyttäjä. Tämä taulu tulee Djangon autentikointimoduulin myötä.



Kuvio 18. ER-kaavio tietokannan rakenteesta

Käyttäjillä soittolistoja *PlayList*, jossa on kappaleita *Track*. Yhteys on moni-moneen, eli yksi kappale voi olla monella soittolistalla ja yhdellä soittolistalla voi olla monta kappaletta. Kappaleella on esittäjä *Artist* ja se on nauhoitettu levylle *Album*. Kappale on nimenomaan *kappale levyllä*, joten vaikka esittäjän kappale olisi täsmälleen sama singlellä, albumilla ja kokoelmalevyllä, kuuluu jokaiselle oma rivi *Track*-taulussa.

Levy-yhtiöillä on yksi tai useampi levymerkki *RecordLabel*, joilla tuotantoa julkaistaan. *Proprietary* on kappaleen oikeudenomistaja. *Country* viittaa oikeudenomistajan sekä artistin kohdalla kansalaisuuteen ja kappaleen kohdalla nauhoitusmaahan. Soi-

tetut kappaleet tallennetaan raportointia varten tauluun *PlayedTrack* ja kappale on soitettu jonkin ohjelman *Show* aikana.

Tietokantapalvelin

Hankitulla vuokrapalvelimella oli tietokantaohjelmistoksi valittavissa PostgreSQL ja MySQL, joista alkujaan valittiin käyttöön ensimmäisenä mainittu. Tietokantaa kokeillessa kuitenkin huomattiin, että haku oli kirjaimia *A*:sta *Z*:aan lukuun ottamatta merkkikokoriippuvainen. Siis tietokantaan tallennettua kappaletta *SYKSYN SÄVEL* ei löytynyt hakusanalla *sävel*, sillä tietokanta piti kirjainta *ä* ja sen kapitalisoitua versiota *Ä* eri kirjaimena.

PostgreSQL sisältää toiminnallisuuden tämän käyttäytymistavan muuttamiseen, mikäli käyttöjärjestelmästä löytyy sille tuki. Tällöin kuitenkin käytettävä merkistö tulee asettaa tietokantaa luotaessa, eikä sitä voi kyseisessä tietokantaohjelmistossa muuttaa myöhemmin (Collation Support 2013). Vuokratussa palvelussa tietokanta täytyi luoda palveluntarjoajan työkalulla, joka ei tukenut kyseistä toimenpidettä. Toinen vaihtoehto olisi ollut päivittää kalliimpaan palvelupakettiin. Kätevin ratkaisu oli siis vaihtaa ohjelmiston tietokannan taustajärjestelmä MySQL:ään, mikä onnistui Djangon ORM:n avulla melko vaivattomasti.

3.2 Olemassa olevan tiedon tuonti

Vanhan tietokannan tietojen analysointi ja tuonti osoittautui ennakko-oletusta vaivalloisemmaksi. Vanha tietokanta oli niin vanhalla Accessilla tehty, ettei sitä saanut auki sellaisilla Access-versioilla, joiden käyttöön meillä oli mahdollisuus. Järviradion henkilökunta onnistui kuitenkin lopulta viemään tietokannan XML-muotoon.

Viedyllä XML-tiedostolla oli sen verran kokoa, että informaatio täytyi lukea tiedostovirrasta.

Virta tarkoittaa tässä yhteydessä menetelmää, jolla luettava tiedostoa ladataan muistiin vain tarvittava määrä kerrallaan sen sijaan, että se luettaisiin aluksi kokonaan

muistiin. Pythonille on XML:n käsittelyyn runsaasti erilaisia kirjastoja, joista kokeilun jälkeen *ElementTree* osoittautui selkeytensä, tehokkuutensa ja virrasta lukemisen tuen ansiosta tähän tapaukseen päteväksi valinnaksi. Seuraavaksi haasteeksi osoittautui se, ettei asiakkaalta saatu XML ollut standardin mukaista. Tätä ElementTree ei hyväksynyt. Esimerkiksi elementtien nimissä oli standardin vastaisesti risuaitoja sekä välilyöntejä.

Tilanteesta teki vielä haastavamman se, ettei tiedostoa saanut suuren kokonsa vuoksi tarkasteltua moderneillakaan tekstieditoreilla. Tähän auttoi Unixeista tutut komentorivityökalut kuten *head*, *tail* ja *sed*. Työkaluilla head ja tail saa tulostettua komentoriville tiedoston alusta tai lopusta haluamansa määrän rivejä ja komentotulkin putkitusominaisuudella näiden avulla saa tiedoston keskeltä rivejä nähtäväksi. Esimerkiksi komennolla

```
tail -1000 tiedosto | head -10
```

tulostuu 10 ensimmäistä riviä *tiedosto*n viimeisestä 1000 rivistä. Tiedoston muokkaaminen standardinmukaiseen kuntoon hoitui sed:llä ja Perlillä.

Tietokantatiedostoa jäsenneltiin lukemalla XML:ää kunnes täyteen tuli kokonainen kappaletietue. Kuviossa 19 on esimerkki tällaisesta tietueesta. Albumien otsikkoja ei vanhaan tietokantaan ollut merkitty lainkaan, joten albumeita luotaessa käytettiin pelkkää levykoodia. Artistit luotiin *ESITTAJA*-elementistä löytyneen nimen perusteella.

```
<RECORD>
  <Lahetyspaivamaara>111126</Lahetyspaivamaara>
  <Lahetyksenalku>2200</Lahetyksenalku>
  <Lahetyksenloppu>0000</Lahetyksenloppu>
  <Huomautus1></Huomautus1>
  <Huomautus2></Huomautus2>
  <HYLLY>G 10595 CD 010</HYLLY>
  <TEOSNIMI>KAIPAAN SUA</TEOSNIMI>
  <ESITTAJA>SORSAKOSKI TOPI</ESITTAJA>
  <SIVU>1</SIVU>
  <URA>10</URA>
  <KESTO>0232</KESTO>
  <OIKOM1>TRAD</OIKOM1>
  <OIKOM2></OIKOM2>
  <OIKOM3></OIKOM3>
  <OIKOM4></OIKOM4>
  <OIKOM5></OIKOM5>
  <MAAkoodi>1</MAAkoodi>
  <LEVYMERKKI>FINPILE</LEVYMERKKI>
  <LEVYKOODI>305</LEVYKOODI>
  <Tallennusmaa>SF</Tallennusmaa>
  <LAJI>1</LAJI>
  <Relay1></Relay1>
</RECORD>
```

Kuvio 19. Esimerkkitietue käsitellystä XML-tiedostosta

Erilaisista kirjoitusasuista sekä kirjoitusvirheistä johtuen yhdestä artistista saattoi tulla muutamia eri kirjoitusasun omaavia versioita. Esimerkiksi *Juice Leskisen* kappaleita voi esiintyä tietokannassa artistinimillä *Juice Leskinen*, *Leskinen Juice etc*, *Leskinen Juice atc.* tai Juice. Myös levy voi olla jaettuna, mikäli levykoodia merkitessä on tapahtunut kirjoitusvirheitä. Näistä korjasimme selkeimmät käsin hakemalla tietokannasta sellaiset albumit, joilla ei ollut kuin yksi kappale.

Kaupallisten radioiden raportointiohjeessa maatieto pyydetään ISO 3166-1 alpha-2 -standardin mukaisena kaksikirjaimisena koodina. Maat koodeineen saatiin tietokantaan Pythonin *iso3166*-kirjastosta. Vanhasta tietokannasta löytyi jonkin verran standardin vastaisia maakoodeja. Tieto ei kuitenkaan ole tekijänoikeusraporteissa pakollinen, joten vain selkeitä tapauksia varten, kuten muunnos *SF*:stä *FI*:ksi, tehtiin muunnostaulu. Jos kirjainyhdistelmästä, kuten *SS*, ei ollut mahdollista päätellä mihin maahan koodilla viitataan, jätettiin kenttä tyhjäksi.

Tietokantarakenne tehtiin kuitenkin sellaiseksi, että mainitut ongelmat eivät estä ohjelman käyttöä. Kappaleiden haku ja itse äänitteen löytäminen kappaleen soittamiseksi onnistuu samalla tavalla kuin edellisen järjestelmän aikana. Vanhan tiedon jäsentelyn yhteydessä ilmenneiden virheiden korjaamiseen on toteutettavissa melko tehokkaitakin menetelmiä, joita on pohdittu ohjelmiston tulevaisuutta käsittelevässä kappaleessa.

