WEB-SOVELLUKSEN TOTEUTUS CATALYST-OHJELMISTOKEHYKSELLÄ

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulun opinnäytetyö

Tietotekniikka

HAMK Riihimäki, kevät 2016

Niklas Bergström

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Riihimäki
Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tekijä Niklas Bergström **Vuosi** 2016

Työn nimi Web-sovelluksen toteutus Catalyst-ohjelmistokehyksellä

TIIVISTELMÄ

Opinnäytetyön idea syntyi tarpeesta päivittää vanha, itse tekemäni Tullin autoveropäätösten hakuun tarkoitettu sivusto käyttäen jotain modernia web-ohjelmistokehystä. Vanha sivusto oli ohjelmoitu PHP:lla ja MySQL:aa hyödyntäen ilman ohjelmistokehystä, ja se oli raskas ylläpidettävä sekä toiminnaltaan hidas.

Tavoitteena oli rakentaa sivusto uudestaan Perl-ohjelmointikielellä ja Catalyst-ohjelmistokehyksellä. Myös tietokantamalli oli syytä uusia. Perl oli minulle entuudestaan jokseenkin tuttu kieli, mutta jouduin opettelemaan paljon uutta, sillä Catalyst ja monet sovelluksessa käytetyt CPAN-moduulit eivät olleet minulle entuudestaan tuttuja. Lisää haastetta aiheutti tietojen muuntaminen tietokantaan sopivaan muotoon.

Opinnäytetyön tarkoituksena on antaa lukijalle jonkinlainen käsitys Catalyst-ohjelmistokehyksen toiminnasta ja esitellä erinäisiä web-tekniikoita käytännössä. Tutkimusmateriaalina ja oppimisaineistona on käytetty enimmäkseen elektronista materiaalia Internetistä, mutta myös kirjallista aineistoa.

Avainsanat Catalyst, Perl, ohjelmistokehys, web-ohjelmointi, tietokanta

Sivut 30 s. + liitteet 17 s.

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Riihimäki
Degree Programme in Information Technology

Author Niklas Bergström **Year** 2016

Subject of Bachelor's thesis Implementing a web application with the Catalyst framework

ABSTRACT

The idea for this thesis arose from a need to update an old website of mine, targeted for searching for vehicle tax records out of a database. The old site was programmed in PHP without any frameworks, and it was using a MySQL database. Maintaining the application was difficult, and its performance was generally poor.

The goal was to rebuild the website using the Perl programming language and the Catalyst framework. The database schema was also due for a redesign in this project. I was familiar with Perl before, but I had to learn a lot of new things, since I was not familiar with the framework or several of the modules used in my application. An added challenge was the conversion of the data into a proper format to fit the new database schema.

The purpose of this thesis is to give the reader a basic understanding of the Catalyst framework, and to demonstrate some web techniques in action. As background materials the author mostly used electronic sources, but also literature in the field.

Keywords Catalyst, Perl, web framework, web programming, database

Pages 30 p. + appendices 17 p.

SISÄLLYS

1 SANASTOT

Lyhenne	Nimi	Selitys
AJAX	Asynchronous Javascript And XML	Yleisnimitys tekniikalle, jolla voidaan rakentaa dynaamisia web-sivustoja.
CSS	Cascade Style Sheets	Kuvauskieli tyylien määrittelemiseen web-sivuille.
CSV	Comma-Separated Values	Tiedosto, jossa sarakkeet on eroteltu jollain merkillä, tyypillisesti pilkulla.
CGI	Common Gateway Interface	Rajapinta, jolla web-sivut voivat kommunikoida palvelimella olevien ohjelmien kanssa.
HTML	HyperText Markup Language	Web-sivustoilla yleisimmin käytetty kuvauskieli.
JSON	JavaScript Object Notation	Tapa esittää dataa helposti luettavassa muodossa.
Perl	Practical Extraction and Report Language	Yleiskäyttöinen, tulkattava ohjelmointikieli.
PHP	Hypertext Preprocessor	Suosittu web-ohjelmointikieli.
SELinux	Security Enhanced Linux	NSA:n kehittämä tietoturvalaajennus Linux-ytimeen.
SQL	Structured Query Language	Kieli, joka on tarkoitettu tiedon hakemiseen tietokannoista. SQL on yleisnimitys ryhmälle useita eri tietokantaohjelmistoja- ja kieliä.
URI	Uniform Resource Identifier	Merkkijono, jolla kerrotaan tiedon sijainti.
XML	eXtended Markup Language	Kuvauskieli, missä käytetään tageja tietueiden määrittelyyn HTML-kielen tapaan. Menettänyt suosiotaan JSON:lle.

Englanti	Suomi
Action	Toiminto
Chained action	Ketjutettu toiminto
Controller	Kontrolleri, ohjain
Development server	Kehityspalvelin
Model	Malli
Environment variable	Ympäristömuuttuja
File context	Tiedostokonteksti
Production server	Tuotantopalvelin
Relationship	Relaatio, suhde
Schema	Skeema, tietokantamalli
Script	Skripti, ohjelma
View	Näkymä

2 JOHDANTO

Tein sivuston autoveropäätösten hakuun vuonna 2014. Tuolloin käytin PHP:n 5-versiota ja MySQL-tietokantaa, ja julkaisin sivuston CentOS Linux 6 -alustalle. Kun sivustoon piti myöhemmin lisätä toiminnallisuutta, se ei onnistunutkaan kovin helposti. Vaikka olinkin käyttänyt eräänlaista pseudo-MVC-suunnittelua ja erottanut koodin eri tiedostoihin, lopputulos oli kuitenkin vaikeasti ylläpidettävä ja sekava.

Suurin ongelma oli se, että näkymät ja koodi eivät olleet vahvasti eroteltuja, joten näkymistä tulikin sekamelska HTML- ja PHP-koodia. On perin selvää, että pyörää ei yleensä kannata keksiä yhä uudestaan, vaan on suotavaa käyttää valmiita työkaluja ja kirjastoja. Koska olin jo paljon käyttänyt PHP:a ennestään, ajattelin oppimiskokemuksena kokeilla jotain muuta web-tekniikkaa ja ohjelmointikieltä.

Opinnäytetyön kohdeyleisönä ovat ihmiset, jotka tahtovat ymmärtää paremmin Catalyst-ohjelmistokehyksen toimintaa, tai jotka tahtovat tutustua nykyaikaiseen www-ohjelmointiin Perl-kielellä. Jos haluaa ymmärtää yksityiskohdat sovelluksen toteutuksesta ja lähdekoodista, perustason tuntemus Perl- ja JavaScript-ohjelmointikielistä, relaatiotietokannoista, sekä HTML-kuvauskielestä on eduksi.

3 TEKNIIKAT

Tässä luvussa käydään läpi omassa sovelluksessani ja yleensäkin web-ohjelmoinnissa käytettyjä tekniikoita.

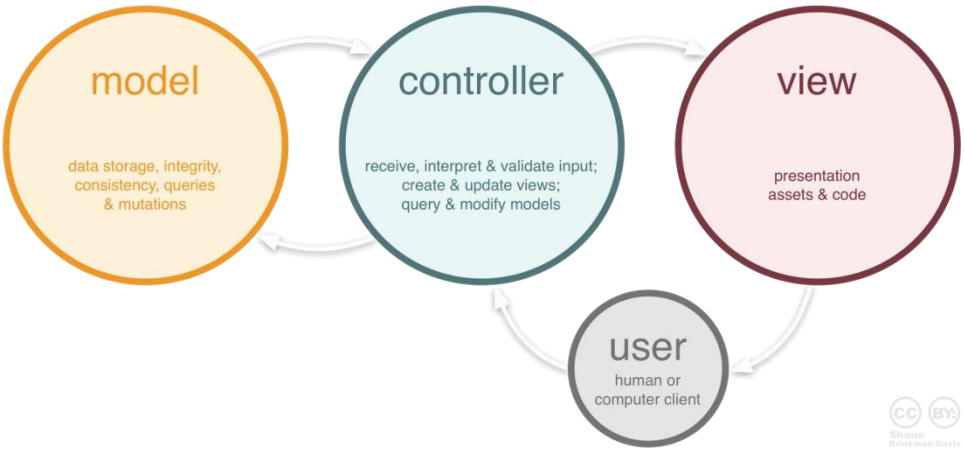
3.1 MVC-arkkitehtuuri

Model-View-Controller, eli MVC-arkkitehtuuri on ohjelmistokehityksessä käytetty kehitystyyli, jossa sovelluksen logiikka on jaettu eri osioihin kehitysprosessin ja ylläpidon helpottamiseksi (Kuva 1). Nykyään suosituimmat web-ohjelmistokehykset käyttävät MVC-arkkitehtuuria.

Nimensä mukaisesti MVC-sovellus koostuu malleista (models), näkymistä (views) ja kontrollereista (controllers). Tyypillisesti MVC:a hyödyntävä web-sovellus toimii niin, että kontrolleri käsittelee selaimen tekemän pyynnön, joka sitten pyynnön perusteella hyödyntää malleja tiedon käsittelyyn ja näkymiä tiedon näyttämiseen käyttäjälle. Kontrolleri siis sisältää tarvittavan tiedon pyynnön käsittelyyn. Mallit sisältävät objekti- ja tietokantalogiikan, ja näkymien tarkoitus on tuottaa sivu palautettavaksi selaimelle, tyypillisesti HTML-muodossa, mutta joskus myös esimerkiksi XML- tai JSON-muodossa. Monesti näkymissä käytetään jotain template-kirjastoa.

Hyvänä nyrkkisääntönä voisi pitää sitä, että kaikenlainen ohjelmointi tulisi rajoittaa malleihin ja kontrollereihin niin pitkälti kuin mahdollista. Näin näkymät saadaan pidettyä tarpeeksi yksinkertaisina, että tarpeen tullen

esimerkiksi www-suunnittelija vailla ohjelmointikokemusta voi muokata niitä.



Kuva 1. Kaavio MVC-sovelluksen toiminnasta.

MVC-kehityksessä on ainakin kaksi koulukuntaa. Eräät noudattavat ns. fat controller – thin model -tapaa, niin kuin minäkin tässä sovelluksessa. Toiset taas suosivat thin controller – fat model -tapaa. Argumentteja siitä, kumpi on parempi, löytyy puolin ja toisin, mutta molemmat ovat toimivia tapoja. Minusta tuntui luontevammalta käyttää edellä mainittua tapaa, koska useimmat löytämäni esimerkit ja dokumentaatio tekivät samoin.

3.2 AJAX

AJAX on lyhenne sanoista Asynchronous JavaScript And XML. Se on tekniikka, jolla voidaan tehdä JavaScriptin avulla asynkronisesti palvelupyyntöjä HTTP-palvelimelle ja hakea lisää tietoa tarvittaessa. Näin ollen voidaan päivittää osa sivusta dynaamisesti ilman että tarvitsee ladata koko sivua uudestaan. Palvelin ja asiakasohjelma kommunikoivat siis taustalla kun käyttäjä työskentelee sivulla. (Brinzarea-Iamandi, Darie & Hendrix 2009: 14.)

AJAX:n yhteydessä käytetään useimmiten XML-kuvauskieltä tai JSON-muotoista tietoa (Brinzarea-Iamandi, ym 2009: 71). JSON:n tapauksessa olisikin korrektimpaa puhua AJAJ-tekniikasta, mutta tämä termi ei ole saavuttanut laajaa suosiota. Toisaalta ei ole pakko käyttää kumpaakaan, AJAX-tekniikalla voidaan lisätä sivuun suoraan vaikka teksti- tai HTML-muodossa olevaa tietoa.

Käytännön esimerkkejä AJAX-tekniikasta voidaan etsiä vaikkapa Googlen etusivulta. Google tarjoaa dynaamisesti hakuehdotuksia, kun aletaan kirjoittamaan jotain hakukenttään. Siinä hyödynnetään AJAX-tekniikkaa palauttamaan selaimelle lista suosituista hakusanoista käyttäjän kirjoittaman syötteen perusteella. Prosessi tapahtuu taustalla käyttäjän huomaamatta.

3.3 Linux

Linux on suomalaisen Linus Torvaldsin vuodesta 1991 kehittämä käyttöjärjestelmäydin, eli kernel. Tästä huolimatta Linuxilla viitataan yleensä käyttöjärjestelmään eikä pelkästään ytimeen. Jotkut ihmiset käyttävätkin nimitystä GNU/Linux erottamaan käyttöjärjestelmä järjestelmäytimestä. Linux on Unix-like käyttöjärjestelmä, mutta ei aito UNIX.

Linux-käyttöjärjestelmästä on olemassa monta eri jakelua. Itse suosin Red Hat Enterprise Linuxiin pohjautuvaa CentOS-jakelua. Muita suosittuja jakeluita ovat muun muassa Ubuntu, Debian, ja Fedora.

3.4 Perl

Perl on Larry Wallin kehittämä ja ensimmäistä kertaa vuonna 1987 julkaisema ohjelmointikieli. Se on useamman ohjelmointiparadigman kieli, tukien ainakin proseduraalista-, funktionaalista-, ja olio-ohjelmointia (Perl.org a). Wall kehitti Perlin alunperin skriptikieleksi UNIX-järjestelmille, ja se saavutti nopeasti suosiota ylläpitäjien joukossa. Internetin yleistyminen kasvatti kielen suosiota entisestään, ja siitä tuli nopeasti suosituin web-ohjelmointikieli (O'Reilly & Smith).

Perlistä on nykyään kaksi eri versiota, Perl 5 ja Perl 6. Perl 5 on hyvin laajalle levinnyt, ja Linux/UNIX-ympäristöissä yleisesti käytetty ohjelmointikieli jo yli 20 vuoden ajan. Perliä on tavallisesti käytetty tekstin manipulointiin, web-sovelluksiin sekä ns. shelliskriptaukseen. Perl 6 julkaistiin vuonna 2000, mutta on vieläkin virallisesti kehityksen alla eikä ole näinollen saavuttanut suurta suosiota.

Vaikka erityisesti sellaiset skriptikielet kuten Python, PHP ja Ruby ovatkin nakertaneet Perlin kannatusta, Perl 5 ei todennäköisesti ole kuolemassa pois vielä pitkään aikaan. Siitä pitävät huolen vankka kannattajakunta, CPAN-arkisto, ja useimpien Linux- ja BSD-jakeluiden mukana tuleva Perl-komentotulkki, puhumattakaan kaikista Perl-koodinpätkistä, jotka pitävät edellä mainitut käyttöjärjestelmät toiminnassa. Perliä onkin kutsuttu nimityksellä "the duct-tape of the Internet." (Perl.org a.)

Opinnäytetyössä keskitytään Perl 5-versioon, ja aina kun puhutaan Perlistä niin viitataan nimenomaan 5-versioon, ellei erikseen mainita. Kirjoitushetkellä uusin versio on 5.22, mutta kehitysympäristöön on asennettu versio 5.16. Sinänsä tällä ei ole merkitystä, koska tässä tapauksessa tarvetta kaikista uusimmille ominaisuuksille ei ole.

3.4.1 CPAN

Yksi Perl-ohjelmointikielen vahvuuksista on CPAN-arkisto, josta löytyy suuri määrä muiden käyttäjien kehittämiä ja ladattavissa olevia Perl-moduuleita. Moduulit ovat yleensä suunniteltuja jonkun tietyn ongelman

ratkaisemiseksi, ja niitä voi etsiä osoitteesta http://search.cpan.org/. Jos esimerkiksi tarvitsee ohjelmaansa toiminnallisuutta, jolla voisi muodostaa etäyhteyden FTP-palvelimelle, voi sivustolta etsiä tarkoituksiinsa sopivan moduulin. Tässä tapauksessa löytyi sopiva moduuli Net::FTP, joka voidaan asentaa komentoriviltä käsin cpan-ohjelmalla. Jos käytössä on järjestelmänvalvojan oikeudet, voidaan moduulit asentaa järjestelmän laajuisesti, jolloin kaikilla Perl-ohjelmilla on samat moduulit käytettävissään. Jos taas ei ole tarvittavia oikeuksia, voi cpan-ohjelma myös local::lib-toimintoa hyödyntämällä asentaa moduulit käyttäjän kotihakemistoon, jolloin ne toimivat vain käyttäjän itsensä ajamissa Perl-ohjelmissa.

