Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma
ICT-yrittäjyys

Juha Koskelainen

Selainpohjainen osaamisenhallinnan sovellus tehtaan työnjohdolle

Opinnäytetyö 2016

Tiivistelmä

Juha Koskelainen
Selainpohjainen osaamisenhallinnan sovellus tehtaan työnjohdolle, 49 sivua
Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma
ICT-yrittäjyys
Opinnäytetyö 2016
Ohjaaja: TkT, lehtori Pasi Juvonen, Saimaan ammattikorkeakoulu

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa tietokantapohjainen sovellus osaamisenhallinnan työkaluksi. Ohjelman tilaajana toimi puujalosteita valmistava Metsä Wood Punkaharju.

Asiakkaan toiveiden mukainen sovellus oli luonteva toteuttaa web-teknologioita ja avoimen lähdekoodin ratkaisuja hyödyntäen. Sovellus rakentui muun muassa Python-ohjelmointikielen, Flask-sovelluskehyksen, SQLite-tietokantamoottorin, SQLBuilder-kirjaston, jQuery-kirjaston ja DataTables-taulukkokomponentin varaan.

Projektin tuloksena syntyi palvelinosasta, asiakasosasta sekä niitä yhdistävästä rajapinnasta koostuva verkkoselaimen kautta käytettävä ohjelma.

Asiasanat: web-ohjelmointi, asiakas-palvelin-malli, HTTP, Ajax

Abstract

Juha Koskelainen
Browser-based Competence Management Application for Plant Supervisors, 49 Pages
Saimaa University of Applied Sciences
Technology Lappeenranta
Degree Programme in Information Technology
Specialisation in Entrepreneurship
Bachelor´s Thesis 2016
Instructor: D.Sc. (Eng.) Pasi Juvonen, Senior lecturer, Saimaa University of Applied Sciences

The purpose of the project was to develop a database-driven application to assist in competence management. The project was commissioned by Metsä Wood Punkaharju, a manufacturer of engineered wood products.

The implementation of the application was carried out using common web technologies and selected open source solutions. These included Python 3, Flask web framework, SQLite database engine, SQLBuilder toolkit, jQuery library, and DataTables grid component.

As a result of the project, an application based on the client-server model was created along with an API that enables access to the data on the server through a web browser or any HTTP client.

Keywords: web development, client-server model, HTTP, Ajax

Sisällys

Termit

API	Application programming interface. Määritelty rajapinta tiedon käsittelyyn.
CSS	Cascading Style Sheets. Kuvauskieli jolla määritellään web-sivun ulkoasu.
HTML	HyperText Markup Language. Kuvauskieli jolla määritellään web-sivun rakenne ja sisältö.
JavaScript	Verkkoselainten tukema ohjelmointikieli.
JSON	JavaScript Object Notation. Tiedonsiirtoon suunnattu tiedon tallennusmuoto.
osaamismatriisi	Taulukko joka sisältää henkilöstön osaamistietoja.
RFC	Request for Comments. Määrittelydokumentin julkaisumuoto.
SQL	Structured Query Language. Relaatiotietokantojen tukema komentokieli.
viite-eheys	Tietokantataulujen avainkenttiin perustuva tekniikka, jolla varmistetaan toisiinsa liittyvien tietojen eheys.

1 Johdanto

Yli kaksisataa henkilöä työllistävässä tehtaassa esimiestyö ja henkilöstöhallinto ovat merkittävässä asemassa, ja niiden tueksi tarvitaan myös tietoteknisiä työkaluja. Metsä Wood -yrityksen Punkaharjun vaneritehtaalla on käytetty osaamisenhallinnan työkaluna Microsoft Excel -taulukkolaskentaohjelmalla toteutettua osaamismatriisia. Opinnäytetyöni tarkoituksena oli suunnitella ja toteuttaa tämän ratkaisun korvaava ohjelma, joka soveltuisi tehtävään laskentataulukkoa paremmin.