3.3 Asiakasohjelma

Asiakasohjelma, eli ohjelmiston käyttöliittymäpuoli, toteutettiin WWW-selaimella käytettäväksi. Radion kuuntelijat pääsevät hakemaan levystöstä kappaleita sekä seuraamaan statistiikkaa. Henkilökunnalla on kirjautumisen jälkeen pääsy myös työkaluja hallinta-osioihin, sekä mahdollisuus luoda soittolistoja levystönäkymässä. Toteutettu asiakasohjelma käyttää JavaScriptiin nojaavia Ajax-tekniikoita, joissa sivun sisältöä päivitetään ohjelmallisesti palvelimelta pyydettyjen tietojen mukaan.

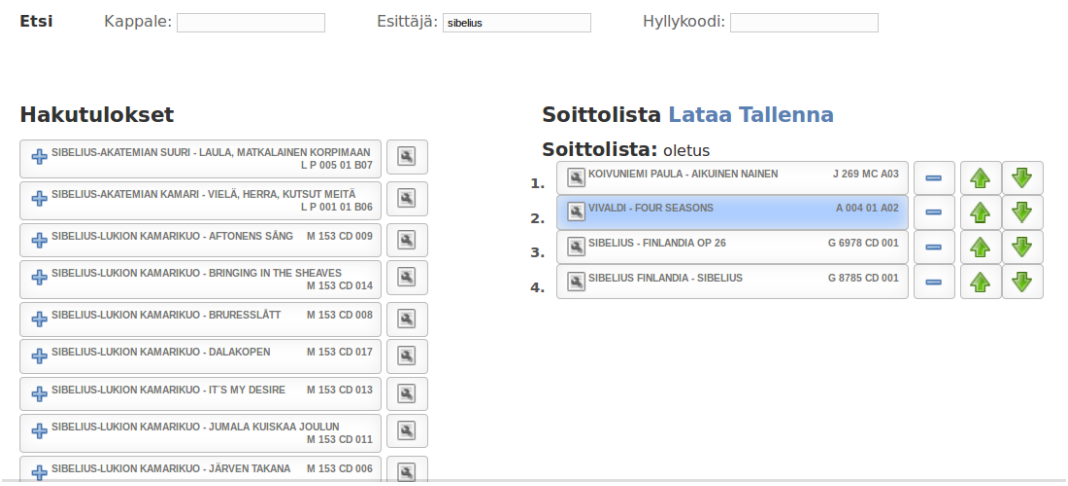
3.3.1 Levystö

Levystöosion käyttöliittymän ulkoasu ja toiminnallisuus riippuvat siitä onko käyttäjä kirjautuneena. Kirjautumattomat käyttäjät voivat käyttää levystöosiota vain Järviradion levytietokannan tietojen tutkimiseen. Näin kuuntelijat voivat esimerkiksi etsiä toivekappaleitaan levystöstä esittäjän nimen, kappaleen otsikon ja hyllykoodin perusteella. Usean hakukentän täyttäminen tuo rajatun hakutuloksen. Eli jos kappalehakukenttään syötetään esimerkiksi hakusana *nainen* ja artistihakukenttään *Koivuniemi*, niin hakutulokseksi saadaan *Paula Koivuniemen* kappale *Aikuinen nainen* ja muiden esittäjien kappaleet, joiden nimessä esiintyy sana *nainen*, jäävät pois.

Kaikki käyttäjät voivat myös katsoa lisätietoja kappaleesta klikkaamalla kappaleen nimen oikealla puolella olevaa kysymysmerkkiä. Käyttäjälle näkyvä lisätietoikkunasta ruutukaappaus kuviossa 20 Tällöin käyttäjä pääsee näkemään tarkempia yksityiskohtia kappaleesta, kuten keston ja kappaleen oikeudenomistajat.

Kuvio 20. Kappaleen tiedot omassa JavaScriptillä toteutetussa ikkunassaan

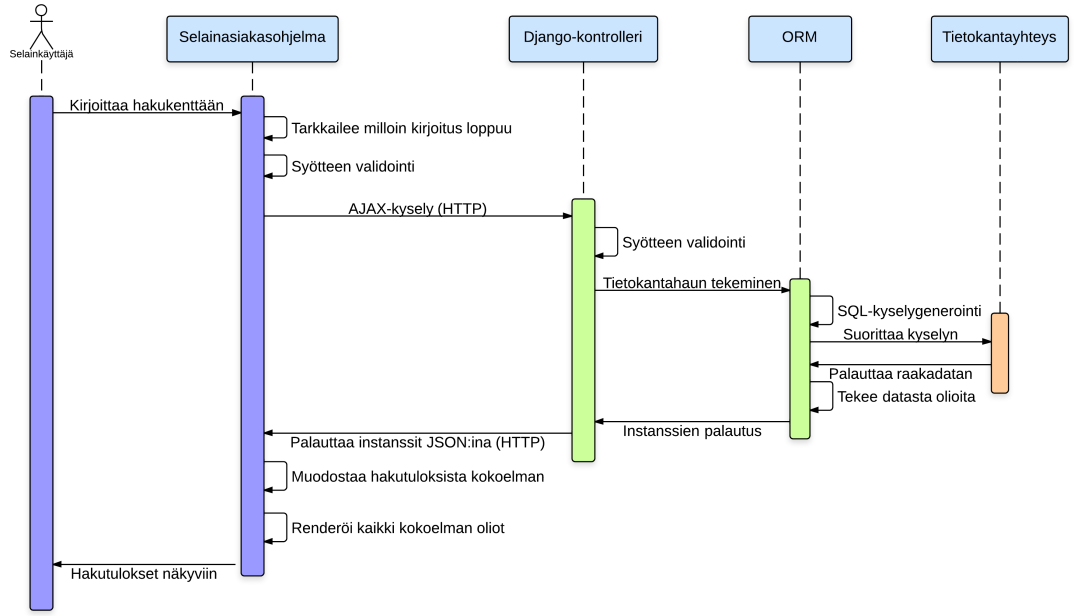
Kirjautuneet käyttäjät voivat luoda levystöosiolla henkilökohtaisia soittolistoja, jotka tallentuvat palvelimelle. Näin käyttäjä saa aina käyttöönsä omat soittolistat riippumatta tietokoneesta tai selaimesta, jota hän käyttää. Kirjautuneen käyttäjän näkymää esittää kuvio 21. Valitulle soittolistalle lisätään kappaleita valitsemalla niitä hakutuloslistalta. Jos käyttäjällä ei ole yhtään soittolistaa, järjestelmä luo sellaisen automaattisesti. Käyttäjä voi myös poistaa soittolistalta kappaleita tai muuttaa niiden soittojärjestystä. Kun soittolistan kappaleiden järjestystä muutetaan, asiakasohjelma odottaa hetken viimeisestä muutoksesta ennen palvelimelle tallentamista, jottei palvelimella olevaan soittolistaan tarvitse päivittää joka muutosta erikseen. Tämä vähentää tietoliikenteen määrää merkittävästi sekä toimii paremmin virhetilanteissa, kuten internetyhteyden väliaikaisessa katkeamisessa. Levystönäkymässä tehdyt soittolistat voi merkitä soitetuksi työkaluosiossa, jolloin ne tulevat näkyviin tehtyihin tekijänoikeusraportteihin.



Kuvio 21. Ruutukaappaus kirjautuneen käyttäjän levystönäkymästä

Kuvio 22 esittää toteutetun levystöhaun viestikaavion. Siinä asiakaspuolen tapahtumia edustavat violetit pylväät ja palvelinpuolta edustavat vihreät pylväät. Näiden välinen yhteys käyttää http-protokolaa. Syötteen validointi tehdään sekä asiakas- että palvelinpuolella tietoliikenteen määrän vähentämiseksi sekä tietoturvan vuoksi.

Käyttäjän kirjoittaessa hakusanan hakukenttään lähettää asiakasohjelma ajax-pyynnnön, jonka mukana on validoitu hakusana. Palvelin palauttaa hakutulokset HTTP-vastauksen mukana JSON:na, joka sisältää listan kuvion 24 mukaisia avain-arvopareja, jotka edustavat kappaleen tietueiden nimiä ja niiden arvoja. Tämä tieto jäsennellään (engl. *parse*) olioiksi, joista muodostetaan hakutuloslista näkyville käyttäjän selaimeen.



Kuvio 22. Viestikaavio levystöhausta

Selainasiakasohjelma toteutettiin Backbone.js-kehysympäristöön, joka on tehty helpottamaan HTML-näkymän erottamista omalle tasolleen käyttöliittymälogiikasta. Ympäristö tarjoaa rajapinnan näkymien, malliluokkien ja mallikokoelmien tekoon. Backbonessa näkymien, mallien ja kokoelmien välille voidaan tehdä melko helposti sidoksia, jolloin esimerkiksi käyttöliittymässä lomakkeeseen tehdyt muutokset päivittyvät automaattisesti lomakkeen tietoihin kiinnitettyyn olioon.