3.4.2 Catalyst

Catalyst on Perl-ohjelmointikielelle kehitetty, MVC-arkkitehtuuria noudattava ohjelmistokehys web-kehitykseen. Catalyst aloitti elämänsä forkkina Maypole-ohjelmistokehyksestä vuonna 2005. Catalyst tukee useita web-palvelimia ja tietokantoja. Se tukee muunmuassa CGI-, FastCGI-, sekä mod_perl-tekniikoita. Catalyst noudattaa DRY-periaatetta (Don’t Repeat Yourself), jonka ideana on se, että vältetään saman koodin kirjoittamista useampaan otteeseen esimerkiksi niin, että kaikki määritykset tehdään vain kerran. (Wikipedia 2015a.) Se on suunniteltu suurten ja keskikokoisten web-sovellusten toteuttamiseen (Perl.org b). Näin ollen lieneekin järkevämpää toteuttaa pienikokoiset projektit jollain kevyemmällä ohjelmistokehyksellä, kuten esimerkiksi Dancerilla. Catalystia jaellaan pääasiassa CPAN:n välityksellä, mutta sen voi asentaa myös monessa Linux-jakelussa pakettienhallinnasta ja FreeBSD:ssa ports-kokoelmasta.

Catalyst hyödyntää Moosea, joka on laajennus Perlin objektijärjestelmään. Sen tavoitteena on muun muassa helpottaa olio-ohjelmointia yksinkertaistamalla syntaksia ja vähentämällä koodin määrää. Lisäksi se tarjoaa uusia ominaisuuksia, kuten datan validointia. Moose on CPAN-moduuli, ja sen asennus ja käyttö tapahtuu muiden moduulien tapaan.

3.5 Muut web-ohjelmointikielet ja ohjelmistokehykset

Muita web-ohjelmoinnissa suosittuja ohjelmointikieliä ovat Perlin lisäksi Python, PHP, Ruby, ja JavaScript. Kaikki edellä mainitut ovat tulkittavia kieliä, toisin kuin esimerkiksi C, joka on käännettävä kieli. Toisin sanoen, järjestelmässä sijaitseva komentotulkki kääntää reaaliajassa koodin konekielelle, eikä näin ollen käyttäjän tarvitse itse kääntää koodia kääntäjällä konekielelle. Huono puoli tulkittavissa kielissä on, että ne ovat hitaampia kuin käännetyt kielet, koska komentotulkin pitää joka suorituskerran yhteydessä tulkita ja kääntää ohjelma uudestaan. Hyvä puoli tulkittavissa kielissä on, että ne toimivat usein järjestelmästä riippumatta; esimerkiksi Linuxilla kirjoitettu Perl-skripti toimii myös vaikkapa BSD-järjestelmissä ja Windowsissa, kunhan koodi on kirjoitettu alustariippumattomasti ja järjestelmästä löytyy Perl-komentotulkki.

TIOBEn indeksin mukaan kirjoitushetkellä Perl on tällä hetkellä 8. suosituin kieli 2,25 % markkinaosuudella. Edellä ovat PHP (6. sijoitus ja 2,77 %) ja Python (4. sijoitus ja 4,4 %), kun taas jäljessä tulevat Ruby 11. sijalla 2,05 % osuudella, ja JavaScript 2,20 % osuudella. Näemme siis että kaikki kyseiset kielet ovat vielä voimissaan, eivätkä PHP tai Perl ole äänekkäimpien Pythonin kannattajien ennustuksista huolimatta vieläkään kuolleet pois. Itseasiassa Perl on kasvattanut taas suosiotaan kun vertaillaan TIOBEn indeksin vuotta 2015 ja 2016, kasvattaen markkinaosuutta 0,86 %. (Tiobe.com) Javascriptia käytettiin pitkään lähinnä asiakaspuolen ohjelmointikielenä, jolla saatiin luotua toiminnallisuutta web-sivuille, mutta viime vuosina se on kasvattanut suosiotaan myös palvelinpuolella.

Eräs Perlin heikkous on olio-ohjelmointi; vaikkakin se tukee sitä, niin Python ja Ruby hoitavat homman paljon kivuttomammin. Tässäkin tapauksessa tosin CPAN tulee avuksi, koska siellä on tarjolla Moose. Perlin suurin vahvuus verrattuna muihin kieliin on CPAN-arkisto ja laaja tuki. Kuten sanottu, useimmista Linux- ja BSD-järjestelmistä löytyy valmiina Perl-komentotulkki. Monista järjestelmistä löytyy tosin nykyään valmiina Python-komentotulkki, mutta Ruby- ja PHP-tulkit ovatkin sitten harvemmassa, ja vaativat erillisen asennuksen. Koska Perl on ollut niin kauan olemassa, Internetistä löytyy ohjeita ja dokumentaatiota lähes joka lähtöön, mutta niin löytyy PHP:lle ja Pythonillekin.

Catalystin lisäksi muita suosittuja MVC-periaatetta noudattavia web-ohjelmointikehyksiä ovat muun muassa Django (Python), Ruby on Rails (Ruby), Laravel (PHP) ja AngularJS (JavaScript). Django on äärimmäisen suosittu, ja sille löytyy paljon dokumentaatiota ja tutoriaaleja. Tässä asiassa se on paljon edellä Catalystia, jolle on vaikeampi löytää kattavaa dokumentaatiota ja esimerkkejä. Django myös sisältää toiminnallisuutta, joka luo automaattisesti käyttöliittymän sovelluksessa käytettävän tietokannan hallintaan.

Yritysmaailmassa suosiossa olevia web-ohjelmointikieliä ovat pitkään olleet Java ja C#, joille löytyy useita ohjelmistokehyksiä. Eräs hyvin tunnettu ja suhteellisen laajalti käytetty on Microsoftin kehittämä ASP.NET MVC, joka tukee C#- ja Visual Basic.NET-kieliä.

3.6 Web-rajapinnat

Webin alkuaikoina 1990-luvulla useimmat dynaamiset web-sovellukset toimivat CGI-protokollan kautta, jolloin voitiin ajaa esimerkiksi shell-skriptejä, Perl- tai C -ohjelmia palvelimella verkkoselaimen kautta. CGI.pm-moduuli tuli osaksi Perlin ydintä vuonna 1997, ja se sisälsi useita hyödyllisiä funktioita web-ohjelmoinnin helpottamiseksi. (Diment & Trout 2009: 1.)

2000-luvulle tultaessa kävi kuitenkin ilmi, että CGI-tekniikka oli auttamatta liian hidas palvelemaan kasvaneita liikennemääriä ja nopeampia Internet-yhteyksiä. Tilalle tulikin sellaisia tekniikoita kuten

FastCGI ja mod_perl. Siinä missä CGI avaa aina uuden prosessin palvelemaan asiakasohjelman HTTP-pyyntöä, FastCGI ja mod_perl pitävät prosessin auki jatkuvasti taustalla.

mod_perl on Apachelle julkaistu liitännäinen, jolla saadaan Perl-komentotulkki integroitua palvelinohjelmistoon; se on toimintaperiaatteeltaan samankaltainen kuin PHP:lle tarkoitettu mod_php, mutta vähemmän rajoittuneempi. Tällä tavalla saadaan huomattavasti parempi suorituskyky kuin CGI:n kautta, mutta sillä on omat rajoituksensa. mod_perliä käytettäessä muutokset web-sovellukseen vaativat palvelinohjelmiston uudelleenkäynnistyksen.

PSGI on rajapinta Perlillä kirjoitettujen web-sovellusten ja web-palvelinten välillä. PSGI-sovellus on Perl-aliohjelma, joka ottaa argumentiksi ympäristömuuttujan (environment), ja palauttaa sitten viittauksen listaan, joka sisältää HTTP-statuskoodin, sisällön tyypin, ja itse sisällön. (Miyagawa)

3.7 Lighttpd

Lighttpd (puhekielessä lighty) on avoimen lähdekoodin HTTP-palvelin. Se on kirjoitettu C-kielellä, ja on sekä kevyt että nopea, vaatien vain vähän laitteistoresursseja. Valitsin lighttpd:n, koska omat käytännön kokemukseni osoittavat, että se on paljon nopeampi kuin Apache, ja koska se osaa automaattisesti käynnistää FastCGI-prosesseja, toisin kuin nginx.

3.8 MariaDB

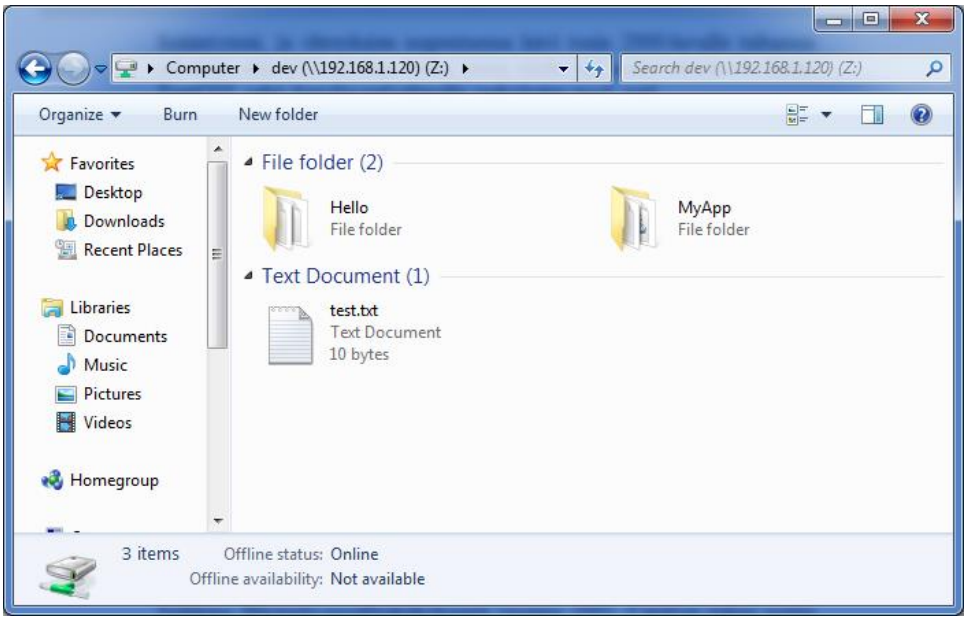
MariaDB on MySQL-tietokantaan pohjautuva relaatiotietokanta. Molemmat ovat ilmaisia ja vapaasti ladattavissa. Ne ovat pitkälti yhteensopivia keskenään. MariaDB:a käyttävät monet suuret yritykset, muun muassa Google, Red Hat, ja Mozilla (Wikipedia 2015b). Käyttämäni CentOS Linux pohjautuu Red Hat Enterprise Linuxiin, joten MariaDB on pitkälti korvannut MySQL:n pakettienhallinnassa. Tästä syystä käytän MariaDB:a MySQL:n sijasta.

Ruotsalaiset Michael Widenius ja David Axmark kehittivät avoimen lähdekoodin tietokantajärjestelmän nimeltä MySQL, ja julkaisivat sen vuonna 1995 GNU GPL-lisenssin alle. Siitä syntyi yritys nimeltä MySQL AB. Vuonna 2009 kilpaileva yritys Oracle osti MySQL:n, jolloin Widenius aloitti uuden projektin MySQL:n perustalta, jonka nimeksi antoi MariaDB. Syyksi on kerrottu muun muassa se, että Oraclen hankinnan jälkeen MySQL:n avoimen lähdekoodin tulevaisuudesta ei ole täyttä varmuutta. (Dyer 2015: xv, xix.)

3.9 Kehitysympäristö

Kehitysympäristö koostui virtuaalikoneelle asennetusta 64-bittisestä CentOS Linux 7.0 -jakelusta, jolle oli asennettu Apache, Perl 5, cpan, sekä MariaDB. Lisäksi virtuaalikoneelle oli asennettuna Samba-palvelin, jolla

saatiin hakemisto jaettua verkkolevyksi. Tavoitteena tässä oli, että kehitystyötä voisi tehdä helpommin Windowsista käsin ja tiedostot voisi tallentaa suoraan virtuaalikoneelle ilman FTP/SFTP-ohjelmaa, jolloin riittää pelkkä Windows Explorer (Kuva 2). Virtuaalikoneelle asetettiin staattinen IP-osoite, ja sitä hallitaan SSH:n kautta.

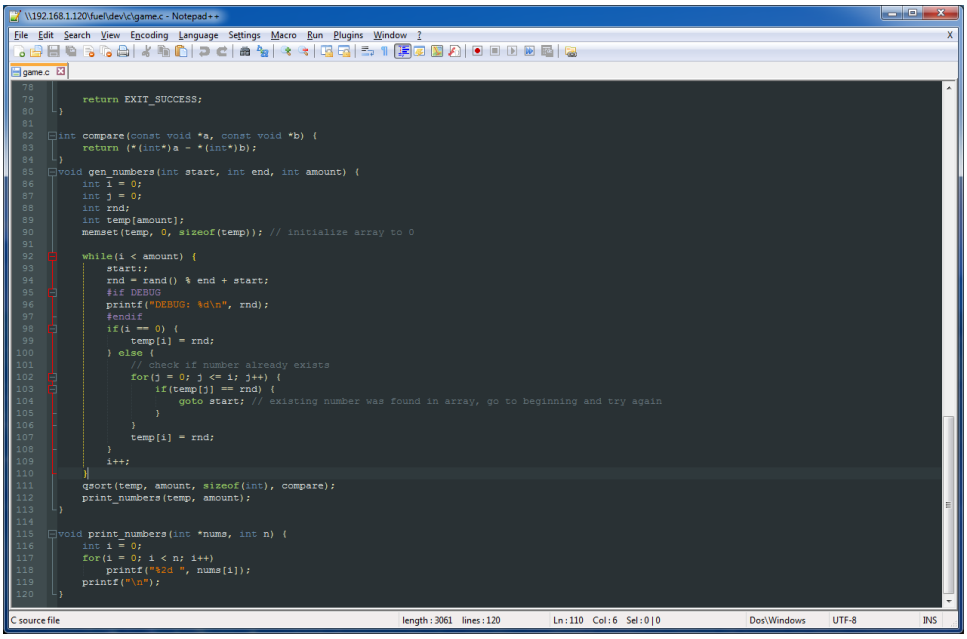


Kuva 2. Virtuaalikoneen verkkolevy Windows Explorerissa.

Myös Windowsille on saatavilla web-kehitysympäristöjä, kuten esimerkiksi XAMPP, joka on kokoelma johon kuuluu PHP, Perl, MySQL, sekä Apache-palvelin, mutta koska julkaisualustana toimii Linux-ympäristö, päätin käyttää Linuxia ohjelmiston kehitykseenkin mahdollisten ongelmien välttämiseksi. Perl tulee useimmissa Linux-distribuutioissa esiasennettuna, kun taas Windowsissa joutuu asentamaan kolmannen osapuolen ohjelmiston. Esimerkkejä näistä kolmannen osapuolen ohjelmistoista ovat ActivePerl ja Strawberry Perl.

3.9.1 Notepad++

Notepad++ on ilmainen Windows-käyttöjärjestelmille tarkoitettu tekstieditori (Kuva 3). Editori sisältää monia ohjelmoijalle hyödyllisiä ominaisuuksia, kuten koodin värityksen, ja automaattisen rivityksen sekä sulkeiden ja lainausmerkkien lisäämisen.