Asiakkaan toiveena oli kevyt ja yksinkertainen työkalu, jolla osaamistietoja on helppo päivittää ja jolla voidaan tuottaa visuaalisia raportteja. Suunnittelun pohjaksi asiakas toimitti minulle osittaisen mallin ohjelman toivotusta käyttöliittymästä sekä osaamismatriisin, josta oli poistettu työntekijöiden nimet.

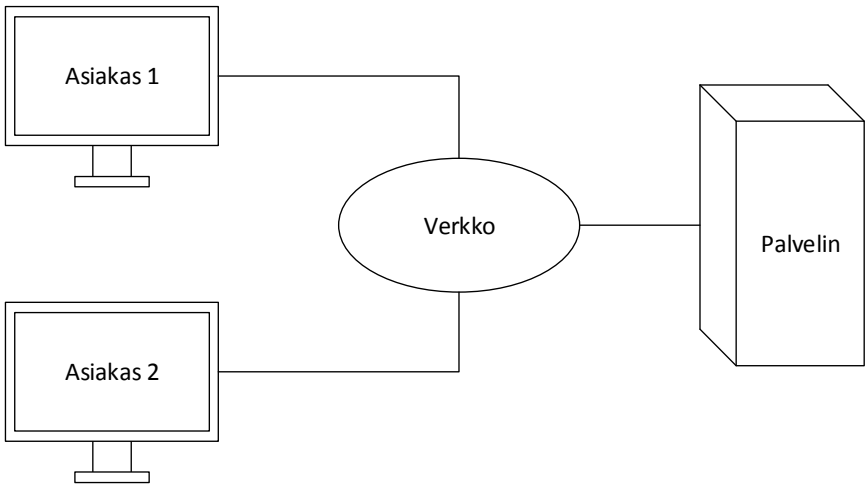
Ohjelman suunnittelua ohjasi viisi pääasiallista tavoitetta. Ensinnäkin ohjelman tulisi palvella mahdollisesti useita samanaikaisia käyttäjiä. Toiseksi itse ohjelman asentamisen, päivittämisen ja ylläpitämisen tulisi olla vaivatonta. Kolmanneksi ohjelmalla tulisi olla asiakkaan toimittaman mallin mukaisesti selkeä ja yksinkertainen käyttöliittymä. Neljäs tavoite oli toteuttaa ohjelma asiakas-palvelin-mallin mukaisesti siten, että se tarjoaisi yksinkertaisen rajapinnan osaamistietojen käsittelyyn. Viidentenä tavoitteena oli toteuttaa asiakkaan toivomat visuaaliset raportit.

Opinnäytetyöni oli siis sovellusprojekti, jonka toteutin web-tekniikoita ja valmiita avoimen lähdekoodin ratkaisuja hyödyntämällä. Ohjelma on luonteeltaan web-sovellus, mutta se on nykyisessä muodossaan tarkoitettu vain intranetissä käytettäväksi. Ohjelma muodostuu palvelinpuolesta, joka on toteutettu Python 3 -ohjelmointikielellä, sekä asiakaspuolesta, joka rakentuu verkkoselaimeen ladattavista resursseista. Esittelen tässä raportissa ohjelman toteuttamiseen tarvitut tekniikat sekä lopputuloksena syntyneen ohjelman.