Kuviossa 23 yksinkertaistettu esimerkki kappaletta edustavasta Backbone-malliluokasta. Siinä rivillä 2 määritellään mallille avain, jonka avulla olio löytää uniikin tunnisteensa kun palautettua JSON:a jäsennellään. Tunniste on määritelty esimerkki JSON:ssa kuviossa 24 rivillä 7. Tunnisteen avulla kehysympäristön on mahdollista tehdä muutoksia vastaaviin riveihin etäällä olevaan tietokantaan.

```
1  Track = Backbone.Model.extend {
2    idAttribute: 'pk'
3    urlRoot: '/ajax/audiolibrary/track'
4  }
```

Kuvio 23. Backbone Kappaleluokan määrittäminen

Rivillä 3 kuviossa 23 määritellään luokalle ajax-osoite, johon kyseisen luokan olioiden muokkaamiseen liittyvät ajax-kutsut tehdään. Luokkaan ei erikseen tarvitse määritellä attribuutteja, sillä se saa JSON:ssa olevat avain-arvoparit attribuutikseen. Koska

Backbone-malliluokan määritys ei ota kantaa jäsenneltävään tietoon, ei asiakasohjelman luokkia tarvitse muuttaa esimerkiksi Djangon-tietokantojen muuttuessa. Kuitenkin halutessaan malliluokalle voi määritellä esimerkiksi attribuuttien oletusarvoja tai tarvittavia metodeja.

```
1  {
2    "album": 11441,
3    "title": "AAMUTÄHTI",
4    "country": null,
5    "artist": 1101,
6    "shelf": "G 3171 CD 004",
7    "pk": 76743,
8    "track_no": 3,
9    "proprietaries": [3893,3895,5578],
10   "year": 0,
11   "duration": 271,
12   "isrc": "",
13   "side": 1
14 }
```

Kuvio 24. Yhden kappaleen otos JSON-hakutuloksista.

Kuviossa 25 on yksinkertaistettu versio asiakasohjelman hakukokoelman toteutuksesta. Rivillä 3 määritellään kokoelman alkion tyyppi, joka tässä tapauksessa on kuviossa 23 esitelty kappale-malliluokka eli Track. Kokoelmalle on määritelty riviltä 4 lähtien search-niminen aliohjelma, Tämä aliohjelma suorittaa ajax-kyselyn rivilla 2 asetettuun URL:iin.

```
1  SearchTrackCollection = TrackCollection.extend {
2    url: '/ajax/audiolibrary/search_tracks.json'
3    model: Track
4    search: (search) ->
5      @fetch {
6        data: {
7          search_word: search
8        }
9      }
10 }
```

Kuvio 25. Backbone luokan määritys hakutuloskokoelmalle

Backbonella kokoelma tai yksittäinen olio voidaan kiinnittää näkymään. Näkymien tehtävä on toimia käyttöliittymästä tulevan syötteen vastaanottajana ja selaimes-

sa näkyvän käyttöliittymän interaktiivisen sisällön tuottajana. Kuviossa 26 hakutuloslistan näkymäluokka asetetaan tarkkailemaan omistamansa kokoelman muutoksia. Tämä tehdään sitomalla näkymän metodeja kokoelman tapahtumiin kokoelman *on*-aliohjelmalla. Tämä tehdään kuviossa riveillä 5 ja 6. Rivillä 7 asetetaan näkymä kuuntelemaan viestikanavaa (engl. message bus), jonka kautta hakukäskyt tulevat.

```
1  SearchTrackListView = TrackListView.extend {
2    tagName: 'ul'
3    initialize: ->
4      @tracks = new SearchTrackCollection
5      @tracks.on 'sync', => do @tracks_synced
6      @tracks.on 'reset', => do @tracks_synced
7      Backbone.on 'set_search', @tracks.search
8  }
```

Kuvio 26. Backbone-näkymäluokan toteutus hakutuloksille

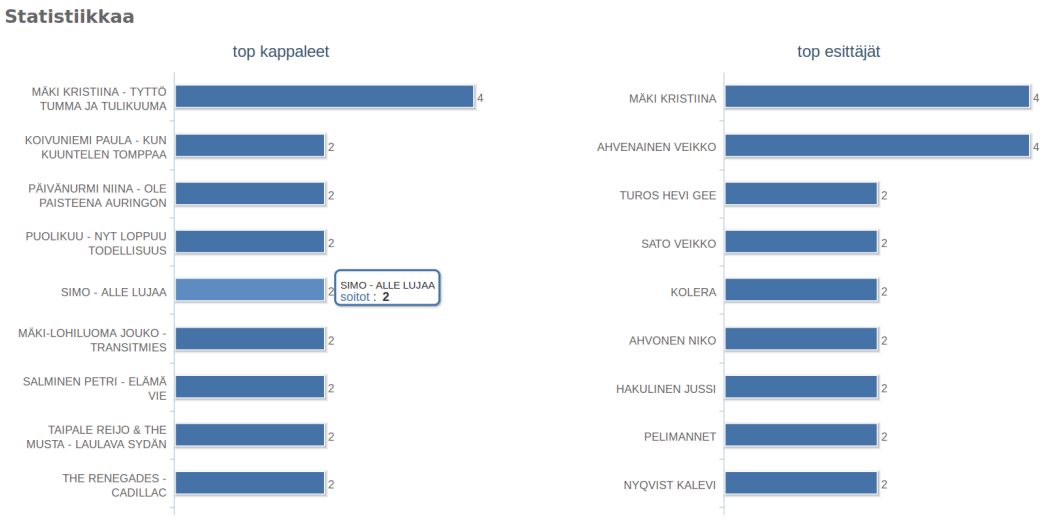
Kuviossa 27 on listattuna riisuttu versio yksittäisen kappaleen näkymäluokasta. Se on kiinnitettynä HTML-elementtiin, jonka tapahtumia näkymäolio kuuntelee. Pelkällä hakutuloksella ei ole kuin yksi toiminto, joka määritellään riveillä 3, 4 ja 5. render-aliohjelmaa kutsutaan, kun näkymä pitää päivittää esimerkiksi olion tietojen muuttuessa.

```
1  TrackListItemView = Backbone.View.extend {
2    tagName: 'li'
3    events: {
4      'click .trackitem': 'on_click'
5    }
6
7    render: ->
8      @$el.html @template {
9        track: @track.toJSON(),
10       artist: @artist.toJSON()
11     }
12     return @
13 }
```

Kuvio 27. Backbone -näkymäluokan toteutus kappaleen esittämiselle

3.3.2 Statistiikka

Kuviossa 28 on ruutukaappaus toteutetusta statistiikkaosiosta, jossa esitetään pylväskaaviot soitetuimmista kappaleista ja esittäjistä.



Kuvio 28. Ruutukaappaus asiakasohjelman statistiikka-osiosta

Aineisto tilastoihin kerätään palvelimella, josta se palautetaan JSON-muodossa asiakasohjelmalle, josta se piirtää pylväskaaviot Highcharts-nimisellä JavaScript -kirjastolla. Kyseisellä kirjastolla saa tehtyä moderneja ja interaktiivisia kaavioita web-sivuille. Highcharts tukee pylväskaavioiden lisäksi yleisempiä kaaviotyyppejä, kuten ympyrä-, alue- ja viivakaavioita.

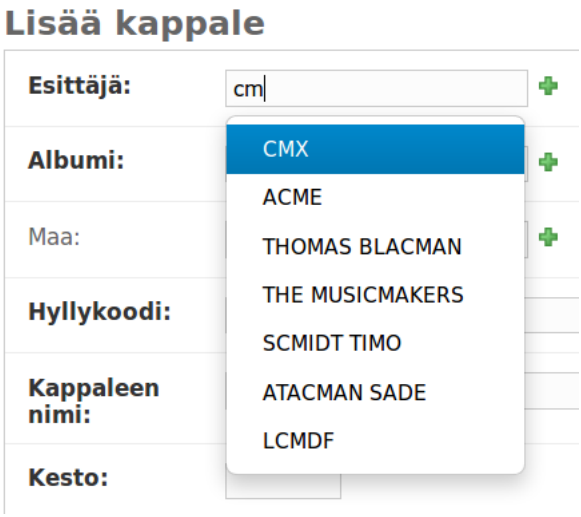
3.3.3 Työkalut

Työkaluosiossa ei ole tällä hetkellä kuin yksi varsinainen toiminto, jonka avulla henkilökunta voi merkitä soittolistan soitetuksi. Jotta soitot voidaan raportoida oikein, täytyy käyttäjän syöttää lomakkeeseen soittolista, jonka kappaleet merkitään soitetuiksi sekä ohjelma, jonka aikana kappale on soitettu tai sen ajankohta. Tämän jälkeen asiakasohjelma esittää käyttäjälle listan soitetuiksi merkattavista kappaleista, jotta käyttäjä voi varmistua tietojen oikeellisuudesta ennen lopullista hyväksymistä. Lomake osaa käsitellä tyypillisimmät virhetilanteet esimerkiksi, kun kappaleiden yhteispituus on pidempi kuin lähetyksen kesto.