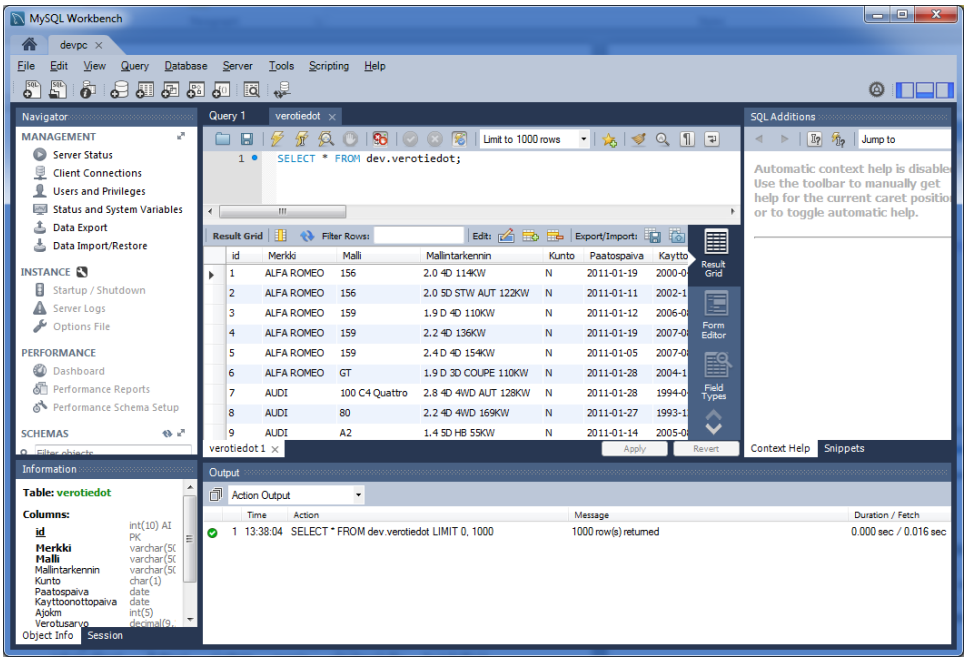


Kuva 3. Kuvakaappaus Notepad++ -tekstieditorista.

3.9.2 MySQL Workbench

MySQL Workbench on MySQL-tietokannoille tarkoitettu kehitystyökalu, joka helpottaa suuresti kehitysprosessia tarjoamalla graafisen käyttöliittymän ja useita ominaisuuksia tietokantojen ja palvelimen hallintaan. Ohjelma toimii myös MariaDB-palvelimen kanssa hyvin, vaikka 100% yhteensopivuutta ei tietääkseni luvata.

MySQL Workbenchillä voidaan yhdistää MySQL/MariaDB -palvelimelle, jos palvelin on ensin konfiguroitu oikein ja luotu käyttäjätunnus jolla on tähän oikeudet. Se korvaa tehokkaasti komentoriviohjelmat, eikä myöskään tarvita phpMyAdminin tapaista web-käyttöliittymää. Kuten alla olevasta kuvasta näkyy (Kuva 4), ohjelman kautta voidaan esim. ajaa SQL-komentoja palvelimella. Ohjelma on kätevä myös siitä, että sillä voidaan luoda paikallisesti tietokantamalleja ja kaavioita graafisen käyttöliittymän avulla, jotka sitten voidaan helposti ajaa verkon yli palvelimelle. Sillä voidaan tallentaa myös SQL-tietokantamalleja tekstimuodossa.



Kuva 4. MySQL Workbench.

3.9.3 Kehitysympäristön asennus

Internetistä löytyy valmiita virtuaalikoneen levykuvia, jotka on tarkoitettu Catalyst-kehitysympäristöksi. Tässä tapauksessa tosin lähdin liikkeelle lähes tyhjästä ja aloitin prosessin tavallisen Linux-käyttöjärjestelmän asennuksesta tyhjälle virtuaalikoneelle. Käyttöjärjestelmänä toimi CentOS Linux 7.0 64-bit. Kaikki tässä osiossa mainitut asennukset tapahtuvat pääkäyttäjänä (root). Koska kyseessä on kehitysympäristö, poistin SELinux:n käytöstä. Asensin ensin CentOS:n pakettienhallinnalla, eli yum:lla, Samban, CPAN:n, ja MariaDB:n.

Seuraavaksi konfiguroin palomuurin ja Samba-palvelimen. Sallin TCP-liikenteelle sisääntulevan portin 80 Apachelle, portin 3000 Catalystin kehityspalvelimelle, ja portin 3306 MariaDB-palvelimelle. Lisäksi sallin portit 137-138 (UDP) ja 139 (TCP), jotta Samba ja verkkolevy toimisivat.

Sitten asensin Catalystin ja kehitystyökalut. Käynnistin ja konfiguroin CPAN shellin, eli CPAN-ohjelman:

```
$ cpan
```

Jotta ohjelma olisi päivittänyt itsensä, kirjoitin sovelluksen sisäiseen komentokehotteeseen:

```
install CPAN
exit
```

Tämän jälkeen asensin cpanminus-ohjelman, joka on vaihtoehto CPAN-ohjelmalle, ja joka vaatii vähemmän konfigurointia. Asensin cpanminusin CPAN-ohjelmalla, koska näin sain uudemman version kuin mitä CentOS:n pakettienhallinnasta saisi.

```
$ cpan App::cpanminus
```

Asensin cpanminusilla alkuun välttämättömät kirjastot web-sovelluksen kehitykseen.

```
$ cpanm Catalyst::Runtime Catalyst::Devel
```

CPAN, ja cpanminus -ohjelmilla voi asentaa tai päivittää moduuleita, mutta ei kuitenkaan poistaa. Niitä voi myös asentaa pakettienhallinnan kautta. Tämä on nopeampi tapa, mutta kaikkia moduuleita ei sieltä löydy. Lisäksi CPAN:n kautta asentamalla saa yleensä uusimman version, kun taas pakettienhallinnasta voi saada vanhan version.

Huomasin, että oletuksena MariaDB:n pääkäyttäjällä ei ole salasanaa, mikä ei ole kovin suotavaa tietoturvan kannalta, mutta kehitysympäristössä asialla ei ole suurta merkitystä. MySQL Workbench ei aluksi suostunut yhdistämään palvelimelle pääkäyttäjänä. Selvisi, että eräs tapa ratkaista ongelma on luoda uusi käyttäjä asiakaskoneen IP-osoitteen kanssa, jolle annetaan pääkäyttäjän oikeudet.

```
GRANT ALL ON *.* TO root@'192.168.1.110' IDENTIFIED BY
'salasana';
FLUSH PRIVILEGES;
```

4 TIETOKANTA

4.1 Tietokannan vaatimusmäärittely

Kun suunnitellaan jotain tietokantaa käyttävää web-ohjelmistoa, lienee hyvä aloittaa tietokannan suunnittelusta, ja selvittää, mitä vaatimuksia ohjelmisto tulee asettamaan tietokannalle.

Sovelluksessa käytetään tietokantana MariaDB:a, joka pohjautuu pitkälti MySQL-tietokantaan. Valittavana on MyISAM-, ja InnoDB-tietokantamoottorit. MyISAM on vanhempi ja tukee vain taulutason lukituksia, mutta tietyissä lukuoperaatioissa marginaalisesti nopeampi, kun taas InnoDB on uudempi ja ACID-yhteensopiva, eli tukee mm. foreign key -relaatioita, transaktioita ja rivitason lukituksia. Siitäkin huolimatta, että MyISAM on tietyissä raskaissa lukuoperaatioissa marginaalisesti nopeampi ja tukee full text -indeksointia, valitsin InnoDB:n koska se on vikasietoisempi, modernimpi, ja koska sovellus tulee hyödyntämään foreign key -relaatioita taulujen välillä.

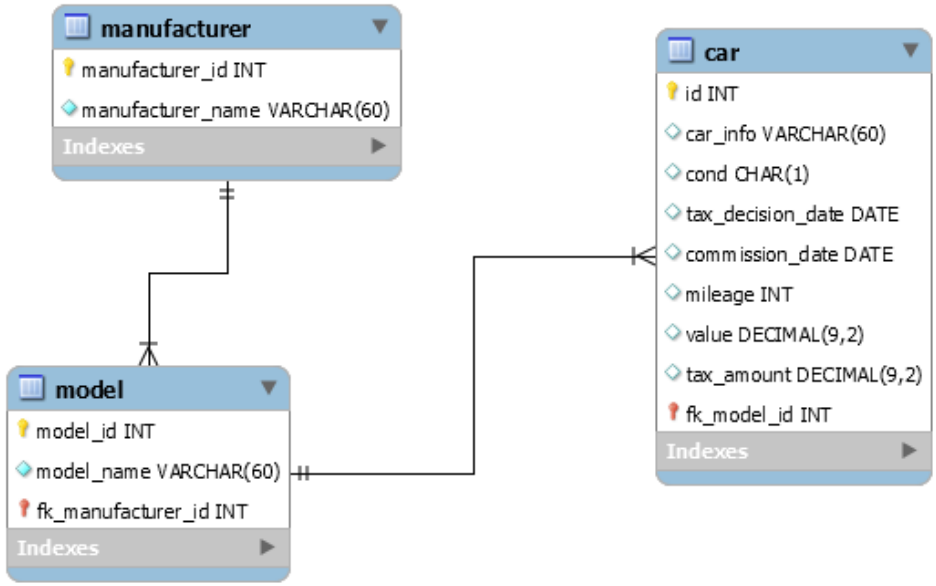
Web-sovellus tulee tekemään pelkkiä lukuoperaatioita tietokantaan, kun taas kirjoitusoperaatiot eli datan lisäys hoidetaan harvoin, enintään kerran kuukaudessa komentoriviltä Perl-skriptillä manuaalisesti tai crontabilla.

4.2 Tietokannan suunnittelu

Vanhassa PHP:lla rakennetussa sivustossa kaikki data oli yhdessä suuressa taulussa. Esimerkiksi merkki ja malli olivat tallennettuna VARCHAR-kenttiin. Tämä tarkoitti sitä, että saadakseen merkki- ja mallitiedot hakulomakkeeseen dynaamisesti, jouduttiin lukemaan koko taulu läpi *SELECT DISTINCT* -tyyppisellä SQL-lauseella. Indeksoinnista huolimatta tämä johti huonoon käytännön suorituskykyyn. Eräs tapa ratkaista tämä olisi ollut rakentaa jonkinlainen staattinen välimuisti johonkin tiedostoon,

mutta tämäkään ei ole optimaalinen ratkaisu, sillä silloin välimuisti pitäisi muistaa päivittää manuaalisesti datan lisäyksen yhteydessä.

Näin ollen oli syytä päivittää tietokanta 2010-luvulle hyödyntäen relaatiotietokannan parhaita ominaisuuksia, eli suhteita. Suunnittelin yksinkertaisen tietokantamallin, jossa on kolme taulua. Koska ohjelma käsittelee ajoneuvoja, tein taulut nimeltä **car, model, ja manufacturer** Ensimmäisessä on ajoneuvon tiedot, toisessa on mallit, ja viimeisessä merkit. Alla oleva kuva havainnollistaa taulujen rakennetta (Kuva 5).



Kuva 5. Uuden tietokannan rakenne.

Taulujen välillä on siis yksi-moneen -suhteita. Jokaisella valmistajalla (manufacturer) voi olla monta mallia (model), ja jokaiseen malliin voi kuulua monta autoa (car). Vaikka tiedot kantaan tulevatkin CSV-muodossa olevista Excel-taulukoista jotka ovat suomeksi, ja ohjelman käyttöliittymä tulee suomeksi, olen ottanut yhtenäisen linjan siinä mielessä, että ohjelman koodi ja tietokantarakenne kirjoitetaan englanniksi. Näin ei pääse sekoittumaan suomea ja englantia liiaksi, ja ohjelman ulkoasu pysyy yhtenäisenä.

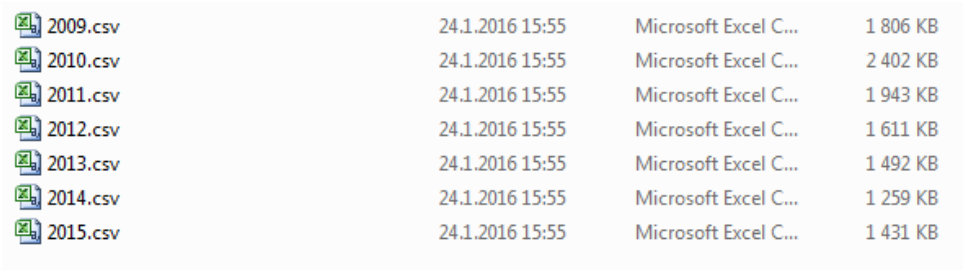
4.3 Excel-tietojen siirto tietokantaan

Jotta saataisiin tieto siirrettyä monesta erillisestä XLS-tyyppisestä Excel-tiedostosta MariaDB-tietokantaan kolmeen eri tauluun, vaaditaan vähän tietojen käsittelyä. Alla on esitelty tarkemmin vaiheet tämän saavuttamiseksi.

4.3.1 Excelistä CSV

Lähdemateriaalina toimivat Excel-tiedostot sisältävät yhden vuoden autoverotuspäätökset jaoteltuna kuukausittain eri sivuille. Ensiksi käytin

netistä löytyvää Visual Basic -makroa joka yhdisti kunkin vuoden tiedot yhdelle sivulle, jotta sitä ei tarvitsisi tehdä manuaalisesti. Tämän jälkeen piti muuttaa ne CSV-tiedostoiksi, mutta tässä tapauksessa oli hyväksi muuttaa erotinta eli delimiteriä, koska Suomessa käytetään desimaalierottimena pilkkua, kun taas useimmissa maissa ja ATK-järjestelmissä käytetään pistettä. Jotta tiedostojen automaattinen seulominen ja käsittely onnistuisi helpommin, muutin Windowsin asetuksista CSV-tiedostojen vakioerottimen pilkusta ns. pipe-merkiksi. Tämän jälkeen tallensin Excelillä tiedostot CSV-muotoon ja laitoin ne tiettyyn hakemistoon. Pikainen tarkistus notepad++:lla varmisti, että tiedot olivat nyt eroteltuja |-merkillä pilkun sijasta. Jotta välttäisin mahdollisimman paljon virheitä tiedostojen käsittelyssä myöhemmin, muunsin tiedostojen koodaustavaksi UTF-8 ja muunsin ne Windows-muodosta Unix-muotoon notepad++:lla.



Kuva 6. Manuaalisen käsittelyn seurauksena syntyneet CSV-tiedostot.

Seuraavaksi käytin Linuxista löytyvää cat-ohjelmaa yhdistämään kaikki yllä näkyvät (Kuva 6) CSV-tiedostot yhteen tiedostoon nimeltä all.csv:

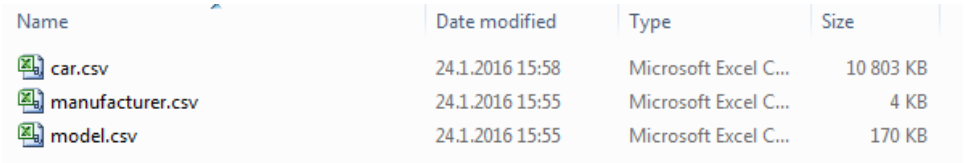
```
$ cat input/*.csv > all.csv
```

4.3.2 Ohjelmallinen erotus

Tuli aika saattaa yllämainitut tiedostot kolmeen MariaDB-tauluun. Kirjoitin Perl-skriptin nimeltä parse.pl (Liitteet 11-13), joka lukee nämä tiedostot, ja luo niiden perusteella kolme uutta CSV-tiedostoa nimiltään car, model, ja manufacturer. Tarkoitus oli näin saada mysqlimport-ohjelmalla tuotua kantaan tiedot. Ohjelma mm. lisää tiedostoihin tarvittavat tiedot foreign key -kenttiin, jotta relaatiot toimisivat suoraan. CSV-tiedostojen käsittelyyn käytin CPAN-moduulia Text::CSV. Kehitysvaiheessa huomasin outoja bugeja, muun muassa tietyt mallit osoittivat väärään merkkiin. Tutkimustyö osoitti, että ohjelmassa ei ollut tarpeeksi tarkistuksia tietojen eheyden varmistamiseksi, ja tietyissä tilanteissa tapahtui virheitä, ja kantaan ajettiin tuolloin vääriä tietoja. Tämän takia jouduin lisäämään tarkistuksia, mutta tämä hidasti ohjelman toimintaa hyvin paljon.