3.3.4 Hallinta

Ohjelmiston hallintapuoli pohjautuu Djangon hallintapaneeliin, jota mukautettiin käytön helpottamiseksi erilaisin keinoin. Hallintapuolen etusivulle muun muassa lisättiin pikatoimintopalkki useimmin käytettyjä toimintoja varten.

Djangon oletuksena yhteyksien hallintaan taulujen välillä on pudotusvalikko, joka täytetään käytettävillä vaihtoehdoilla. Esimerkiksi kappaleen lisäys -lomakkeessa albumi-kenttään tulisi valittavaksi kaikki tietokannassa olevat albumit. Tämä on sekä epäkäytännöllistä että tietomäärän kasvaessa liian raskasta. Tämän vuoksi kehysympäristöön asennettiin kolmannen osapuolen *django-autocomplete-light*-lisäosa, jonka avulla pudotusvalikot saa vaihdettua kuviossa 29 esitetyn kaltaisiksi. Kyseisen listan sisältö haetaan JavaScriptillä Ajax-pyyntönä dynaamisesti kenttään kirjoitettaessa.

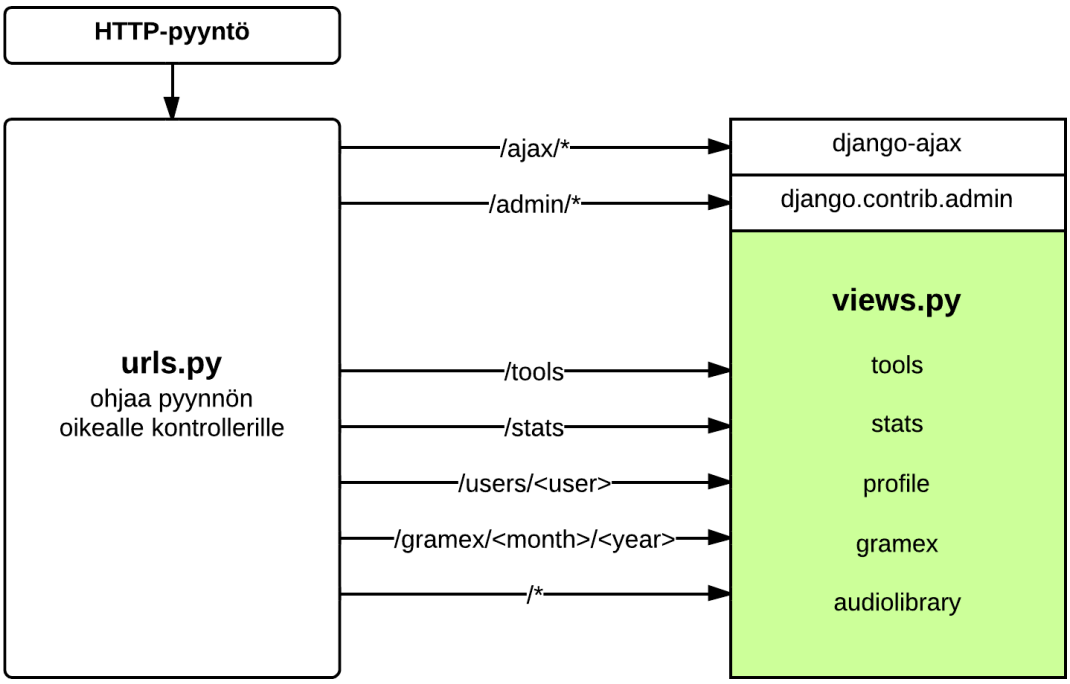


Kuvio 29. Esimerkki autocomplete-light:n HTML-vimpaimesta

Albumeiden hallintaan toteutettiin kaksi erillistä näkymää: yhden sekä usean esittäjän versiot. Usean esittäjän versio on Djangon hallintapaneelin oletuslomake, johon syötetään albumin tiedot ja jokaiseen kappaleeseen kappaleen tiedot erikseen. Tietokannassa kappaleella täytyy olla tieto esittäjästä, mutta usein albumin tietoja syöttäessä esittäjä on kaikilla kappaleilla sama. Koska saman tiedon syöttäminen monta kertaa on turhaa ja se lisää kirjoitusvirheiden mahdollisuutta. Yhden esittäjän albumin lisäämistä varten tehtiin mukautettu lomake, jossa kaikille kappaleille yhteiset tiedot kysytään vain kerran. Näkymistä on ruutukaappaukset liittessä 1.

3.4 Palvelinohjelmisto

Palvelinpuolen (engl. *server-side*) ohjelmisto on Python-kielinen sovellus, jolle Web-palvelin ohjaa HTTP-pyynnön. Pyynnön käsittelijä valitaan pyynnön URL:n perusteella kuvion 30 esittämällä tavalla.



Kuvio 30. HTTP-pyynnön käsittely

Django-ajax, kolmannen osapuolen Django-laajennus, huolehtii */ajax/*-alkuiseen ja Djangon mukana tuleva hallintapaneelimoduuli */admin/*-alkuiseen osoitteeseen tulleet pyynnöt. Muut pyynnöt ohjataan *views.py*-tiedoston funktioihin, joista on esimerkkejä kuvion 31 listauksessa.

```
1   @ensure_csrf_cookie
2   def audiolibrary(request):
3     return render(request, 'audiolibrary.html')
4
5   @login_required
6   def tools(request):
7     playlist_select_form = PlayListSelectForm(request.user)
8     data = {
9         'playlist_select_form': playlist_select_form
10    }
11    return render(request, 'tools.html', data)
12
13  @ensure_csrf_cookie
14  def gramex(request, year, month):
15    report = GramexReport.generate(int(year), int(month))
16    filename = 'gramex.{0}.{1}.txt'.format(year, month)
17    return file_dl(filename, report, None, 'windows-1252')
```

Kuvio 31. HTTP-pyynnön käsitteleviä funktioita

Riveillä 1, 5 ja 13 olevat @-merkillä alkavat rivit ovat Djangon tarjoamia dekoraattoreita. *@ensure_csrf_cookie* varmistaa, että Djangon Cross site -pyynnön väärentämiseltä (Cross Site Request Forgery, CSRF) suojaavan moduulin tarvitsema eväste lähetetään ja *@login_required* ohjaa kirjautumattomat käyttäjät kirjautumissivulle.

Funktioiden täytyy palauttaa *HttpResponse*-objekti, joka sisältää HTTP-vastauksen sisällön, HTTP-tilakoodin sekä HTTP-otsikkotiedot. Riveillä 3 ja 11 käytetty *render*-funktio kääntää mallinteen HTML-sivuksi ja palauttaa sen pohjalta vaaditun vastausobjektin. Tekijänoikeusmaksuja varten tehtävän raportin tekee *GramexReport*-luokka, ja rivillä 17 käytetty *file_dl* on itse kirjoitettu funktio, joka muodostaa ladattavaa tiedostoa tarjoavan HTTP-vastauksen. Selain näyttää WWW-sivun sijasta siis *avaa tiedosto* -valintaikkunan, jonka kautta tiedoston voi suoraan avata haluamassaan ohjelmassa tai tallentaa tietokoneelle.

3.4.1 Hallintapaneeli

Henkilökohtaisia tietoja sisältävien tietomallien muokkaamiseen hallintapaneelin kautta tehtiin kuviossa 32 esitetyn kaltainen pohjaluokka, joka on periytetty Djangon ModelAdmin-luokasta. Muut kuin pääkäyttäjät pääsevät muokkaamaan ja katselemaan ainoastaan omistamiaan asioita, kuten soittolistoja tai ohjelmia.

```
1   class PersonalModelAdmin(admin.ModelAdmin):
2     def _check_permissions(self, request, id):
3       if not self.queryset(request).filter(id=id).exists():
4         raise PermissionDenied()
5
6     # overriden methods
7     def queryset(self, request):
8       qs = super(...).queryset(request)
9       if not request.user.is_superuser:
10        qs.filter(user=request.user)
11      return qs
12
13    def get_form(self, request, obj=None, **kwargs):
14      if not request.user.is_superuser:
15        self.exclude.append('user')
16      return super(...).get_form(request, obj, **kwargs)
17
18    def save_model(self, request, model, form, change):
19      if not request.user.is_superuser:
20        model.user = request.user
21      model.save()
22
23    def change_view(self, request, objid,
24                    form_url='', extra_context=None):
25      self._check_permissions(request, object_id)
26      return super(...).change_view(request, objid,
27                                    form_url, extra_context)
```

Kuvio 32. Pääsyn rajoittaminen muiden omistamiin tietueihin

Riveillä 7–11 on esitetty, kuinka metodin *queryset* ylikirjoittamalla saadaan kaikkien Djangon suorittamien kyselyiden tuloksista suodatettua muiden käyttäjien omistamat tulokset pois, mikäli käyttäjä ei ole pääkäyttäjä. Näin ollen selausnäkymässä näkyy vain käyttäjän omat tulokset. Riviltä 13 alkava *get_form*-metodi palauttaa muokkauslomakkeen ja tässä yhteydessä rivillä 15 lisätään *exclude*-listaan käyttäjä. Tällöin Django ei näytä lomakkeella käyttäjä-kenttää ollenkaan. Tätä varten on vielä rivillä 18 ylikirjoitettu *save_model*-metodi, jota kutsutaan lomaketta tallentaessa. Mikäli lomaketta ei muokkaa pääkäyttäjä, käyttäjää ei lomakkeella kysytä ja käyttäjäksi asetetaan tässä vaiheessa lomakkeen lähettänyt käyttäjä. Rivillä 23 on esitetty ylikirjoitettu *change_view*-metodi, joka palauttaa tietomallin muokkausnäkymän.