Nykyään ohjelma käsittelee 12 megatavun CSV-tiedoston parissa minuutissa, kun siinä ennen meni vain sekunteja. Tämä johtuu Perl-kielen heikkoudesta käsiteltäessä hyvin suuria määriä pieniä list- tai hash-tyyppisiä tietorakenteita. Jos nopeus olisi kriittinen tässä tilanteessa, pitäisi optimoida olemassaolevaa koodia, mutta koska tietojen lisäys kantaan tapahtuu harvoin, en näe tähän kiireellistä syytä. Nykyisellään ohjelma

tuntuisi toimivan, eikä virheitä tiedoissa ole näkynyt. Toinen vaihtoehto olisi ehkä hyödyntää sellaisia työkaluja kuin sed ja awk tietojen erottamiseen.



Kuva 7. parse.pl-ohjelman tuottamat, tietokantatauluja vastaavat CSV-tiedostot.

parse.pl-ohjelma tuottaa kolme CSV-tiedostoa, jotka vastaavat tietokannan tauluja. Yllä olevassa kuvassa (Kuva 7) näkyvät nämä tiedostot. Alla olevassa kuvassa (Kuva 8) näkyy alku car.csv-tiedostosta, eli tiedot jotka menevät car-tauluun. fk_model_id viittaa model-taulun id-kenttään. Taulujen suhteet on luotu tässä jo valmiiksi ennenkuin mitään on edes siirretty tietokantaan.

```
id|car_info|cond|tax_decision_date|commission_date|mileage|value|tax_amount|fk_model_id
1|1.8 T 2D COUPE 124KW|N|20090804|19930331|220|4708.00|1271.16|3794
2|2.0 4D AUT 110KW|N|20090901|20011030|143|5753.00|1760.41|1750
3|3D 1.9 JTD HB 103KW|N|20090904|20041001|136|7275.00|1746.00|1749
4|2.0 JTS 2D COUPE AUT 122KW|N|20090917|20070907|8|21704.20|6532.96|1789
5|3.0 2D CABRIOLET 141 KW|N|20090924|19980810|116|10163.00|2744.01|1811
6|3.0 2D CABRIOLET 141KW|N|20090924|19981223|103|10409.00|2810.43|1811
7|1.7 4D 77KW|N|20091008|19940314|170|540.40|125.37|1768
8|2.0 3D COUPE 122KW|H|20091009|20070613|40|19921.22|5996.28|1798
```

Kuva 8. car.csv

Tämän jälkeen tein Bash-skriptin nimeltä upgrade.sh (Liite 14), joka on suunniteltu automatisoimaan tietojen siirtoprosessi mahdollisimman pitkälle. Se oli myös kehityksessä hyvä apuväline, koska se tuhoaa sovelluksen kannalta olennaiset, olemassaolevat taulut tietokannassa, asentaa tietokantamallin tiedostosta schema.sql, ajaa sitten edellä mainitun parse.pl-ohjelman, ja lopulta ajaa mysqlimport-ohjelmalla vastikään luodut tiedot kantaan. Se myös yhdistää automaattisesti cat-ohjelmalla edellä nähdyllä tavalla input-hakemiston CSV-tiedostot tiedostoon all.csv, jonka sitten parse.pl lukee. Ohjelmat on siis suunniteltu toimimaan yhdessä.

5 CATALYST-SOVELLUS

5.1 Suunnittelu

Sovellus tulee käyttöön sivustolle, joka ei ole erityisen raskaasti kuormitettu. Oli tarkoitus, että sivuston rakenne olisi seuraavanlainen; muutama staattinen sivu sekä autoveropäätösten hakukone. Hakukoneeseen tulisi lomake, missä voisi hakea ajoneuvoja merkin, mallin, ja vuosimallin (käyttöönottopäivän) perusteella.

Suunnittelin ensin, että autoveropäätösten hakukone toimisi perinteisellä HTML-lomakkeella GET- tai POST -kyselyiden kautta, mutta kesken kehitysprosessin heitin tämän idean roskakoriin, ja päätin tehdä dynaamisen hakulomakkeen hyödyntäen AJAX-tekniikkaa. Samaan aikaan päätin lisätä sovellukseen toiminnallisuutta luomalla erillisen sivun, jossa voi selata ajoneuvoja merkin ja mallin perusteella hakemiston tyylisesti.

5.2 Yleistä

Taulukko 1. Tyypillinen Catalyst-sovelluksen hakemistorakenne.

Polku	**Kuvaus**
lib/	Sisältää sovelluksen toiminnallisuuden, eli mallit, näkymät, ja kontrollerit
root/	Staattiset tiedostot, esim. kuvat ja CSS
script/	Komentoriviltä ajettavat apuohjelmat
t/	Komentoriviltä ajettavat testiohjelmat
Makefile.PL	Listaus sovelluksen tarvitsemista moduuleista

Makefile.PL on tiedosto, johon voidaan lisätä sovelluksessa käytettyjä CPAN-moduuleita. Näin varmistetaan, että kun siirretään sovellus tuotantopalvelimelle ja ajetaan kyseinen ohjelma, niin mitään moduuleita ei jää puuttumaan, vaan ohjelma valittaa moduulien puuttumisesta. Kyseinen ohjelma luo myös MYMETA-tiedostot, joita voidaan hyödyntää, jos tahdotaan esimerkiksi cpanminus-ohjelmalla paketoida kaikki tarvittavat moduulit Catalyst-sovelluksen mukaan, jolloin moduulit ovat tuotantoympäristössäkin sovelluskohtaisia, eikä tarvitse asentaa niitä järjestelmän laajuisesti.

lib-hakemistosta löytyy sovelluksen .pm-tiedosto, sekä alihakemistot Controllers, Models, ja Views. Catalyst-sovellus on käytännössä paketti (Perl package), jonka aloituspiste kyseinen tiedosto on. Täällä voidaan esimerkiksi muuttaa sovelluksen asetuksia tai ottaa käyttöön Catalyst-liitännäisiä.

script-hakemistosta löytyvät seuraavat ajettavat apuohjelmat: *cgi, create, fastcgi, server*, ja *test*. Kehitysprosessissa useimmin käytettyjä ovat *create* ja *server*. Nämä ohjelmat saavat tiedostonimeensä etuliitteen Catalyst-

projektin nimen perusteella. *test*-ohjelmallekin lienee oma käyttötarkoituksensa, mutta suoritin itse testaamisen *prove*-ohjelmalla, josta myöhemmin lisää.

server.pl on kehityspalvelin (development server), joka kuuntelee porttia 3000. Kun annetaan sille –r parametri, niin palvelin osaa käynnistää itsensä uudelleen aina kun tiedostoissa näkyy muutoksia. Näin ei tarvitse manuaalisesti aina käynnistää sitä uudelleen, mikä nopeuttaa kehitysprosessia.

create.pl-skriptillä voidaan luoda näkymiä, malleja, ja kontrollereita. cgi.pl ja fastcgi.pl ovat tuotantoon tarkoitettuja ohjelmia, joilla saadaan Catalyst-sovellus toimimaan CGI/FastCGI-rajapintoja tukevan HTTP-palvelimen kanssa.

5.3 Toteutus

Tässä osiossa tutkitaan kehittämääni Catalyst-sovellusta nimeltään Autoulkomailta, joka tuli käyttöön samannimiselle sivustolle. Tämä tulee tapahtumaan askel askeleelta, jotta lukijalle muodostuisi jonkinlainen kuva siitä, miten sovellus syntyi. Erinäiset kokeiluvaiheet ja virhearviot on jätetty tästä pois, ja vain olennaiset askeleet on kuvattu. Tämä versio on ensimmäinen julkaisukelpoinen versio, sovelluksen kehitys tullee jatkumaan myöhemmin ja lähdekoodi muuttumaan edelleen. Koodirivit, jotka alkavat dollarinmerkillä ($) kertovat, että komento on kirjoitettu Linuxin komentoriville. Aloitin projektin nimeltä Autoulkomailta ja vaihdoin kyseiseen hakemistoon:

```
$ catalyst.pl Autoulkomailta
$ cd Autoulkomailta
```

Koska tietokanta on UTF-8-muodossa, on suotavaa laittaa kaikki muukin samaan muotoon. Lisäsin autoulkomailta.conf-tiedostoon rivin:

```
encoding utf8
```

Seuraavaksi muokkasin lib/Autoulkomailta.pm-tiedostoa alla olevan kuvan (Kuva 9) mukaisesti, sekä lisäämällä alkuun:

```
use utf8;
```

```
__PACKAGE__->config(
    name => 'Autoulkomailta',
    encoding => 'UTF-8',
    default_view => "HTML",
    # Disable deprecated behavior needed by old applications
    disable_component_resolution_regex_fallback => 1,
    enable_catalyst_header => 1, # Send X-Catalyst header
);
```

Kuva 9. Autoulkomailta.pm

Tässä vaiheessa oli hyvä tehdä näkymä, sillä määrittelin aikaisemmin default_view:n eli oletusnäkymän. Käytin sovelluksessa Template::Toolkit-kirjastoa, jota Catalyst tukee suoraan.

```
$ script/autoulkomailta_create.pl view HTML TT
```

Tällä komennolla saadaan luotua HTML-niminen näkymä, joka on tyyppiä TT, eli Template::Toolkit. Uusi näkymä löytyy polusta lib/Autoulkomailta/View/HTML.pm. Avasin tiedoston ja tein sinne muutoksia. Lisäsin asetuksiin wrapper-direktiivin, jotta kaikki templatet sisällyttäisivät page.tt-nimisen tiedoston automaattisesti. Määrittelin tässäkin enkoodaukseksi UTF-8. (Kuva 10)

```
__PACKAGE__->config(
    TEMPLATE_EXTENSION => '.tt',
    WRAPPER => "page.tt",
    ENCODING => "utf8",
    render_die => 1,
);
```

Kuva 10. HTML.pm

Koska päätin lisätä sovellukseen AJAX-toiminnallisuutta, tarvitsin näkymän joka osaa tuottaa tietoa JSON-muodossa. Tähän löytyi avuksi Catalyst::View::JSON-moduuli, jolla sain tehtyä JSON-nimisen näkymän, joka on tyyppiä JSON:

```
$ script/autoulkomailta_create.pl view JSON JSON
```

Oletuksena sovelluksessa on yksi kontrolleri, joka löytyy polusta lib/Autoulkomailta/Controllers/Root.pm. Tänne määrittelin etusivun sekä muut staattiset sivut. (Kuva 11)

```
sub index :Path :Args(0) {
    my ( $self, $c ) = @_;
}

sub verotus :Local { }

sub tuonti :Local { }

sub tietoa :Local { }

=head2 default

sub default :Path {
    my ( $self, $c ) = @_;
    $c->response->status(404);
    $c->stash(template => "error.tt");
}
```

Kuva 11. Root.pm

Staattisten sivujen lisääminen on tässä tapauksessa helppoa ja vaivatonta. Kontrollereissa sivujen ja URL-osoitteiden määrittely tapahtuu luomalla toimintoja (action). Toiminnot ovat käytännössä Perl-aliohjelmia (subroutine), eli funktioita. Antamalla toiminnoille Local-määrittelyn, sovellus osaa luoda niille automaattisesti URL-osoitteen toiminnon ja kontrollerin nimen perusteella. Otetaan tästä esimerkiksi tietoa-niminen toiminto, johon nyt osoittaa URL-osoite /tietoa, ja jonka templatena toimii

tietoa.tt, ilman sen kummempia määrittelyjä. default-toiminnossa nähdään, miten voidaan asettaa kuhunkin toimintoon oma vapaavalintainen template muokkaamalla Catalystin stashia. Stash on hash-tyyppinen rakenne, jossa voidaan edellä mainitun esimerkin mukaisesti vaikka vaihtaa templatea tai lähettää templatelle tietoa. Templateja etsitään oletuksena root-hakemistosta, mutta se on vaihdettavissa jos tahdotaan. Itse en nähnyt tähän syytä, vaan tein tässä vaiheessa templatet kaikille edellä määritellyille sivuille root-hakemistoon.

Ei ole järkevää laittaa kaikkea toiminnallisuutta Root-kontrolleriin, joten tein erilliset Search- ja Ajax-kontrollerit:

```
$ script/autoulkomailta_create.pl controller Search
$ script/autoulkomailta_create.pl controller Ajax
```

Search-kontrolleriin tulisi kaikki käyttäjälle näkyvä hakukonetoiminnallisuus, mutta Ajax-kontrolleriin tulisi kaikki sellainen, mistä käyttäjän selaimessa ajettavat Javascript-komentosarjat hakisivat tietonsa, jotta saataisiin AJAX-tekniikalla tehtyä nykyaikainen ja dynaaminen web-sovellus.

Kun näkymät oli luotu ja root-kontrolleria muokattu, oli aika tehdä muutama template. Tein root-hakemistoon wrapperin nimeltä page.tt sekä aikaisemmin määriteltyjen sivujen mukaisesti tiedostot index.tt, verotus.tt, tuonti.tt, tietoa.tt, ja error.tt. Wrapperi, eli page.tt on tässä toteutettu niin, että ellei stashiin määritetä no_wrapper-muuttujan arvoksi 1, se sisältyy automaattisesti. Jos avataan selaimessa URI /verotus, Catalyst renderöi ensiksi page.tt:n jonka sisälle sitten mahdutetaan verotus.tt. page.tt:ssä on ehtolausekkeita joiden perusteella no_wrapper toimii, sekä [% content %] -tagi, johon templaten sisältö tulee.