Kaikki hallintapaneelin lomakkeet, joissa on viitaus muihin tauluihin, muokattiin käyttämään *django-autocomplete-light* -lisäosaa. Lomakkeissa, joita ei ollut syytä

erikseen toteuttaa, käytettiin *autocomplete_light.modelform_factory*-metodia, joka palauttaa automaattisesti muutetun Djangon oletuslomakkeen. Samoin kuin Djangon hallintapaneelin kohdalla myös autocomplete-light:lla käytettävät tietomallit tuli rekisteröidä ja kyselyistä tehdä kuviossa 33 kuvattu toiminnallisuus muiden kuin omien tulosten suodattamiseen.

```
1  class PersonalModelBase(acl.AutocompleteModelBase):
2    def choices_for_request(self):
3      choices = self.choices.all()
4      if not self.request.user.is_superuser:
5          choices.filter(user=self.request.user)
6      return self.order_choices(choices)
```

Kuvio 33. Automaattitäydennyslistan suodatus käyttäjän omistamiin tietueihin

Yhden esittäjän albumin syöttämistä varten tehtiin lomake, joka on periytetty albumin oletuslomakkeesta. Sille on riveillä 3, 6 ja 9 lisätty kentät artistin, maan ja tallennusvuoden merkitsemistä varten. Koska nämä tiedot ovat tietokannassa kappalekohtaisia, lomakkeen konstruktoriin on lisätty listauksessa riveillä 20–27 esitetty toiminnallisuus, jossa asetetaan näihin kenttiin alkuarvot kyseisen albumin ensimmäisestä kappaleesta.

```
1   class AlbumExForm(AlbumForm):
2     ...
3     artist = forms.ModelChoiceField(
4                queryset=Artist.objects.all(),
5                widget=ChoiceWidget('ArtistAutoComplete'))
6     country = forms.ModelChoiceField(
7                 queryset=Country.objects.all(),
8                 widget=ChoiceWidget('CountryAutoComplete'))
9     year = forms.IntegerField()
10
11    # Constructor
12    def __init__(self, *args, **kwargs):
13        super(AlbumExForm, self).__init__(*args, **kwargs)
14
15        # Capitalize labels
16        for name in ('artist', 'country', 'year'):
17            self.fields[name].label = _(name).capitalize()
18
19        # If modifying existing form, get initial data
20        if 'instance' in kwargs:
21          album = kwargs['instance']
22          tracks = Track.objects.filter(album=album.id)
23          if tracks:
24            track = tracks[0]
25            self.fields['artist'].initial = track.artist.id
26            self.fields['country'].initial = track.country
27            self.fields['year'].initial = track.year
```

Kuvio 34. Lomake yhden esittäjän albumin syöttämistä varten

Kyseisen lomakkeen käyttöön hallintapaneelissa tehtiin vielä *ModelAdmin*-luokasta toteutus, jota on avattu kuviossa 35. Riveillä 2 ja 3 on kerrottu käytettävät lomakkeet, joista *form* on päälomake albumin tietoja varten ja *inlines* on alilomakkeet. Tässä tapauksessa albumin sisältämien kappaleiden tietoja varten. Koska kappaleille yhteiset tiedot syötetään albumilomakkeessa, on kyseiset kentät piilotettu käytetystä alilomakkeesta. Tämän vuoksi lomakkeita tallennettaessa Djangon kutsuma *save_formset*-metodi on ylikirjoitettu niin, että kappaleisiin asetetaan siinä albumilomakkeesta kerätyt tiedot esittäjästä, maasta ja tallennusvuodesta.

```
1   class AlbumExAdmin(AlbumAdmin):
2     form = AlbumExForm
3     inlines = [TrackExInline]
4     ...
5
6     def save_formset(self, request, form, formset, change):
7       tracks = formset.save(commit=False)
8       data = form.cleaned_data
9       for track in tracks:
10        track.artist = data['artist']
11        track.country = data['country']
12        track.year = data['year']
13        track.save()
14      formset.save_m2m()
```

Kuvio 35. Yhden esittäjän albumin lomakkeen tallennus

Asiakas halusi, että kappaleiden kesto voidaan syöttää samassa muodossa kuin se on levyllä merkitty, jolloin puolitoista minuuttia kirjoitettaisiin *0130* tai etunollaa merkitsemättä *130*. Tämä tallennettaisiin tietokantaan 90 sekuntina. Tätä tarkoitusta varten tehtiin kenttä, jonka koodilistaus on esitetty kuviossa 36. Kenttä on periytetty Djangon *TextInput*-kentästä, joka tekee HTML-lomakkeelle tavallisen yksirivisen tekstikentän.

```
1   class DurationInput(TextInput):
2     input_type = 'number'
3
4     def render(self, name, value, attrs=None):
5       if value is None:
6           value = 0
7         else:
8           value = Utils.seconds_to_duration(value)
9         return super(...).render(name, value, attrs)
10
11    def value_from_datadict(self, data, files, name):
12        value = super(...).value_from_datadict(data,
13                                               files, name)
14        return Utils.duration_to_seconds(value)
```

Kuvio 36. Kappaleen keston syöttämiseen tarkoitettu kenttä

Rivillä 2 asetettu *input_type*-muuttuja kertoo Djangon lomakkeenvalidointijärjestelmälle, että tähän kenttään hyväksytään vain lukuarvoja. Django pyytää *render*-metodia palauttamaan HTML-lomakkeelle tulostettavan elementin antaen sille *value*

parametrina tietokannassa olevan arvon ja vastaavasti lomakkeelle syötettyä arvoa voidaan käsitellä *value_from_datadict*-metodissa ennen tietokantaan tallennusta. Riveillä 8 ja 14 kutsutaan *seconds_to_duration*- ja *duration_to_seconds*-metodeita, jotka tekevät varsinaisen muunnoksen lomakkeelle syötetystä 0130 tietokannan 90 sekunniksi ja päinvastoin.

3.4.2 Soittojen raportointi

Tekijänoikeusjärjestöille lähetettävän raportin luontiin tehtiin kuviossa 37 esitetty funktio, jolle annetaan parametreina aikaväli, jolta raportti luodaan. Rivillä 2 haetaan tietokannasta aikavälillä tehdyt soitot. Tämän jälkeen ne käydään soitto kerrallaan läpi ja kerätään rivillä 9 raportissa vaaditut tiedot *report*-hajautustauluun, josta tulostetaan asettelultaan vaaditunmukainen rivi Djangon mallinnemoottorilla rivillä 22. Silmukan käytyä läpi kaikki soitot ja kerättyä rivit listaan, rivillä 25 listan rivit yhdistetään rivinvaihtomerkeillä ja koodataan raportointiohjeessa vaadittuun *windows-1252*-merkistöön.

```
1   def generate_report(cls, start, end):
2     played_tracks = PlayedTrack.objects.filter(
3       start__gte=start, start__lt=end)
4
5     reports = []
6     for play in played_tracks:
7       proprietaries = get_proprietaries(play.track)
8       other = props.pop()
9       report = {
10        'date': play.start,
11        'track': play.track.title,
12        'proprietaries': proprietaries,
13        'artist': play.track.artist.name,
14        'albumid': get_album_identifier(play.track),
15        # other fields correspondingly
16        ...
17        'other': other
18      }
19
20      if report['end'] == '0000':
21        report['end'] = '2400'
22
23      row = render_to_string('gramexreport.html', report)
24      reports.append(row)
25
26    return '\r\n'.join(reports).encode('windows-1252')
```

Kuvio 37. Tekijänoikeusjärjestöille lähetettävän raportin luonti

Rivillä 7 kutsuttu *get_proprietaries*-funktio palauttaa kappaleen oikeudenomistajista kuusialkioisen listan, jonka viimeinen alkio sisältää kaikki loput oikeudenomistajat, mikäli heitä on enemmän kuin viisi. Loput oikeudenomistajat näytetään raportointiohjeen mukaisesti *muut tiedot*-kentässä. Funktio huolehtii myös vaaditusta tiedon muokkaamisesta niin, että mikäli oikeudenomistajaksi on merkitty *TRAD* tai *KANSANSÄVEL*, merkitään kenttään yhdysviiva. Rivillä 14 kutsuttu *get_album_identifier* poistaa levykoodista turhia merkkejä, kuten välilyöntejä ja yhdysviivoja, mikäli ne ei muuten mahtuisi raporttiin. Mikäli kappaleen soitto päättyisi keskiyöllä, tulostuisi se muodossa *0000*, joka korjataan ohjeenmukaisesti muotoon *2400* rivillä 20.