Mallien teon aloitin luomalla tietokantamallit. Ne ovatkin ainoat mallit mitä tässä sovelluksessa tarvitaan.

```
$ script/autoulkomailta_create.pl model DB DBIC::Schema
Autoulkomailta::Schema create=static dbi:mysql:mydb root
```

Kyseinen komento luo automaattisesti tietokantayhteyttä ja tietokantaobjekteja kuvaavat mallit:

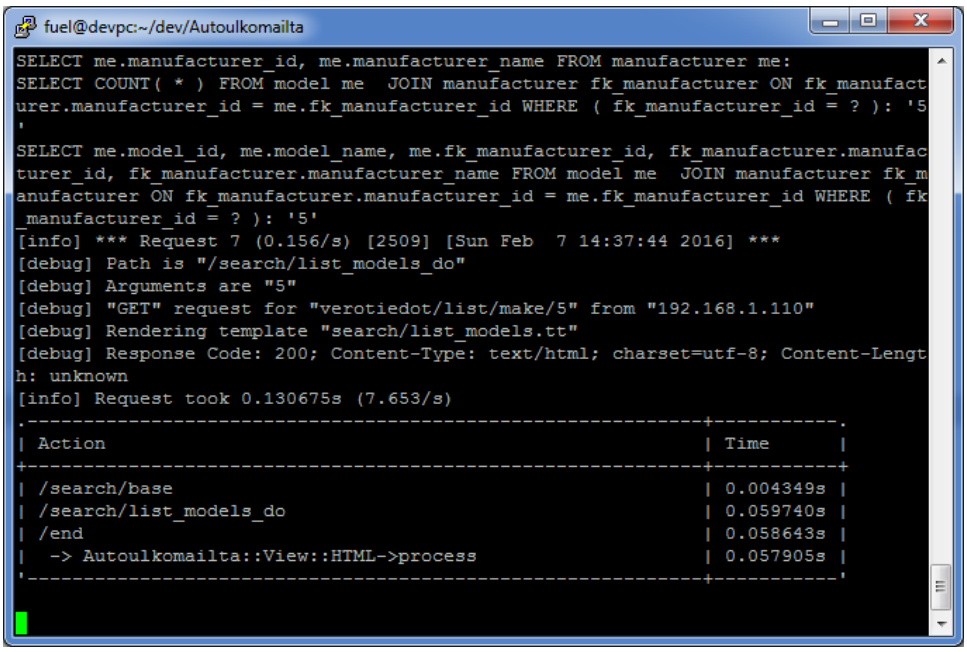
```
lib/Autoulkomailta/Models/DB.pm
lib/Autoulkomailta/Schema/Result/Car.pm
lib/Autoulkomailta/Schema/Result/Manufacturer.pm
lib/Autoulkomailta/Schema/Result/Model.pm
```

Tässä on luotu DBIx::Class::Schema-tietokantamalli tietokannasta, joten sovelluksessa voidaan hyödyntää DBIx::Class::ResultSet-moduulia tietojen etsimiseen tietokannasta. Jokainen taulu saa oman PM-tiedostonsa, jonka sisällä on määritelty sarakkeet ja suhteet. Hyödyntämällä DBIx::Class-moduulien ominaisuuksia sovelluksessa ei tarvitse manuaalisesti kirjoittaa SQL-lausekkeita. Tässä vaiheessa, kun oli tarkoitus alkaa kehittämään tietokantaan liittyvää toiminnallisuutta, voidaan asettaa ympäristömuuttuja, joka laittaa SQL-kyselyiden osalta debug-tilan päälle.

```
$ DBIX_CLASS_STORAGE_DBI_DEBUG=1
script/autoulkomailta_server.pl -r
```

Tällä komennolla saadaan kehityspalvelin näyttämään mahdolliset SQL-kyselyt joka sivulatauksella (Kuva 12). Tällä tavalla asetettuna se

vaikuttaa vain kyseiseen prosessiin. Jos kyseinen muuttuja asetettaisi vaikkapa käyttäjän .bashrc-tiedostoon (vain Bash-ympäristössä), kehityspalvelimessa olisi aina ajettaessa tuo SQL debug-tila päällä.



Kuva 12. Kehityspalvelin SQL debug -tilassa. Huomaa SQL-lausekkeet.

Tietokantamallit oli nyt määritelty ja kehityspalvelin sopivassa tilassa joten tässä vaiheessa aloin kirjoittamaan Ajax-kontrolleriin toiminnallisuutta. Ensin määrittelin omia muuttujatyyppejä käyttämällä hyväksi MooseX::MethodAttributes- ja MooseX::Types-moduuleita. Lisäsin alkuun omat PosInt- (Positive Integer) ja YearInt-muuttujamäärittelyt, jotta saisin parannettua sovelluksen tietoturvallisuutta, ja vähennettyä tarkistuksien määrää toiminnoissa. Lisäksi on otettava pois käytöstä namespace::autoclean, muuten nämä itse määritellyt tyypit eivät toimi. Otin myös käyttöön MooseX::Types::Moose-moduulista valmiiksimääritellyt Int, ja Str-tyypit (Kuva 13).

```
#use namespace::autoclean;
use MooseX::MethodAttributes;
use MooseX::Types -declare => [
    qw(PosInt YearInt)
];
use MooseX::Types::Moose qw(Int Str);

BEGIN { extends 'Catalyst::Controller'; }

# create custom subtypes to reduce the amount of
# checks we have to write in the subroutines

subtype PosInt,
 as Int,
 where { $_ > 0 };

subtype YearInt,
 as Int,
 where { $_ >= 1899 && $_ <= 2021 };
```

Kuva 13. Ajax.pm, määrittelyt kahdelle omalle muuttujatyypille.

Näitä omia muuttujatyyppejä voidaan hyödyntää samalla tavalla kuin Moosen sisäänrakennettuja tyyppejä, joita ovat muun muassa Str (string) ja Int (Integer). Näillä voidaan rajoittaa argumenttien tyyppiä, mitä toiminnot kontrollerissa ottavat vastaan. Jos edellä määritellylle PosInt-tyyppisen argumentin haluavalle toiminnolle annetaan tavallinen Str-tyypin argumentti tai alle 0:n oleva Int-argumentti, niin Catalyst ei edes löydä toimintoa ja palauttaa käyttäjälle 404 Not Found -tyypin virheilmoituksen. :Args-toiminnolla voidaan määrittää montako argumenttia toiminto ottaa vastaan. Args(2) ottaisi vastaan 2 mitä tahansa argumenttia, kun taas Args(Int,Int,Int) ottaisi vastaan 3 Int-tyypin argumenttia.

Ajax.pm-kontrollerissa käytin chained-tyyppisiä toimintoja, jotka ovat nimensä mukaisesti ketjutettuja. Alla olevassa kuvassa (Kuva 14) on yksinkertainen ketjutettu esimerkki, jossa käytetään lisäksi Moosen Int-tyypin muuttujaa rajoituksena. Lisäksi siinä näytetään, miten tietokannasta voidaan hakea tietoa. get_models-toiminto hakee listan malleista valmistajan id-numeron perusteella ja tallentaa kannasta haetut tiedot model_data-muuttujaan stashin sisälle, ja kutsuu sitten JSON-näkymää. base-toiminnon sisällä määritellyt stash-muuttujat periytyvät kaikille jotka käyttävät sitä ketjun alkuna.

```
sub base :Chained("/") :PathPart("ajax") :CaptureArgs(0) {
    my ($self, $c) = @_;

    $c->stash->{schema} = $c->model("DB")->schema;
    $c->stash(no_wrapper => 1); # disable wrapper
}

# returns list of models by given manufacturer ID in JSON format
sub get_models :Chained("base") :PathPart("get_models") :Args(Int) {
    my ( $self, $c, $id ) = @_;

    my $rs = [$c->stash->{schema}->resultset("Model")->search({fk_manufacturer_id => $id})];
    my @list;

    foreach(@$rs) {
        push @list, { model_id => $_->model_id, model_name => $_->model_name };
    }
    $c->stash(model_data => \@list);

    delete $c->stash->{schema};
    $c->forward("View::JSON");
}
```

Kuva 14. Ajax.pm, JSON-tietoa palauttava toiminto.

Toiminto do_search Ajax.pm-tiedostossa käyttää rajoituksia laajemmin, mukaanlukien itsemääriteltyjä tyyppejä (Liitteet 3-4). Tällä tavalla saadaan parannettua tietoturvaa ja helpotettua testaamista. Lisäksi ei tarvita suuria määriä tietojen tyyppi- ja eheysvarmistuksia toiminnon sisälle. Tein ajax/do_search.tt-templaten (Liite 9) kyseistä toimintoa varten. do_search ei tuota JSON-muodossa tietoa, vaan tuottaa tiedot HTML-muodossa taulukkoon, jonka hakukone voi hakea suoraan XMLHttpRequest:lla ja lisätä HTML:n joukkoon. Templatelle lähetetään Data::Page-tyyppinen objekti, jonka avulla voidaan luoda sivutustoiminnallisuus.

Kun AJAX-toiminnallisuus oli saatu valmiiksi ja testattua pikaisesti selaimella, oli aika siirtyä itse hakukoneeseen. Ensiksi tein root/search-hakemistoon templaten search_ajax.tt (Liite 10). Sitten lisäsin Search-kontrolleriin search-nimisen toiminnon, joka on varsin yksinkertainen. Siinä määritellään vain template, sekä muuttujat min_year ja max_year. Näitä muuttujia käytetään alasvetovalikoiden automaattiseen luontiin search_ajax.tt-templaten sisällä alla olevan esimerkin (Kuva 15) mukaisesti. Search-kontrollerin (Liitteet 5-6) sisällä search-toiminnossa laitetaan stashiin pari muuttujaa, joihin päästään käsiksi templaten sisältä. Näin helpotetaan ohjelman muuttamista, jos joskus halutaan vaihtaa kyseisiä vuosilukuja (Kuva 16).

```
<label for="year_start">Vuosimalli alkaen:</label>
<select name="year_start" id="year_start" onchange="check_year()">
<option value="[% min_year - 1 %]" selected="selected">Valitse</option>
[% year = min_year -%]
[% WHILE year <= max_year -%]
<option value="[% year %]">[% year %]</option>
[% year = year + 1 -%]
[% END -%]
</select>
```

Kuva 15. search_ajax.tt: Alasvetovalikon luonti Template::Toolkitia hyödyntämällä.

```
sub search :Chained("base") :PathPart("search") :Args(0) {
    my ( $self, $c ) = @_;

    $c->stash->{template} = "search/search_ajax.tt";
    $c->stash(max_year => 2020, min_year => 1900);
}
```

Kuva 16. Search.pm: search-toiminto.

Kuva 17. Hakulomake ja hakutuloksia.

Hakulomake (Kuva 17) tarvitsee JavaScriptin toimiakseen. Kirjoitin JavaScript-koodin tiedostoon scripts.js (Liitteet 7-8), jossa on kaikki tarvittavat funktiot lomakkeen toimintaan ja tietojen hakuun liittyen. Nykyään sellaiset ulkoiset kirjastot kuten jQuery ovat äärimmäisen suosittuja, mutta tässä tapauksessa toiminnallisuus oli helppoa toteuttaa ilmankin tällaisia kirjastoja. JavaScript-koodi hyödyntää sisäänrakennettua XMLHttpRequest-toimintoa tietojen hakemiseen. Kun käyttäjä painaa lomakkeen lähetysnappia, lomakkeesta otetaan tarvittavat tiedot, ja tietojen perusteella rakennetaan ja lähetetään HTTP-kysely Ajax-kontrollerin do_search-toiminnolle. Se taas palauttaa HTML-muodossa olevan taulukon, joka lisätään sellaisenaan lomakkeen alapuolelle. Näin saadaan aikaiseksi dynaaminen hakukone suhteellisen pienellä vaivalla.

Toiminnallisuus verotietojen listaukseen tuli myös samaan Search-kontrolleriin; näin saadaan hyödynnettyä ketjutettuja toimintoja, koska sekä hakulomake että listaus tulevat saman URI:n alle polkuun /verotiedot/. Toimintoja tämän toiminnallisuuden luomiseksi on useita: list, list_models, list_models_do, list_cars, list_cars_by_page, list_cars_page. Syy runsaalle määrälle on se, että näin saadaan käsiteltyä tehokkaasti kaikki argumentit. Kaikki toiminnot Search-kontrollerissa periytyvät base-nimisestä toiminnosta (Kuva 18), joka toimii ketjun lähtökohtana.

```
sub base :Chained("/") :PathPart("verotiedot") :CaptureArgs(0) {
    my ($self, $c) = @_;
    my $schema = $c->model("DB")->schema;
    my $rs = $schema->resultset("Manufacturer")->search({});
    $c->stash(schema => $schema,
              template => "search/index.tt",
              manufacturer_list => [$rs->all],
              );
}
```

Kuva 18. Search.pm: base-toiminto.

Täälläkin hyödynnetään itse määriteltyjä MooseX::Types-muuttujatyyppejä samalla tavalla kuin edellä mainituissa Ajax-kontrollerin esimerkeissä. On huomioitava, että vaikka molemmissa kontrollereissa käytetään PosInt-nimistä itse määriteltyä muuttujatyyppiä, Ajax-kontrollerissa se hyväksyy vain kokonaisluvun joka on 1 tai suurempi, kun taas Search-kontrollerissa se hyväksyy kokonaisluvun joka on 0 tai suurempi. Olisi kenties viisasta nimetä nämä jotenkin eri tavalla, mutta se jääköön tehtäväksi seuraavassa versiossa.

Template-tiedostot listaustoimintoihin ovat hyvin yksinkertaisia (Liitteet 15-17). Listaustoimintoja ovat list, joka näyttää listan kaikista merkeistä; list_models, joka näyttää kaikki kyseiseen merkkiin kuuluvat mallit (Kuva 19); sekä list_cars, joka näyttää kaikki kyseiseen malliin kuuluvat ajoneuvot (Kuvat 20-22). Niihin kaikkiin saadaan tiedot kontrollerissa olevien toimintojen kautta laittamalla stashiin tietokantakyselystä saatu DBIx::Class::ResultSet-tyyppinen tietorakenne, joka sitten käydään templatessa läpi Template::Toolkitin FOREACH-silmukkaa käyttämällä.

```
# list all models of given manufacturer id
sub list_models :Chained("base") :PathPart("list/make") :Args(0) {
    my ( $self, $c ) = @_;

    $c->response->body("No arguments given.");
}
sub list_models_do :Chained("base") :PathPart("list/make") :Args(PosInt) {
    my ( $self, $c, $m_id ) = @_;

    $c->stash->{template} = "search/list_models.tt";
    my $rs = $c->stash->{schema}->resultset("Model")->search({ fk_manufacturer_id => $m_id },
                                                             { prefetch => "fk_manufacturer" });
    return unless $rs->count;
    $c->stash->{model_list} = [$rs->all];
}
```

Kuva 19. Search.pm: toiminnot mallien listaamiseksi merkin perusteella.

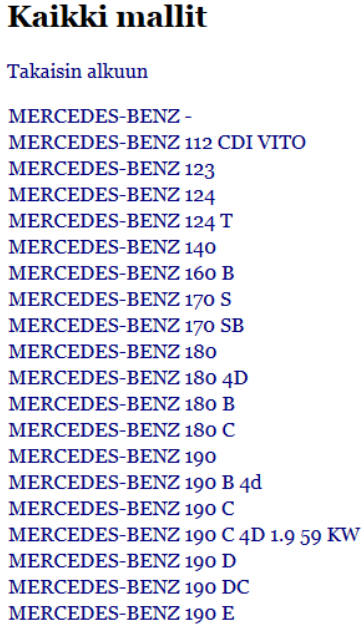
Yllä olevassa kuvassa näkyy, kuinka toteutin esimerkiksi mallien listauksen ajoneuvon merkin id-numeron perusteella. list_models-toiminnon tarkoitus on vain kertoa käyttäjälle, että argumentti puuttuu URL-osoitteesta. Tämä ei olisi ollut välttämätöntä, sillä olisinhan voinut vaan antaa 404-sivun käyttäjälle nähtäväksi. list_models_do-toiminnossa tapahtuu kaikki merkittävä. Ensiksi asetetaan oikea template, eli search/list_models.tt. Sitten käytetään base-toiminnossa stashiin tallennettua tietokantaskeemaa hyväksi, ja etsitään annetulla valmistajan id-numerolla. Prefetchiä käyttämällä luodaan SQL JOIN -tyyppisiä kyselyitä, eli sillä voidaan siis yhdistellä tietoja eri tauluista. Tämä kyseinen koodinpätkä hakee kaikki mallit, joiden fk_manufacturer_id on sama kuin annettu kokonaisluku, ja suorittaa sitten JOIN-operaation

manufacturer-taulua vastaan. Käytännössä operaatio tuottaa seuraavanlaista SQL-koodia jos haetaan id-numerolla 1:

```
SELECT
me.model_id,
me.model_name,
me.fk_manufacturer_id,
fk_manufacturer.manufacturer_id,
fk_manufacturer.manufacturer_name
FROM model me
JOIN manufacturer fk_manufacturer
ON fk_manufacturer.manufacturer_id = me.fk_manufacturer_id
WHERE ( fk_manufacturer_id = ? ): '1'
```

Huomioi, että SQL-kyselyissä käytetään automaattisesti ns. prepared statement -lausekkeita, jotka eivät ole yhtä alttiita SQL-injektio -tyyppisille hyökkäyksille kuin perinteiset kyselyt. Jos ei löytynyt tuloksia, eli jos $rs->count palauttaa 0 niin lopetetaan, muuten asetetaan tulos model_list-muuttujaan stashin sisälle.

Kuva 20. list: listaus, jossa näkyvät kaikki merkit.



Kuva 21. list_models: listaus merkin mukaan.

Mallikohtaiset tiedot

Yhteensä 41 tulosta. Näytetään 1-41.

Tulokset

Merkki	Malli	Mallin tarkennin	Kunto	Päätöspäivä	Käyttöönottopäivä	Mittarilukema tkm	Verotusarvo	Vero
MERCEDES-BENZ	190 E	1.8 4D 80 KW	Normaali	08-12-2009	09-10-1990	241	1198.00 €	319.78 €
MERCEDES-BENZ	190 E	1.8 4D 80KW	Normaali	17-03-2011	28-03-1991	127	1719.20 €	459.02 €
MERCEDES-BENZ	190 E	1.8 4D 80KW	Normaali	02-10-2009	18-10-1990	178	1698.90 €	453.60 €
MERCEDES-BENZ	190 E	1.8 4D 80KW	Normaali	28-11-2011	10-02-1992	370	1071.00 €	295.59 €
MERCEDES-BENZ	190 E	1.8 4D 80KW	Normaali	21-05-2010	02-10-1990	294	1316.00 €	351.37 €
MERCEDES-BENZ	190 E	1.8 4D 80KW	Normaali	22-11-2013	18-03-1991	216	1096.20 €	344.20 €
MERCEDES-BENZ	190 E	1.8 4D 80KW	Normaali	11-03-2010	20-03-1991	114	1721.00 €	459.40 €
MERCEDES-BENZ	190 E	1.8 4D 80KW	Normaali	12-01-2011	13-02-1991	191	1468.60 €	392.11 €
MERCEDES-BENZ	190 E	1.8 4D 80KW	Normaali	24-10-2013	04-02-1991	257	1146.60 €	360.03 €
MERCEDES-BENZ	190 E	1.8 4D AUT 80KW	-	20-11-2013	04-02-1993	53	1798.30 €	395.62 €
MERCEDES-BENZ	190 E	1.8 4D AUT 80KW	Normaali	18-02-2011	08-10-1991	113	1766.10 €	473.31 €
MERCEDES-BENZ	190 E	1.8 4D AUT 80KW	Normaali	16-12-2009	18-04-1991	211	1713.00 €	457.34 €
MERCEDES-BENZ	190 E	1.8 E Aut.	Normaali	02-07-2010	18-01-1993	264	1583.40 €	437.01 €
MERCEDES-BENZ	190 E	1.9 4D 80KW	Normaali	15-08-2013	11-01-1991	206	1244.60 €	390.80 €
MERCEDES-BENZ	190 E	2.0 4D 87KW	Normaali	23-12-2009	20-10-1988	368	830.00 €	222.30 €
MERCEDES-BENZ	190 E	2.0 4D 87KW	-	16-09-2014	06-07-1989	142	1157.10 €	365.64 €
MERCEDES-BENZ	190 E	2.0 4D 87KW	Normaali	25-10-2012	27-03-1991	158	1799.00 €	568.48 €
MERCEDES-BENZ	190 E	2.0 4D 87KW AUT	-	09-10-2014	20-02-1990	175	1164.10 €	367.85 €
MERCEDES-BENZ	190 E	2.0 4D 90 KW	Normaali	30-09-2010	11-01-1991	160	1874.60 €	502.39 €
MERCEDES-BENZ	190 E	2.0 4D 90KW	Normaali	18-01-2010	26-02-1992	161	1737.00 €	481.06 €
MERCEDES-BENZ	190 E	2.0 4D 90KW	-	22-09-2015	16-04-1985	-	906.00 €	262.74 €
MERCEDES-BENZ	190 E	2.0 4D 90KW	Normaali	21-10-2010	09-12-1992	183	1851.50 €	512.86 €
MERCEDES-BENZ	190 E	2.0 4D 90KW	Normaali	09-09-2009	04-10-1988	76	1345.40 €	355.18 €

Kuva 22. list_cars: listaus mallin mukaan.

Lopuksi lisäsin Makefile.PL-tiedostoon sovelluksessa käytettävät CPAN-moduulit.

```
requires 'DBIx::Class::Row';
requires 'DBIx::Class::ResultSet';
requires 'Data::Page';
requires 'MooseX::MethodAttributes';
requires 'MooseX::Types::Moose';
```

Kaiken tämän jälkeen, kun sovellusta oli testattu kehitysympäristössä selaimen avulla, oli aika kirjoittaa automaattisia testejä. Näitä kutsutaan yleisesti nimellä yksikkötesti (unit test). Testitiedostot ovat t/-hakemistossa, ja niitä ajetaan prove-ohjelmalla. Jotta saisin testattua erityisesti kontrollerit, tein t/-hakemistoon tiedostot controller_Root.t, controller_Search.t, ja controller_Ajax.t. En nähnyt tarvetta kirjoittaa kovin laajoja automatisoituja testejä, koska sovelluksessa ei käsitellä kriittistä tai arkaluontoista dataa, eikä sovelluksessa ole mitään erityisen monimutkaista toiminnallisuutta. Allaolevissa kuvissa on näytetty testitiedostot Root-kontrollerille (Kuva 23) ja Ajax-kontrollerille (Kuva 24). Root-kontrollerin testauksessa on hyödynnetty Test::WWW::Mechanize-moduulia.

```
use strict;
use warnings;
use Test::More;
use Test::WWW::Mechanize::Catalyst;

use Catalyst::Test 'Autoulkomailta';
use Autoulkomailta::Controller::Root;

my $mech = Test::WWW::Mechanize::Catalyst->new(catalyst_app => "Autoulkomailta");

$mech->get_ok("/");
is($mech->ct, "text/html");
$mech->get_ok("/verotus");
$mech->get_ok("/tuonti");
$mech->get_ok("/tietoa");

done_testing();
```

Kuva 23. controller_Root.t

```
use strict;
use warnings;
use Test::More;

use Catalyst::Test 'Autoulkomailta';
use Autoulkomailta::Controller::Ajax;

ok( request('/ajax')->is_success, 'Request should succeed' );
ok( request('/ajax/get_models/1')->is_success, 'Request should succeed' );

ok ( !request('/ajax/get_models/a')->is_success, 'Request should fail' );
ok ( !request('/ajax/do_search/')->is_success, 'Request should fail' );

ok ( request('/ajax/do_search/1/2/1990/2000/abc/asc/1')->is_success, 'Request should succeed' );
ok ( !request('/ajax/do_search/1/2/1990/2000/abc/asc/0')->is_success, 'Request should fail' );
ok ( !request('/ajax/do_search/1/2/1/2/abc/asc/0')->is_success, 'Request should fail' );
ok ( !request('/ajax/do_search/1/2/1/2/abc/asc/1')->is_success, 'Request should fail' );
ok ( !request('/ajax/do_search/a/a/1990/2000/abc/asc/0')->is_success, 'Request should fail' );
done_testing();
```

Kuva 24. controller_Ajax.t

Kun olin kirjoittanut testitiedostot valmiiksi, ajoin ne komennolla:

```
$ prove -wl t
```

Tämä ajoi kaikki testit t-hakemistossa, ja tulos oli alla olevan kuvan mukainen (Kuva 25), eli kaikki meni niinkuin pitikin.

```
fuel@devpc:~/dev/Autoulkomailta
[debug] Path is "/"
[debug] Arguments are "verotiedot/list/model/a"
[debug] "GET" request for "verotiedot/list/model/a" from "127.0.0.1"
[debug] Rendering template "error.tt"
[debug] Response Code: 404; Content-Type: text/html; charset=utf-8; Content-Leng
th: unknown
[info] Request took 0.011064s (90.383/s)
.------------------------------------------------------------+-----------.
| Action                                                     | Time      |
+------------------------------------------------------------+-----------+
| /default                                                   | 0.000664s |
| /end                                                       | 0.005013s |
|  -> Autoulkomailta::View::HTML->process                    | 0.004097s |
'------------------------------------------------------------+-----------'

t/controller_Search.t .. ok
t/model_DB.t ........... ok
t/view_HTML.t .......... ok
t/view_JSON.t .......... ok
All tests successful.
Files=9, Tests=31, 11 wallclock secs ( 0.07 usr  0.04 sys +  9.61 cusr  0.67 csy
s = 10.39 CPU)
Result: PASS
[fuel@devpc Autoulkomailta]$
```

Kuva 25. Testaamisen tulos.

5.4 Julkaisu

Nyt kun sovellus oli saatu julkaisuvalmiiksi, oli aika tehdä pieniä muutoksia siihen ja pakata se. Kopioin tämän version omaan hakemistoonsa, ja poistin debug-tilan käytöstä muokkaamalla lib/Autoulkomailta.pm-tiedostoa, sekä annoin samalla versionumeron 2.00 sovellukselle. Poistin myös käytöstä kehityksessä käytettyjä moduuleita kuten Data::Dumper. Kun kaikki tällaiset oli saatu tehtyä, varmistin että kaikki tarvittavat moduulit oli määritelty Makefile.PL-tiedostossa. Tämän jälkeen ajoin tämän komennolla

```
$ perl Makefile.PL
```

Jos jostain syystä olisi tarvetta paketoida kaikki tarvittavat moduulit mukaan, sekin onnistuu tässä vaiheessa cpanminus-ohjelmalla:

```
$ cpanm -L extlib --installdeps .
```

Tämä komento asentaa moduulit Makefile.PL:n tuottaman MYMETA-tiedoston perusteella extlib-alihakemistoon. Itse kuitenkin jätin tämän vaiheen pois tuotantovalmiista sovelluksesta, koska asennan moduulit mieluummin järjestelmän laajuisesti, jotta ne ovat useamman sovelluksen käytettäessä tarvittaessa, ja koska Catalyst-sovelluksen koko paisuu muuten kymmeniin megatavuihin. Seuraava askel oli joka tapauksessa pakata sovellus helppoa tuotantoon siirtämistä varten:

```
$ make manifest
$ make dist
```

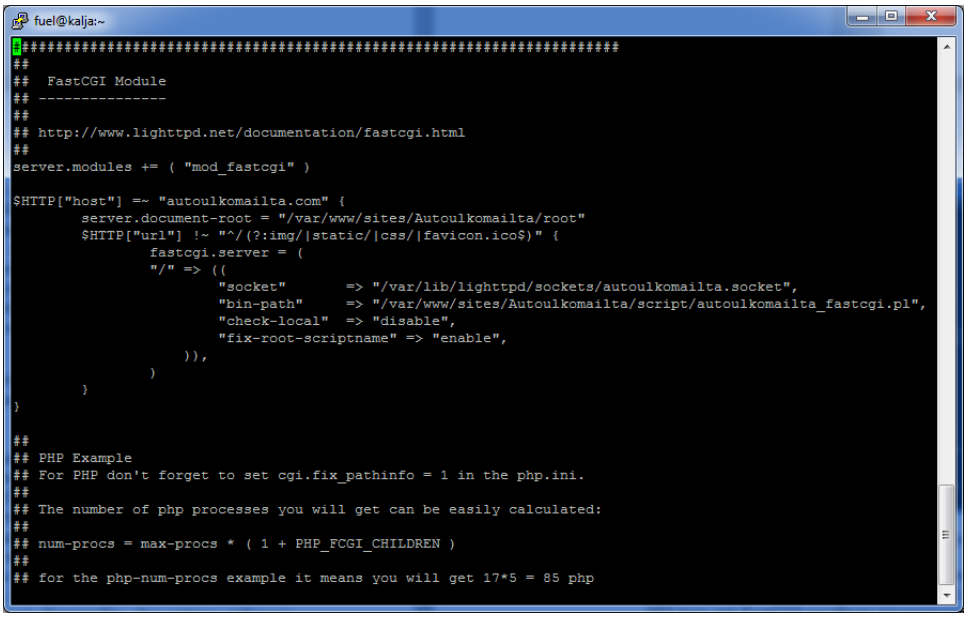
Nämä komennot loivat Autoulkomailta-2.00.tar.gz-tiedoston, eli gzipatun tarball-arkiston, joka voidaan vaikka kopioida scp-ohjelmalla tuotantopalvelimelle.

Catalyst tukee useita eri web-palvelimia kuten Apache, lighttpd, nginx, Starman, ja eri ympäristöjä kuten CGI, FastCGI, PSGI, ja mod_perl. Jos halutaan ajaa sovellus PSGI-ympäristössä, niin voidaan käyttää Makefile.PL:n luomaa psgi-tiedostoa sovelluksen juuressa. Eräs vaihtoehto tähän on käyttää Starman-palvelinohjelmistoa, jota voi käyttää sellaisenaan tai laittaa jokin muu palvelinohjelmisto välityspalvelimeksi eteen. Kokeilin sellaista julkaisutapaa, että laitoin Starman-palvelimen kuuntelemaan porttia 5000, ja Apachen toimimaan käänteisenä välityspalvelimena (reverse proxy) Starmanille. Jos päätyy tällaiseen ratkaisuun, niin suorituskyvyn kannalta lienee suotavampaa käyttää nginxia.

Oman sovellukseni julkaisu tapahtui lighttpd + FastCGI -alustalle. Tuotantopalvelin käyttää kehityspalvelimen tavoin myös CentOS Linux 7 -käyttöjärjestelmää. Vuokrasin virtuaalipalvelimen jossa on SSD-massamuisti ja riittävästi keskusmuistia. Tällöin tiedon hakeminen tietokannasta nopeutuu verrattuna tavalliseen kiintolevyyn, ja MariaDB-palvelin voidaan konfiguroida käyttämään osa keskusmuistista kyselyiden tallentamiseen välimuistiin, nopeuttaen tiedonhakua entisestään. Lighttpd:n sisäinen FastCGI-moduuli käynnistää automaattisesti annetun määrän prosesseja Catalyst-sovellukselle konfiguroinnin perusteella. Ensin asensin tuotantopalvelimelle yum:lla paketit lighttpd, lighttpd-fastcgi, ja spawn-fcgi. Sitten asensin cpanm:lla moduulit FCGI ja FCGI::ProcManager. Tämän jälkeen kun kaikki tarvittava oli asennettu, laitoin lighttpd:n hakemiston alla sijaitsevaan conf.d/fastcgi.conf-tiedostoon Catalyst-sovelluksen asetukset alla olevan kuvan (Kuva 26) mukaisesti. socket-asetus määrittää UNIX-socketin polun, mitä käytetään sovelluksen kanssa kommunikointiin. bin-path taas osoittaa autoulkomailta_fastcgi.pl-skriptin sijaintiin. fix-root-scriptname tarvitaan tässä tapauksessa, muuten URL-osoitteet sovelluksen sisällä menevät todennäköisesti rikki. Vaikka lighttpd:n vakioasetukset käyttävät kotihakemistona /var/lib/lighttpd-hakemistoa, ja sockets-hakemistoa sen alla, sitä ei kuitenkaan ole olemassa asennuksen jälkeen. Ne pitää siis luoda erikseen:

```
$ mkdir –p /var/lib/lighttpd/sockets
$ chown lighttpd:lighttpd /var/lib/lighttpd/sockets
```

Tämä luo /var/lib/lighttpd-hakemiston ja sen alle sockets-alihakemiston. Sockets-alihakemiston omistajaksi ja ryhmäksi laitetaan sama käyttäjä kuin millä HTTP-palvelinta ajetaan, tässä tapauksessa lighttpd.



Kuva 26. Lighttpd:n FastCGI-konfiguraatio Catalyst-sovellukselle.

Purin pakatun Catalyst-sovelluksen /var/www/sites/-hakemiston alle. Pikainen perl Makefile.PL -komento osoitti mitkä kaikki CPAN-moduulit puuttuvat. Ohjelma valittaa puuttuvista moduuleista, ja tarjoaa mahdollisuutta automaattiseen asennukseen CPAN-ohjelman avulla, mutta ainakaan omalla kohdallani se ei toiminut, joten asensin kaikki puuttuvat moduulit pääkäyttäjänä cpanminus-ohjelmalla. Jos jokin Catalyst-sovelluksen osa käyttää moduulia, jota ei ole määritelty Makefile.PL-tiedostossa, niin sovellus saattaa kyllä käynnistyä ja toimia näennäisesti hyvin, kunnes käyttäjä osuu siihen sovelluksen osaan mikä tarvitsee puuttuvaa moduulia. Tässä vaiheessa eteen tulee virheruutu, ja kun debug-tila ei ole päällä, voi vain ihmetellä mikä meni vikaan.