3.4.3 Hallintakomennot

Tekijänoikeusraporttien lähettämiseksi sekä tietokannan varmuuskopioimiseksi erilliselle FTP-palvelimelle toteutettiin komennot, jotka asetettiin ajastetusti ajettavaksi. Komennot toteutetaan periyttämällä ne Djangon *BaseCommand*-pohjaluokasta, ylikirjoittamalla vähintään niiden *handle*-metodi ja tallentamalla tiedosto ohjelmiston *management/commands*-hakemistoon. Komento suoritetaan antamalla Djangon hallintatyökalulle *django-admin.py* komennoksi kyseinen Python-lähdekooditiedoston nimi ilman tiedostotarkennetta.

Kuviossa 38 on esitetty *emailreport.py*-tiedostoon tallennettu luokka. Käyttöjärjestelmä suorittaa komennon *django-admin.py emailreport* ajastetusti aina kuun vaihtuessa. Koodissa ensin lasketaan riveillä 3–4 kulunut kuukausi ja luodaan riveillä 8–10 sähköpostiviesti Djangon tarjoaman *EmailMessage*-luokan avulla. Lopuksi riveillä 12–15 luodaan raportti ja lisätään se viestiin liitetiedostona. Sekä rivillä 16 lähetetään sähköposti vastaanottajalleen.

```
1   class Command(BaseCommand):
2     def handle(self, *args, **options):
3       last_month = date.today().replace(day=1)
4                      - timedelta(days=1)
5       month = last_month.month
6       year = last_month.year
7
8       subject = 'Raportti {0}/{1}'.format(month, year)
9       body = 'Raportti liitetiedostona.'
10      email = EmailMessage(subject, body, from_adr, to_adr)
11
12      filename = 'gramex.{0}.{1}.txt'.format(month, year)
13      report = generate_monthly_report(year, month)
14      mimetype = mimetypes.guess_type(...)[0]
15      email.attach(filename, report, mimetype)
16      email.send()
```

Kuvio 38. Komennon toteutus, joka lähettää tekijänoikeusjärjestöille menevän raportin

4 POHDINTA

4.1 Oppimisprosessi

Perinteiden ja hyvien tapojen mukaisesti tämänkään ohjelmistoprojektin alussa asiakkaalla ei ollut tarkkaa, pitkälle mietittyä vaatimusmäärittelyä toteutettavan ohjelmiston toiminnallisuudesta ja lopullisista ominaisuuksista. Tietotekniikkaan tottumaton henkilö ei voi tietää, mitä kaikkia ominaisuuksia ohjelmistolta voitaisiin haluta tai mitä tietotekniikalla voi ylipäänsä tehdä. Toisinaan asioista voi olla jopa vääriä käsityksiä. Ohjelmiston toimittajan näkökulmasta asiaa täytyy lähestyä kyselemällä, kuinka ohjelmisto tulisi liittymään ohjelmiston käyttäjän työrutiiniin ja minkälaisia asioita sen toivottaisiin helpottavan. Näiden seikkojen perusteella voi alkaa rakentamaan yleiskuvaa siitä, minkälainen lopputuloksen tulisi olla. Ominaisuus, joka oli ohjelmiston selkein, tärkein ja alusta asti tiedossa, oli soitettujen kappaleiden raportointi tekijänoikeusjärjestöille. Se toimikin hyvänä pohjana ohjelmiston suunnittelulle.

Kaikki tavoitteet toteutettiin onnistuneesti ja kaikki projektin aikana esiintyneet ongelmat saatiin lopulta ratkaistua. Vasta asiakkaan käytössä alkoi tulla ilmi syitä toteuttaa asioita eri tavalla alkuperäisestä suunnitelmasta. Esimerkiksi suunnitellussa levystöhaussa oli ainoastaan yksi hakukenttä, jolla haettiin sekä kappaleista, artisteista että hyllykoodeista, mutta haettujen kappaleiden löytämistä helpotti huomattavasti omat rajaavat hakukenttänsä. Toisena esimerkkinä muutettiin hakutulosten lajittelu raidan järjestysnumeron mukaan silloin, kun tuloksissa on vain yhden levyn kappaleita.

Tämä ohjelmistoprojekti kehitti vahvasti ammatillista osaamistamme. Python oli molemmille ennestään tuttu ohjelmointikieli ja Djangoakin olimme käyttäneet sen verran, että osasimme valita sen kehysympäristöksi. Ymmärrys molempien toiminnasta ja niiden ominaisuuksista syveni projektin aikana merkittävästi. Etsiessä ratkaisua käsillä oleviin ongelmiin luimme dokumentaatiota ja kirjoitimme ohjelmakoodia paljon laajemmalta alalta kuin lopullisesta ratkaisusta käy ilmi. Esimerkiksi vaihtaessa soittolistan kappaleiden järjestystä tutustuimme Djangon signaalijärjestelmään, jolla sai kytkettyä ohjelmakoodia Djangon tapahtumiin. Tässä yhteydessä aikomuksena oli suorittaa tarkistuskoodia aina ennen tietokantaan tallennusta, mikä jäi lopulta tar-

peettomaksi toteutettuamme asiakaspuolelle JavaScriptillä muutostentallennuspuskurin. Vaikka nämä asiat jäivätkin pois lopputuloksesta, oli niistä hyötyä, sillä ne ovat käyttökelpoisia ja hyödyllisiä tekniikoita useaan muuhun tilanteeseen.

Palvelun tai ohjelmiston toteuttaminen web-sovelluksena on jatkuvasti varteenotettavampi vaihtoehto muun muassa etäkäytettävyyden sekä loppukäyttäjän laite- ja käyttöjärjestelmäriippumattomuuden ansiosta. Tämän ohjelmiston vaatimuksissa etäkäytettävyys oli ehdoton ominaisuus, minkä vuoksi päädyimme kyseiseen ratkaisuun. Vaihtoehtona olisi ollut toteuttaa palvelinohjelmiston lisäksi käyttäjän laitteelle asennettava asiakasohjelma, mistä olisi heti seurannut rajoituksia sen suhteen, minkälaisesta ympäristöstä ohjelmistoa voi käyttää. Se olisi myös tuonut ohjelmiston asennuksen vaivan. Nykypäivänä internetselain löytyy lähes joka laitteesta oletuksena. Ratkaisusta oli myös se etu, että kuuntelijat saivat samalla käyttöliittymän levystön selailuun, eikä sitä tarvinnut toteuttaa erikseen.

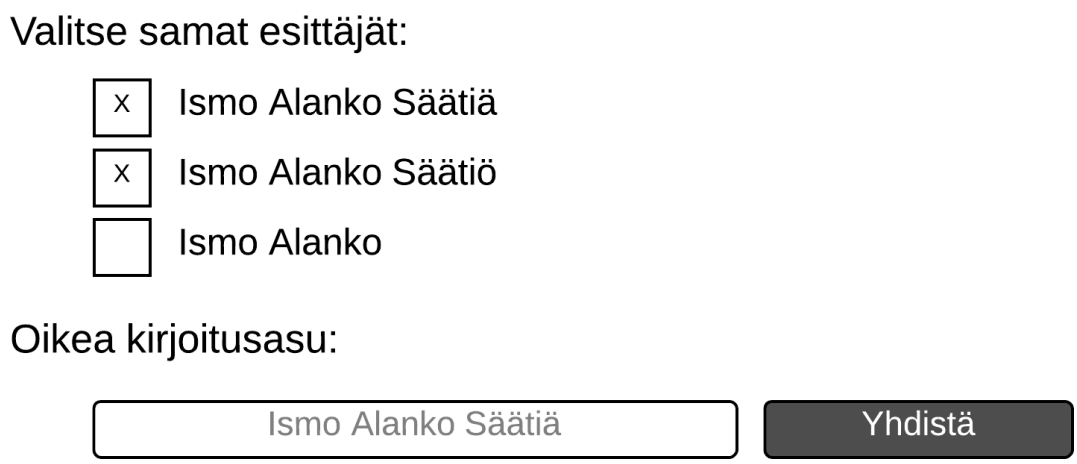
Koska HTML on vain sisällön kuvauskieli, varsinainen interaktiivinen toiminnallisuus täytyi toteuttaa muulla tavoin. Tässä tapauksessa päädyimme JavaScriptiin. Vanhastaan tällaisissa tapauksissa on sovellus toteutettu esimerkiksi Javalla, joka taas vaatii Java Runtime Environmentin asentamista koneelle ja on raskas ja kömpelö sekä käyttöliittymän ulkoasua rajoittava ratkaisu. JavaScript taas on ainoa ohjelmointikieli, jota useimmat selaimet tukevat suoraan. Nykyisin sille löytyy kehysympäristöjä ja kirjastoja, joita käyttäen on helppo toteuttaa monimutkainenkin toiminnallisuus niin, että koodin saa pysymään rakenteellisesti selkeänä ja hyvin toimivana.

4.2 Ohjelmiston tulevaisuus

Vaatimukset ja toiveet ohjelmiston ominaisuuksista tarkentuivat projektin edetessä ja ajatuksia heräsi myös ohjelmiston toimittajilta. Ohjelmisto sinänsä on täysin toimintakelpoinen ja vaatimukset täyttävä jo sellaisena kuin se toimitettaessa. Silti osa ajatelluista mahdollisista ominaisuuksista jäi niiden alhaisen prioriteetin vuoksi toteuttamatta.

Vanhan levytietokannan tuonnista kertovassa kappaleessa esitettyjen ongelmien korjaamiseksi on mahdollista toteuttaa tietojen korjaamista merkittävästi helpottavia

apuohjelmia. Esimerkiksi eri tavoilla kirjoitettujen esittäjien yhdistämiseksi yhdeksi esittäjäksi voisi toteuttaa henkilökunnan työkalut-sivulle kuviossa 39 hahmotellun työkalun, joka ryhmittelisi samankuuloiset esittäjät. Näistä valittaisiin samat, kirjoitettaisiin nimen oikea esitysmuoto ja hyväksyttäisi yhdistäminen, jolloin valitut esittäjät yhdistettäisiin tietokannassa yhdeksi.



Kuvio 39. Ulkoasuhahmotelma esittäjien yhdistämistyökalusta

Samankuuloiset esittäjiä olisi luultavasti melko triviaalia seuloa Pythonin *difflib*-kirjaston avulla, joka tutkii merkkijonoja ja palauttaa merkkijonot, jotka poikkeavat vertailuarvosta haluttuun prosenttiin saakka. Lomakkeessa voisi olla sellainenkin kenttä, johon voisi itse kirjoittaa tai listasta valita myös sellaisia artisteja, jotka eivät automaattisesti tunnistu riittävän samankaltaiseksi.

Toinen tapa parantaa tietokannan luotettavuutta olisi Wikipedian hyväksihavaitsema tapa valjastaa sivuston käyttäjät halutessaan avustamaan tietojen eheyttämisessä. Tämä olisi toteutettavissa levystön selailu- ja hakusivulle, jossa esimerkiksi pientä ikonia klikkaamalla avautuisi yksinkertainen lomake, jossa olisi kenttiä, joihin käyttäjä voisi syöttää tietoa ja lähettää henkilökunnan tarkistettavaksi, koska Wikipediasta poiketen tietokannan sisältöä ei pidä päästää ulkopuolisten suoraan muuttamaan.

Mahdollisten tietokantamuutosten tekemiseen Django-sovelluksiin on olemassa työkaluja, kuten *South*, joiden avulla tietokannassa olevan tiedon saa tuotua helposti rakenteeltaan muutettuun tietokantaan. Näin ollen sellaistakin vaativien ominaisuuksien lisääminen ohjelmaan on tulevaisuudessa mahdollista.

Asiakasohjelman toimivuus on tällä hetkellä selaimelta JavaScript-tuen vaativa. Vaikka JavaScript ja sen käyttö ohjelmistoissa on kehittynyt huomattavasti, osa ihmisistä pitää sitä silti poissa käytöstä turvallisuus- tai häiritsevyyssyistä. Soittolistojen käyttö ilman JavaScriptiä ei onnistu, joten Järviradion juontajien tulee pitää sitä päällä selaimessaan ainakin sivustokohtaisesti, mutta pelkkää levystön selaamista varten olisi tehtävissä täysin ilman JavaScriptiäkin toimiva sivu.

Ohjelmistossamme ja jo Djangossa itsessään on jonkin verran pääasiassa muuttumattomina pysyviä asetuksia, mutta joille olisi mahdollista toteuttaa muokkauslomake työkalut-sivulle. Tällaisia asetuksia on esimerkiksi sähköpostiosoite, johon kuukausittainen tekijänoikeusraportti lähetetään, ja Ajax-tietokantahakujen palautettavien tulosten maksimimäärä. Tällä hetkellä edellä mainittujen asetuksien muuttaminen vaatii muutoksia Python-lähdekooditiedostoihin eli ylläpitäjän oikeuksia ja ymmärrystä siitä, mitä ollaan tekemässä.

LÄHTEET

Alkusanat - versionhallinnasta. 2013. eKirja Git-versionhallintajärjestelmän sivustolla. Viitattu 22. 04. 2013. http://www.git-scm.com/book/fi/Alkusanat-Versionhallinnasta.

Collation Support. 2013. PostgreSQL:n online-dokumentaatio. Viitattu 05. 05. 2013. http://www.postgresql.org/docs/9.1/static/collation.html.

Django-admin.py and manage.py. 2013. Djangon online-dokumentaatio. Viitattu 07. 05. 2013. https://docs.djangoproject.com/en/1.5/ref/django-admin/.

Kansainvälinen standardoimisjärjestö ISO. 2011. Viitattu 20. 04. 2013. http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=53681.

Kaupallisten radioiden musiikin esitystietojen sähköisen raportoinnin ohje. 2009. Dokumentti Gramex ry:n sivustolla. Viitattu 15. 04. 2013. http://www.gramex.fi/fi/musiikin_kayttajat/sahkainen_media_ja_av-tuotanto/radiot, Raportointiohje.

Radioiden mediakortit. 2013. Taulukko RadioMedia ry:n sivustolla. Viitattu 15. 04. 2013. http://www.radiomedia.fi/radioasemat/mediakortit.

Summerfield, M. 2008. Programming in Python 3. Crawfordsville: Addison-Wesley Professional.

Templates. 2013. Djangon online-dokumentaatio. Viitattu 08. 05. 2013. https://docs.djangoproject.com/en/1.5/ref/templates/.

URL dispatcher. 2013. Djangon online-dokumentaatio. Viitattu 27. 04. 2013. https://docs.djangoproject.com/en/1.5/topics/http/urls/.

Vahtolammi, K. 2010. MVC-malli, peruskauraa frameworkkien käyttäjille. Viitattu 21. 04. 2013. http://vahtolam.wordpress.com/2010/09/23/mvc-malli-peruskauraa-frameworkkien-kayttajille/.

LIITTEET

Liite 1. Näkymät albumeiden syöttöön

Liite 2. Kaupallisten radioiden raportointiohje

TEOSTO/GRAMEX Sivu 1 (2)