Kaiken tämän jälkeenkään sovellus ei toiminut, lighttpd ei suostunut edes käynnistymään. Logien tutkiminen vahvisti sen mitä epäilinkin, että SELinux esti FastCGI-sovellusta käynnistymästä, ja niin ollen kaatoi koko palvelimen. SELinuxin tiedostokonteksteja ei ollut konfiguroitu vielä, ja ne estivät näinollen HTTP-palvelinta mm. kirjoittamasta socket-hakemistoon tai suorittamasta autoulkomailta_fastcgi.pl-skriptiä. Ongelma oli suhteellisen helposti korjattavissa muuttamalla tiedostokontekstit pysyvästi semanage-ohjelmalla:

```
$ semanage fcontext -a -t httpd_sys_script_exec_t
/var/www/sites/Autoulkomailta/script/autoulkomailta_fastcgi
.pl

$ restorecon -R -v /var/www/sites/Autoulkomailta/script

$ semanage fcontext -a -t httpd_var_lib_t
/var/lib/lighttpd/

$ semanage fcontext -a -t httpd_var_run_t
'/var/lib/lighttpd/sockets(/.*)?'
```

```
$ restorecon -R -v /var/lib/lighttpd
```

Tässä annettiin autoulkomailta_fastcgi.pl -skriptille httpd_sys_script_exec_t -konteksti, joka sallii sen ajamisen CGI/FastCGI-rajapinnan kautta. /var/lib/lighttpd/sockets-hakemistolle ja sen alle luoduille tiedostoille annettiin httpd_var_run_t-konteksti, joka sallii kommunikoinnin UNIX-socketien välityksellä. /var/lib/lighttpd-hakemistolle annettiin httpd_var_lib_t-konteksti. Tässä vaiheessa käynnistin lighttpd:n, ja totesin pikaisen testauksen perusteella sovelluksen toimivaksi.

6 YHTEENVETO

Lopputuloksena oli web-sovellus, joka on osoittautunut toiminnaltaan nopeaksi ja muokattavuudeltaan helpoksi. Jos halutaan lisätä toiminnallisuutta myöhemmin, on hyvin helppoa jatkaa sovelluksen rakentamista vanhalta pohjalta. Tarkoituksena olisikin, että lisäisin sovellukseen vielä ainakin lomakkeen vapaata tekstihakua varten.

Sovelluksen kehityksessä opin paljon uusia asioita, ja hallitsen tämän projektin seurauksena web-sivujen teon Catalyst-ohjelmistokehyksen avulla kohtalaisen hyvin. Paljon on vielä opittavaa, sillä olihan kehittämäni sovellus varsin yksinkertainen. Jos lisätään sellaista toiminnallisuutta, kuten käyttäjien autentikointi tai muuta vastaavaa, niin sovelluksen monimutkaisuus kasvaa.

Täytyy mainita, että tämänkaltaisessa kehityksessä olisi hyvin suotavaa käyttää jotain versionhallintajärjestelmää, kuten esimerkiksi Git tai Subversion. Nykyinen tapa ylläpitää sovellusta on testata muutokset kehityspalvelimella, ja kopioida sitten scp-ohjelmalla muokatut tiedostot tuotantoympäristöön. Tämä tapa toimii hyvin niin kauan kuin muutokset ovat pieniä, ja muokataan vain paria tiedostoa, mutta vastaisuudessa lienee syytä siirtyä käyttämään jotain versionhallintaa.

LÄHTEET

Brinzarea-Iamandi, B. & Darie, C. & Hendrix, A.:
AJAX and PHP – Building Modern Web Applications – 2nd Edition
Packt Publishing, 2009, ISBN: 978-1-847197-72-6

Diment, K. & Trout, M.: The Definitive Guide To Catalyst
Apress, 2009, ISBN: 978-1-4302-2366-5

Dyer, R.: Learning MySQL and MariaDB
O'Reilly Media, 2015, ISBN: 978-1-449-36290-4

Miyagawa, T. PSGI.
http://search.cpan.org/~miyagawa/PSGI-1.102/PSGI.pod
Viitattu 5.2.2016

O'Reilly, T. & Smith, B. The Importance of Perl.
http://archive.oreilly.com/pub/a/oreilly/perl/news/importance_0498.html
Viitattu 30.10.2015

Perl.org a. About Perl.
https://www.perl.org/about.html
Viitattu 8.2.2016

Perl.org b. Perl web framework – Catalyst.
https://www.perl.org/about/whitepapers/perl-webframework.html
Viitattu 8.2.2016

Tiobe.com. TIOBE Index for February 2016.
http://www.tiobe.com/tiobe_index
Viitattu 15.2.2016

Wikipedia 2015a. Catalyst.
https://en.wikipedia.org/wiki/Catalyst_%28software%29
Viitattu 30.10.2015

Wikipedia 2015b. MariaDB.
https://en.wikipedia.org/wiki/MariaDB
Viitattu 30.10.2015

Liite 1

AUTOULKOMAILTA.PM

```
package Autoulkomailta;
use utf8;
use Moose;
use namespace::autoclean;

use Catalyst::Runtime 5.80;

# Set flags and add plugins for the application.
#
# Note that ORDERING IS IMPORTANT here as plugins are initialized in order,
# therefore you almost certainly want to keep ConfigLoader at the head of the
# list if you're using it.
#
#         -Debug: activates the debug mode for very useful log messages
#   ConfigLoader: will load the configuration from a Config::General file in the
#                 application's home directory
# Static::Simple: will serve static files from the application's root
#                 directory

use Catalyst qw/
    -Debug
    ConfigLoader
    Static::Simple
/;

extends 'Catalyst';

our $VERSION = '0.01';

# Configure the application.
#
# Note that settings in autoulkomailta.conf (or other external
# configuration file that you set up manually) take precedence
# over this when using ConfigLoader. Thus configuration
# details given here can function as a default configuration,
# with an external configuration file acting as an override for
# local deployment.

__PACKAGE__->config(
    name => 'Autoulkomailta',
    encoding => 'UTF-8',
    default_view => "HTML",
    # Disable deprecated behavior needed by old applications
    disable_component_resolution_regex_fallback => 1,
    enable_catalyst_header => 1, # Send X-Catalyst header
);

# Start the application
__PACKAGE__->setup();
```

Liite 2

ROOT.PM

```
package Autoulkomailta::Controller::Root;
use Moose;
use namespace::autoclean;

BEGIN { extends 'Catalyst::Controller' }

#
# Sets the actions in this controller to be registered with no prefix
# so they function identically to actions created in MyApp.pm
#
__PACKAGE__->config(namespace => '');

=encoding utf-8

=head1 NAME

Autoulkomailta::Controller::Root - Root Controller for Autoulkomailta

=head1 DESCRIPTION

[enter your description here]

=head1 METHODS

=head2 index

The root page (/)

=cut

sub index :Path :Args(0) {
    my ( $self, $c ) = @_;
    #$c->stash(load_js => 1);
}

sub verotus :Local { }

sub tuonti :Local { }

sub tietoa :Local { }

=head2 default

Standard 404 error page

=cut

sub default :Path {
    my ( $self, $c ) = @_;
    #$c->response->body( 'Page not found' );
    $c->response->status(404);
    $c->stash(template => "error.tt");
}
```

Liite 3

AJAX.PM – OSA 1

```
package Autoulkomailta::Controller::Ajax;
use utf8;
use Moose;
#use namespace::autoclean;
use MooseX::MethodAttributes;
use MooseX::Types -declare => [
    qw(PosInt YearInt)
];
use MooseX::Types::Moose qw(Int Str);

BEGIN { extends 'Catalyst::Controller'; }

# create custom subtypes to reduce the amount of
# checks we have to write in the subroutines

subtype PosInt,
 as Int,
 where { $_ > 0 };

subtype YearInt,
 as Int,
 where { $_ >= 1899 && $_ <= 2021 };

=head1 NAME


=head2 index

sub index :Path :Args(0) {
    my ( $self, $c ) = @_;

    $c->response->body('Nothing to see here.');
}

sub base :Chained("/") :PathPart("ajax") :CaptureArgs(0) {
    my ($self, $c) = @_;

    $c->stash->{schema} = $c->model("DB")->schema;
    $c->stash(no_wrapper => 1); # disable wrapper
}

# returns list of models by given manufacturer ID in JSON format
sub get_models :Chained("base") :PathPart("get_models") :Args(Int) {
    my ( $self, $c, $id ) = @_;

    my $rs = [$c->stash->{schema}->resultset("Model")->search({fk_manufacturer_id => $id})];
    my @list;

    foreach(@$rs) {
        push @list, { model_id => $_->model_id, model_name => $_->model_name };
    }
    $c->stash(model_data => \@list);

    delete $c->stash->{schema};
    $c->forward("View::JSON");
}
```

Liite 4

AJAX.PM – OSA 2

```
sub do_search :Chained("base") :PathPart("do_search") :Args(Int, Int, YearInt, YearInt, Str, Str, PosInt) {
    my ( $self, $c, $manufacturer_id, $model_id, $year_start, $year_end, $sort_by, $order, $page ) = @_;

    # some basic checks
    $manufacturer_id = 0 if $manufacturer_id < 0;
    $model_id = 0 if $model_id < 0;

    $year_start = $year_end if($year_start > $year_end);

    # create datetime objects
    my $ys = DateTime->new(year=>$year_start, month => 1, day => 1);
    my $ye = DateTime->new(year=>$year_end, month => 12, day => 31);
    my $dtf = $c->stash->{schema}->storage->datetime_parser;

    # can't insert client-supplied strings directly into the DBIx::Class::ResultSet order_by-clause
    # due to possible SQL injection, so I'm using these workarounds
    my @sorts = (
        "car_info", "commission_date", "tax_decision_date", "mileage", "value",
        );
    my $sorter = $sorts[0];
    for(@sorts) {
        $sorter = $_ if $_ eq $sort_by;
    }
    my $q_order = ($order eq "asc" ? "ASC" : "DESC");

    # a nonzero model id or manufacturer id was given
    if($model_id || $manufacturer_id) {
        my $rs = $c->stash->{schema}->resultset("Car")->search(
                                    {
                                      # search based either on the model id or manufacturer id
                                      ($model_id ? ("fk_model_id" => $model_id) : ("fk_model.fk_manufacturer_id" => $manufacturer_id) ),
                                      "commission_date" =>
                                          { -between => [
                                              $dtf->format_datetime($ys),
                                              $dtf->format_datetime($ye),
                                            ],
                                          }
                                    },
                                    {
                                      prefetch => "fk_model",
                                      order_by => \"$sorter $q_order",
                                      page => $page,
                                      rows => 50,
                                    },);
        $c->stash(pages => $rs->pager, search_data => [$rs->all]);
    }
    else {
        $c->stash(message => "Vähintään merkki vaaditaan.");
    }
}
```

Liite 5

SEARCH.PM – OSA 1

```
package Autoulkomailta::Controller::Search;
use utf8;
#use Data::Dumper;
use Moose;
use DateTime;
#use Scalar::Util::Numeric qw(isint);
use Data::Page;
use MooseX::MethodAttributes;
use MooseX::Types -declare => [
    qw(PosInt)
];
use MooseX::Types::Moose qw(Int);
#use MooseX:MethodAttributes;
#use Autoulkomailta::Types qw(int);
#use namespace::autoclean;

BEGIN { extends 'Catalyst::Controller'; }

# greater than or equal to zero
subtype PosInt,
 as Int,
 where { $_ >= 0 };

=head1 NAME


=head2 index

sub base :Chained("/") :PathPart("verotiedot") :CaptureArgs(0) {
    my ($self, $c) = @_;
    my $schema = $c->model("DB")->schema;
    my $rs = $schema->resultset("Manufacturer")->search({});
    $c->stash(schema => $schema,
              template => "search/index.tt",
              manufacturer_list => [$rs->all],
              );
}

# index
sub index :Chained("base") :PathPart("") :Args(0) {
}

sub search :Chained("base") :PathPart("search") :Args(0) {
    my ( $self, $c ) = @_;

    $c->stash->{template} = "search/search_ajax.tt";
    $c->stash(max_year => 2020, min_year => 1900);
}

# list all manufacturers
sub list :Chained("base") :PathPart("list") :Args(0) {
    my ( $self, $c ) = @_;
    $c->stash->{template} = "search/list.tt";
    $c->stash->{count} = $c->stash->{schema}->resultset("Car")->count;
}
```

Liite 6

SEARCH.PM – OSA 2

```
# list all models of given manufacturer id
sub list_models :Chained("base") :PathPart("list/make") :Args(0) {
    my ( $self, $c ) = @_;

    $c->response->body("No arguments given.");
}
sub list_models_do :Chained("base") :PathPart("list/make") :Args(PosInt) {
    my ( $self, $c, $m_id ) = @_;

    $c->stash->{template} = "search/list_models.tt";
    my $rs = $c->stash->{schema}->resultset("Model")->search({ fk_manufacturer_id => $m_id },
                                                             { prefetch => "fk_manufacturer" });
    return unless $rs->count;
    $c->stash->{model_list} = [$rs->all];
}

# list all cars under given model id
sub list_cars :Chained("base") :PathPart("list/model") :Args(0) {
    my ( $self, $c ) = @_;

    $c->response->body("No arguments given.");
}
sub list_cars_by_page :Chained("base") :PathPart("list/model") :Args(PosInt, PosInt) {
    my ($self, $c, $m_id, $page_num) = @_;

    $c->stash(page => $page_num);
    $c->forward("list_cars_do");
}
sub list_cars_do :Chained("base") :PathPart("list/model") :Args(PosInt) {
    my ( $self, $c, $m_id ) = @_;

    $c->stash->{template} = "search/list_cars.tt";
    my $current_page = $c->stash->{"page"} || 1;

    my $rs = $c->stash->{schema}->resultset("Car")->search({ fk_model_id => $m_id },
                                                           { prefetch => "fk_model",
                                                             order_by => ["car_info"],
                                                             page => $current_page,
                                                             rows => 100,
                                                             });

    return unless $rs->count;
    $c->stash(pages => $rs->pager, search_data => [$rs->all]);
}
```

Liite 7

SCRIPTS.JS – OSA 1

```
var srs = document.getElementById("search_rs");
var sbmaid = document.getElementById("manufacturer_id");
var sbmdid = document.getElementById("model_id");
var sbys = document.getElementById("year_start");
var sbye = document.getElementById("year_end");
var sbsb = document.getElementById("sort_by");
var orders = document.getElementsByName("order");
var sss = document.getElementById("search_status");

function check_year() {
    if(sbys.options[sbys.selectedIndex].value  > sbye.options[sbye.selectedIndex].value ) {
        sbye.selectedIndex = sbys.selectedIndex;
    }

}
// function to update model list
function list_update() {
    var sbmaid_value = sbmaid.options[sbmaid.selectedIndex].value;
    var sbmdid_value = sbmdid.options[sbmdid.selectedIndex].value;
    var data;

    if(sbmaid_value < 1 || isNaN(sbmaid_value)) {
        while(sbmdid.options.length > 0) {
                sbmdid.remove(0);
        }
        sbmdid.options.add(new Option("Malli", 0));
        return;
    }

    var xhttp = new XMLHttpRequest();
    xhttp.overrideMimeType("application/json");
    xhttp.onreadystatechange = function() {
        if(xhttp.readyState == 4 && xhttp.status == 200) {
            data = JSON.parse(xhttp.responseText);

            if(data === undefined) {
                return;
            }

            while(sbmdid.options.length > 0) {
                sbmdid.remove(0);
            }

            sbmdid.options.add(new Option("Malli", 0));

            for(var i = 0; i < data.model_data.length; i++) {
                var obj = data.model_data[i];
                var opt = document.createElement("option");
                sbmdid.options.add(new Option(obj.model_name.toString(), obj.model_id.toString()));
            }
        }
    };

    xhttp.open("GET", "/ajax/get_models/" + sbmaid_value, true);
    xhttp.send();
}
```