1.7.2009

KAUPALLISTEN RADIOIDEN MUSIIKIN ESITYSTIETOJEN ILMOITTAMINEN

SÄHKÖISEN RAPORTOINNIN OHJE

Tietoja mahtuu yhteensä 460 merkkiä

1.	LÄHETYSPÄIVÄMÄÄRÄ (VVKKPP)	6 MRK.	POS	1	-	6
2.	LÄHETYKSEN ALKU (TTMM)	6 MRK.	POS.	7	-	12
3.	LÄHETYKSEN LOPPU (TTMM)	6 MRK.	POS.	13	-	18
4.	ESITYSRYHMÄTUNNUS	4 MRK.	POS.	19	-	22
5.	TEOSNIMI	50 MRK.	POS.	23	-	72
6.	APUKENTTÄ (ARVO = 00)	2 MRK.	POS.	73	-	74
7.	KESTO (MMSS)	4 MRK.	POS.	75	-	78
8.	KERRAT	4 MRK.	POS.	79	-	82
9.	OIKEUDENOMISTAJA 1	30 MRK.	POS.	83	-	112
10.	OIKEUDENOMISTAJA 2	30 MRK.	POS.	113	-	142
11.	OIKEUDENOMISTAJA 3	30 MRK.	POS.	143	-	172
12.	OIKEUDENOMISTAJA 4	30 MRK.	POS.	173	-	202
13.	OIKEUDENOMISTAJA 5	30 MRK.	POS.	203	-	232
14.	MAAKOODI	1 MRK.	POS.	233		
15.	ESITTÄJÄ	25 MRK.	POS.	234	-	258
16.	LEVYMERKKI	20 MRK.	POS.	259	-	278
17.	LEVYKOODI	15 MRK.	POS.	279	-	293
18.	SIVU	1 MRK.	POS.	294		
19.	URA	2 MRK.	POS.	295	-	296
20.	TALLENNUSMAA	3 MRK.	POS.	297	-	299
21.	LAJI	1 MRK.	POS.	300		
22.	TUNNUS- JA JINGLEMUSIIKIN TUNNUS	1 MRK.	POS.	301		
23.	OHJELMAN NIMI (GRAMEXIN TARVITSEMA TIETO)	63 MRK.	POS.	302	-	364
24.	TALLENNUSVUOSI (GRAMEXIN TARVITSEMA TIETO)	4 MRK.	POS.	365	-	368
25.	ISRC-KOODI (GRAMEXIN TARVITSEMA TIETO)	12 MRK.	POS.	369	-	380
26.	HUOMAUTUS	79 MRK.	POS.	381	-	459
27.	KÄSITTELYKOODI	1 MRK.	POS.	460		

LÄHETYKSEN ALKU: POSITIOT 11 - 12 = TYHJÄ
LÄHETYKSEN LOPPU: POSITIOT 17 - 18 = TYHJÄ

WINDOWS - 1252 MERKISTÖ

TIETUEEN LOPUSSA PITÄÄ OLLA TIETUEEN LOPPUMERKIT.
LOPPUMERKIT OVAT CR LF (ASCII-ARVOT: 13 JA 10).

TIEDOT ESITETYSTÄ MUSIIKISTA ILMOITETAAN TEOSTOON KUUKAUSITTAIN AINA SEURAAVAN KUUKAUDEN 15. PÄIVÄÄN MENNESSÄ JA GRAMEXIIN VIIMEISTÄÄN SEURAAVAN KUUKAUDEN 21. PÄIVÄÄN MENNESSÄ.

TEOSTO
Heli Aulu, puh. (09) 6810 1257, radioiden ohjelmaraportointi: radio.raportit@teosto.fi, www.teosto.fi
Lauttasaarentie 1, 00200 Helsinki

GRAMEX
Arttu Fagerlund, puh. (09) 6803 4012, radioiden ohjelmaraportointi: radiot@gramex.fi, www.gramex.fi
Pieni Roobertinkatu 16, 00120 Helsinki

1. Kirjoitetaan VVKKPP (vuosi, kuukausi, päivä). Tieto on pakollinen.

2. Kirjoitetaan TTMM (tunnit, minuutit). Esim. 0730 *). Tieto on pakollinen. Kentän positiot 11- 12 = tyhjä.

3. Katso kohta 2. Kentän positiot 17 - 18 = tyhjä.

4. Teoston antama **lähetysaseman numerotunnus**, joka on kysyttävä Teostosta. **Tieto on pakollinen**.

5. Teosnimikenttä on aakkosnumeerista tietoa. **Teoksen nimi kirjoitetaan kokonaan siten kuin se on esim. levyn päällä ilmoitettu**. Myös kaikki artikkelit kirjoitetaan. Jos teokselle on ilmoitettu useita nimiä, niin ne kirjoitetaan kenttään 26. Jos teoksen nimeä ei ole ilmoitettu, niin kirjoitetaan TEOSNIMEÄ EI ILMOITETTU.

6. Kenttään kirjoitetaan 00.

7. Kesto kirjoitetaan MMSS (minuutit, sekunnit). Esim. 0245. Tieto on pakollinen.

8. Kenttään kirjoitetaan esim. 0001. Tieto on pakollinen.

9. Tekijän nimi kirjoitetaan kokonaan, ts. myös etunimi mainitaan, mikäli se on tiedossa. Nimi kirjoitetaan järjestyksessä **SUKUNIMI ETUNIMI. Sukunimen ja etunimen väliin jätetään tyhjä väli (esim. THEODORAKIS MIKIS**), joten **pilkkua tai muuta merkkiä ei kirjoiteta suku- ja etunimen väliin eikä jälkeen**. Jos säveltäjän paikalle on merkitty esim. trad. tai kansansävel, niin tähän kenttään kirjoitetaan viiva (-), samoin kuin, jos tekijöiden nimiä ei ole ilmoitettu.

10. - 13. Kirjoitetaan annetut nimet järjestyksessä SUKUNIMI ETUNIMI, jokainen omaan kenttäänsä. Jos nimiä on enemmän kuin viisi, kirjoitetaan loput kenttään 26. Jos nimiä on vähemmän kuin viisi, jätetään loput kentät tyhjiksi.

14. **Jos säveltäjä on suomalainen, kirjoitetaan 1, muuten tyhjä.**

15. Esittäjän nimi kirjoitetaan siten kuin se on äänitteellä ilmoitettu.

16. Levymerkki esim. Warner. Kenttää voidaan käyttää myös huomautuksiin esim. WARNER CDR tai UNIVERSAL PROMO. Levymerkin ja levykoodin puuttuessa äänitteeltä kenttään merkitään esim. OMAKUSTANNE, DEMO, WAVE tai vastaava. Jos latauspalvelusta hankitulla digitaalisella musiikkitiedostolla ei ole levykoodia (tai ISRC-koodia), kentässä ilmoitetaan kauppapaikka.

17. Levykoodi = äänitteen kataloginumero esim. LRCD 117 tai 7243 8 34019 25. Äänitteen levykoodi merkitään, kuten se on äänitteellä ilmoitettu. Jos levykoodin pituus on yli 15 merkkiä, sitä tulee tiivistää jättämällä siitä pois väliviivat ja tarpeellinen määrä välilyöntejä. Mikäli äänitteessä, sen kotelossa tai liitteissä ei ole levykoodia (tai siinä tekstinä olevaa ISRC-koodia), levykoodikenttään merkitään EI ILMOITETTU. Levykooditieto on ehdottoman tärkeä äänitteiden ja niillä olevien kappaleiden tunnistamiseksi ja tieto on pakollinen.

18. Sivu/puoli. CD1=1, CD2=2, CD3=3 jne. A=1, B=2, C=3, D=4 jne., (LP, kasetti tms.).

19. Ura = kappaleen järjestysnumero äänitteellä. Esim. CD:ltä on soitettu kolmas kappale, ura = 03. Uranumero on ehdottoman tärkeä kappaleiden tunnistamiseksi ja tieto on pakollinen.

20. Tallennusmaa = kappaleen äänitysmaa. Kenttään merkitään se maa, jossa kappale on äänitetty. Kotimaisten äänitteiden ja esittäjien ollessa kyseessä merkitään kuitenkin aina FI. Maakoodeina käytetään kaksikirjaimisia ISO 3166 – standardin mukaisia maakoodeja.

21. Äänitteen laji. Kaupallisessa tarkoituksessa julkaistu äänite = 1, suora esitys = 2, demo = 3, muu = 4

22. Kenttään kirjoitetaan T, kun raportoidaan tunnus- ja jinglemusiikkia. Muuten tyhjä.

23. Ohjelman nimi

24. Tallennusvuosi = kappaleen äänitysvuosi tai vaihtoehtoisesti kappaleen julkaisuvuosi, joka käy ilmi äänitteen (P)-merkinnästä.

25. Kappaleen ISRC-koodi (International Standard Recording Code). Kenttään merkitään cd-levyltä tai muulta digitaaliselta tallenteelta (koneellisesti) poimittu kappaleen 12-merkkinen tunnus.

26. Voidaan kirjoittaa mahdollisia lisätietoja.

27. Käsittelykoodi = 1.

Huom! Teosto ja Gramex käyttävät ATK-käsittelyssä isoja kirjaimia.

*)Lähetysajat ilmoitetaan kahdella neljän merkin kentällä, joihin laitetaan numeroita. **Jos kentässä on virheellisiä merkkejä, kuten kirjaimia tai vähemmän kuin neljä numeroa, tapahtumasta tulee aina päivälähetys**.

Päivälähetys tapahtuu välillä 0600 - 2400

Esim. 0601 - 0655 ja 1200 - 1310

Yölähetys tapahtuu välillä 0000 - 0600

Esim. 0000 - 0210 ja 0510 - 0600

Esimerkkejä **virheellisistä merkinnöistä**, jotka tulostuvat päivälähetyksiksi:
- 2400 - 0100
- 2400 - 2410
- 2350 - 0400
- 0558 - 0601
- 00 - 0300
- 0120 – 0110