Liite 8

SCRIPTS.JS – OSA 2

```
// vehicle search form, ajax version
function do_search(page) {

    var make_id = sbmaid.options[sbmaid.selectedIndex].value;
    var model_id = sbmdid.options[sbmdid.selectedIndex].value;
    var year_start = sbys.options[sbys.selectedIndex].value;
    var year_end = sbye.options[sbye.selectedIndex].value;
    var sort_by = sbsb.options[sbsb.selectedIndex].value;
    var order_value;

    if(typeof page === 'undefined' || page < 1) {
        page = 1;
    }

    for(var i = 0; i < orders.length; i++) {
        if(orders[i].checked) {
            order_value = orders[i].value;
            break;
        }
    }

    sss.innerHTML = "Ladataan...";

    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if(xhttp.readyState == 4) {
            if(xhttp.status == 200)
                var data = xhttp.responseText;

                if(data === undefined) {
                    return;
                }
                srs.innerHTML = data;
                sss.innerHTML = "";
            }
            else if(xhttp.status == 404) {
                srs.innerHTML = "Virhe tapahtui tietojen haussa. (404 Not found)";
            }
        };

    xhttp.open("GET", "/ajax/do_search/"+make_id+"/"+model_id+"/"+year_start+"/"+year_end+"/"+sort_by+"/"+order_value+"/"+page, true);
    xhttp.ontimeout = 5000;
    xhttp.ontimeout = function () { srs.innerHTML = "Virhe tapahtui tietojen haussa. (Timed out)"; }
    xhttp.send();
}
```

Liite 9

DO_SEARCH.TT

```
 [% IF pages -%]
 <p>Yhteensä [% pages.total_entries %] tulosta. Näytetään [% pages.first %]-[% pages.last %].</p>
<p>
 [% IF pages.previous_page -%]
 <a href="javascript:do_search([% pages.previous_page %]);">Edellinen sivu</a>
 [% END -%]
 [% IF pages.next_page -%]
 <a href="javascript:do_search([% pages.next_page %]);">Seuraava sivu</a>
 [% END -%]
</p>
 [% END -%]
 [% IF search_data %]
 <h2>Tulokset</h2>
<table class="search_data">
<tr>
 <th>Merkki</th>
 <th>Malli</th>
 <th>Mallin tarkennin</th>
 <th>Kunto</th>
 <th>Päätöspäivä</th>
 <th>Käyttöönottopäivä</th>
 <th>Mittarilukema tkm</th>
 <th>Verotusarvo</th>
 <th>Vero</th>
</tr>
 [% FOREACH s IN search_data -%]
<tr>
 [% UNLESS vars -%]
 [% t_vars = 1 -%]
 [% t_make_id = s.fk_model.fk_manufacturer.manufacturer_id -%]
 [% t_model_id = s.fk_model.model_id -%]
 [% END -%]
 <td>[% s.fk_model.fk_manufacturer.manufacturer_name %]</td>
 <td>[% s.fk_model.model_name %]</td>
 <td>[% s.car_info || "-"%]</td>
<td>[% IF (s.cond == 'N') %]Normaali
 [% ELSIF (s.cond == 'H') %]Huono
 [% ELSIF (s.cond == 'Y') %]Hyvä
 [% ELSIF (s.cond == 'A') %]Alennettu
 [% ELSE %]-
 [% END -%]</td>
 <td>[% s.tax_decision_date.dmy || "-" %]</td>
 <td>[% s.commission_date.dmy || "-" %]</td>
 <td>[% s.mileage || "-" %]</td>
 <td>[% s.value || "-" %] €</td>
 <td>[% s.tax_amount || "-" %] €</td>
</tr>
 [% END -%]
</table>
 [% END -%]
 [% IF message -%]
 <p>[% message %]</p>
 [% END -%]
```

Liite 10

SEARCH_AJAX.TT

```
[% title="Verotietojen haku" %]
<h2>Verotietojen haku</h2>
<noscript><p class="message">Lomake vaatii Javascriptin toimiakseen.</p></noscript>
<p><a href="[% c.uri_for(c.controller("search").action_for("index")) %]">Palaa takaisin</a></p>
<p><a href="[% c.uri_for(c.action) %]">Tyhjennä lomake</a></p>
<form>
<label for="manufacturer_id">Merkki:</label>
<select name="make" id="manufacturer_id" onchange="list_update()">
<option value="0">Merkki</option>
[% FOREACH d IN manufacturer_list -%]
<option value="[% d.manufacturer_id %]">[% d.manufacturer_name %]</option>
[% END -%]
</select>
<label for="model_id">Malli:</label>
<select id="model_id" name="model">
<option value="0">Malli</option>
</select>
<label for="year_start">Vuosimalli alkaen:</label>
<select name="year_start" id="year_start" onchange="check_year()">
<option value="[% min_year - 1 %]" selected="selected">Valitse</option>
[% year = min_year -%]
[% WHILE year <= max_year -%]
<option value="[% year %]">[% year %]</option>
[% year = year + 1 -%]
[% END -%]
</select>
<label for="year_end">Vuosimalli päättyen:</label>
<select name="year_end" id="year_end" onchange="check_year()">
<option value="[% max_year + 1 %]" selected="selected">Valitse</option>
[% year = min_year -%]
[% WHILE year <= max_year -%]
<option value="[% year %]">[% year %]</option>
[% year = year + 1 -%]
[% END -%]
</select>
<label for="sort_by">Järjestys:</label>
<select id="sort_by" name="sort_by">
<option value="car_info">Mallin tarkennin</option>
<option value="commission_date">Käyttöönottopäivä</option>
<option value="tax_decision_date">Päätöspäivä</option>
<option value="mileage">Mittarilukema</option>
<option value="value">Verotusarvo</option>
</select>
<input type="radio" name="order" value="asc" checked="checked">Nouseva
<input type="radio" name="order" value="desc">Laskeva<br>
<input type="button" value="Hae" onclick="do_search()">
<span id="search_status"></span>
</form>
<hr>
<div id="search_rs"></div>
<script src="[% c.uri_for("/scripts.js") %]" type="text/javascript"></script>
<script type="text/javascript">
document.onload = list_update();
</script>
```

Liite 11

PARSE.PL – OSA 1

```
use strict;
use warnings;
use Text::CSV;
use List::MoreUtils qw(uniq);
use Data::Dumper qw(Dumper);
use utf8;

#my $input_files = "/home/fuel/db/input/*.csv"; # input directory
my $directory = "output"; # output directory
my $original_file = "all.csv";
#my $original_file = "kaikki.csv";
my $in_file = "verotiedot.csv";
my $m_out_file = "$directory/manufacturer.csv";
my $mo_out_file = "$directory/model.csv";
my $v_out_file = "$directory/car.csv";
my @makes;
my %models = ();
#my %ids = ();
my @container = ();
my $csv = Text::CSV->new({ sep_char => "|" }) or die "Cannot use CSV: ".Text::CSV->error_diag();

# check if output directory exists, try to create it if it doesn't
unless(-e $directory) {
    if(mkdir($directory, 0755)) {
        print "Created directory $directory\n";
    } else {
        die "Unable to create $directory\n";
    }
}

# we need to start by converting the decimal values into the correct format
open my $rfh, "<:encoding(utf8)", $original_file or die "$original_file: $!";
open my $wfh, ">:encoding(utf8)", $in_file or die "$in_file: $!";

while(my $line = <$rfh>) {
    # skip all lines with column names
    next if $line =~ /Merkki/ or $line =~ /Märke/ or $line =~ /Modell/;
    # replace commas with dots
    #$line =~ s/(\d+),(\d+)/$1\.$2/g;
    $line =~ s/,/\./g;
    print $wfh $line;
}

close $rfh;
close $wfh;

# now that source file is in correct format, we can read data from it
open my $fh, "<:encoding(utf8)", $in_file or die "$in_file: $!";
```

Liite 12

PARSE.PL – OSA 2

```
while (my $line = <$fh>) {
  chomp $line;

  if ($csv->parse($line)) {
      my @fields = $csv->fields();
      push @makes, $fields[0];
      #$models{$fields[0]} = $fields[1];
      push @{$models{$fields[0]}}, $fields[1];
  } else {
      warn "$in_file: Line could not be parsed: $line\n";
  }
}

#$csv->eol("\r\n");
close $fh;

# remove all duplicate makes and sort list
@makes = sort(uniq(@makes));

# sort the model names
foreach my $key (keys %models) {
    @{$models{$key}} = sort(uniq(@{$models{$key}}));
}
my $count = 1;

# write the makes file
open $fh, ">:encoding(utf8)", $m_out_file or die "$m_out_file: $!";
print $fh "manufacturer_id|manufacturer_name\n";
for (@makes) {
    print $fh "$count|$_\n";
    $count++;
}
close $fh or die "$m_out_file: $!";

print "Wrote $m_out_file\n";

$count = 1;

# write the models file
open $fh, ">:encoding(utf8)", $mo_out_file or die "$mo_out_file: $!";
print $fh "model_id|model_name|fk_manufacturer_id\n";
while((my $key, my $value) = each %models) {
    for(@{$models{$key}}) {
        my $index = 0;
        #next unless $_;
        print $fh "$count|$_|";
        # we need the correct index number for the database make-model foreign key
        ++$index until $makes[$index] eq $key or $index > $#makes;
        #$ids{$_} = $count;
        #push @$container, { id => $count, name => $_, make => $makes[$index] };
        push @container, [ $makes[$index], $_, $count ];
        #my ($index) = grep { $makes[$_] eq $key } 0..$#makes;
        $index++;
        print $fh "$index\n";
        $count++;
    }
}
```

Liite 13

PARSE.PL – OSA 3

```
close $fh or die "$mo_out_file: $!";

print "Wrote $mo_out_file\n";

print "Working, this will take a while...\n";

# write data to vehicle file, unfortunately we need to read the source file again
open $fh, "<:encoding(utf8)", $in_file or die "$in_file: $!";
open my $vfh, ">:encoding(utf8)", $v_out_file or die "$v_out_file: $!";
print $vfh "id|car_info|cond|tax_decision_date|commission_date|mileage|value|tax_amount|fk_model_id\n";

$count = 1;

while (my $line = <$fh>) {
    chomp $line;

    if ($csv->parse($line)) {
        my @fields = $csv->fields();
        my $index; # find the corresponding model id number for this vehicle

        # this really kills performance, but it's necessary to retain data integrity
        # now using AoA instead of AoH, should be faster
        foreach my $row (@container) {
            #if(($ref->{make} eq $fields[0]) && ($ref->{name} eq $fields[1])) {
                #$index = $ref->{id};
            if(($row->[0] eq $fields[0]) && ($row->[1] eq $fields[1])) {
                $index = $row->[2];
                last;
            }
        }
        next unless defined $index; # failed to find id, skip

        #if($fields[0] ne "Merkki") { # skip the first line, this seems to be the only method that works in this case
        $fields[7] =~ tr/ //ds;
        $fields[8] =~ tr/ //ds;
        print $vfh "$count|";
        print $vfh $fields[$_]."|" for(2..8);
        print $vfh "$index\n";
        $count++;
    } else {
        warn "$v_out_file: Line could not be parsed: $line\n";
    }
}

print "Wrote $v_out_file\n";


close $vfh;
close $fh;
```

Liite 14

UPGRADE.SH

```
#!/usr/bin/bash
# exit on error
set -e

user=root
host=localhost
password=avprojekti2014
database=mydb
schemafile=schema.sql
out_dir=output
in_dir=input

echo Autoulkomailta.com automated database update script v1.0
echo Just put the CSV-files in the $in_dir directory. Make sure they are all in UTF-8 format!
echo Generating DB output files in the $out_dir directory.
echo Probably a good idea to try migrating data to a test database first.

echo Combining files...
dos2unix input/*.csv
rm all.csv
cat input/*.csv > all.csv

echo Converting files...
perl parse.pl

# we have to disable foreign key checks or deleting the tables will fail
mysql --user="$user" -e "SET GLOBAL FOREIGN_KEY_CHECKS = 0"
echo Deleting old tables...
mysql --user="$user" --database="$database" --execute="DROP TABLE IF EXISTS make,model,vehicle,manufacturer,car;"
echo Installing new schema...
mysql --user="$user" $database < $schemafile

echo Importing new data from CSV...
# we can load the list of makes with mysqlimport because there are no foreign key constraints
mysqlimport --ignore-lines=1 --fields-terminated-by="|" --local --user "$user" --silent "$database" $out_dir/manufacturer.csv
mysqlimport --ignore-lines=1 --fields-terminated-by="|" --local --user "$user" --silent "$database" $out_dir/model.csv
mysqlimport --ignore-lines=1 --fields-terminated-by="|" --local --user "$user" --silent "$database" $out_dir/car.csv
# data is imported, enable foreign key checks again
mysql --user="$user" -e "SET GLOBAL FOREIGN_KEY_CHECKS = 1"
```

Liite 15

LIST.TT

```
 [% title = "Autoveropäätökset selattavassa muodossa" %]
 <h2>[% title %]</h2>
<p>Täältä löydät listattuna Tullin autoveropäätökset vuosilta 2009-2015. Autoja tietokannassa yhteensä: [% count %].<br>
Tiedot ovat vapaasti saatavilla <a href="http://www.tulli.fi/fi/yksityisille/autoverotus/taulukot/autot/index.jsp">Tullin verkkosivuilta</a>.</p>
<p><a href="[% c.uri_for(c.controller("search").action_for("index")) %]">Palaa takaisin</a></p>
<ul class="listdata">
 [% FOREACH m IN manufacturer_list -%]
<li><a href="[% c.uri_for(c.controller("search").action_for("list_models"), (m.manufacturer_id)) %]">[% m.manufacturer_name %]</a></li>
 [% END -%]
 </ul>
```

Liite 16

LIST_MODELS.TT

```
<h2>Kaikki mallit</h2>
<p><a href="[% c.uri_for(c.controller("search").action_for("list")) %]">Takaisin alkuun</a></p>
[% IF model_list -%]
<ul class="listdata">
[% FOREACH m IN model_list -%]
[% UNLESS title -%]
[% title = "Autoveropäätökset :: " _ m.fk_manufacturer.manufacturer_name -%]
[% END -%]
<li><a href="[% c.uri_for(c.controller("search").action_for("list_cars"), (m.model_id)) %]">[% m.fk_manufacturer.manufacturer_name %] [% m.model_name %]</a></li>
[% END -%]
</ul>
[% ELSE -%]
<p>Ei tuloksia</p>
[% END -%]
```

Liite 17

LIST_CARS.TT

```
<h2>Mallikohtaiset tiedot</h2>
[% PROCESS "search/search_template.tt" -%]
[% title = "Autoveropäätökset :: " _ t_make_name _ " " _ t_model_name -%]
[% IF pages -%]
<p><b>Sivu [% pages.current_page %] / [% pages.last_page %]</b></p>
[% END -%]
[% IF pages.previous_page -%]
<a href="[% c.uri_for(c.controller("search").action_for("list_cars"), (t_model_id), (pages.previous_page)) %]">Edellinen</a>
[% END -%]
[% IF pages.next_page -%]
<a href="[% c.uri_for(c.controller("search").action_for("list_cars"), (t_model_id), (pages.next_page)) %]">Seuraava</a>
[% END -%]
<p><a href="[% c.uri_for(c.controller("search").action_for("list_models"), (t_make_id)) %]">Takaisin malliluetteloon</a></p>
```