2 Perustietoja web-arkkitehtuurista

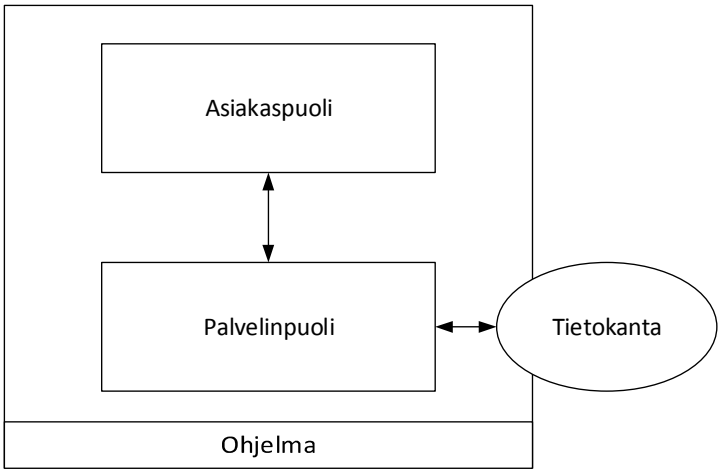
2.1 Asiakas-palvelin-malli

Nimensä mukaisesti asiakas-palvelin-malliin (kuva 2.1) perustuvalla ohjelmalla on kaksi puolta. Asiakaspuolella tarkoitetaan käyttäjän tietokoneelle ladattavaa tai siihen asennettua ohjelman osaa. Palvelinpuoli käsittää ohjelman ydinosan, joka sijaitsee tyypillisesti palvelimeksi nimetyllä tietokoneella. Palvelimen fyysisellä sijainnilla ei ole suuremmin merkitystä, kunhan käyttäjän tietokoneen ja palvelimen välillä on jonkinlainen verkkoyhteys. Asiakas-palvelin-mallissa on siis kyse hajautetusta ohjelmistosta, jossa palvelinpuoli ja asiakaspuoli suorittavat eri tehtäviä. (1; 2.)



Kuva 2.1. Asiakas-palvelin-malli.

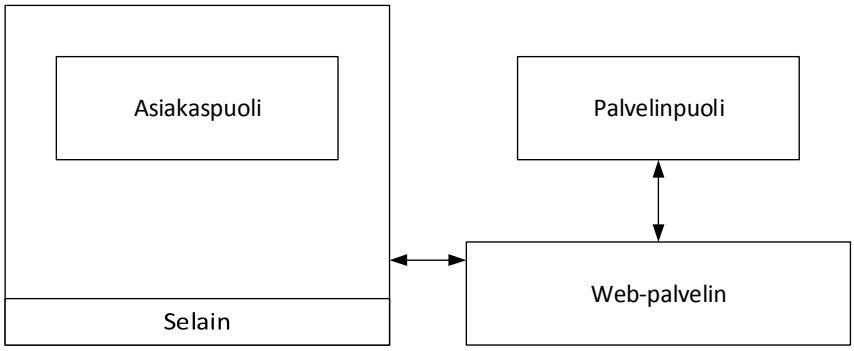
Palvelinpuoli on ohjelman pysyvä osa, johon yleensä liittyy tietokanta, josta ohjelman tietoja haetaan ja jonne niitä tallennetaan (kuva 2.2). Asiakaspuoleen sisältyy ohjelman käyttöliittymä, jonka kautta käyttäjä pääsee käsiksi palvelimella oleviin tietoihin. Mallin perustavia ajatuksia on se, että ohjelmalla voi olla useita samanaikaisia käyttäjiä. (1; 2.)



Kuva 2.2. Tiedonvälitys asiakas-palvelin-mallissa.

2.2 Web-sovellustekniikat

Web-sovellus on verkkoselaimen kautta käytettävä tietokoneohjelma. Kuvassa 2.3 esitetään yksinkertaisen web-sovelluksen osatekijät. Web-sovelluksen asiakaspuoli muodostuu resursseista, jotka ladataan palvelimelta selaimeen. Asiakaspuoli siis toteutetaan selaimen tukemilla tekniikoilla, joista merkittävimmät ovat HTML- ja CSS-kuvauskielet sekä JavaScript-ohjelmointikieli. Palvelinpuoli rakentuu yleensä jostakin web-palvelinohjelmasta sekä sitä hyödyntävästä web-sovelluksen palvelinosasta. Palvelinpuoli on mahdollista toteuttaa millä tahansa ohjelmointikielellä. Tiedonsiirto web-sovelluksen asiakaspuolen ja palvelinpuolen välillä tapahtuu selaimen ja web-palvelinohjelman kautta perinteisesti hypertekstin siirtoprotokollaa (HTTP) käyttäen, mistä kerrotaan tarkemmin luvussa 2.3. Nykyaikaiset selaimet tukevat myös uudempaa HTTP/2-protokollaa, WebSocket-protokollaa sekä WebRTC-teknologiaan liittyviä tiedonsiirtomenetelmiä, mutta näitä ei käsitellä tämän raportin puitteissa.



Kuva 2.3. Yksinkertaisen web-sovelluksen osatekijät.

Suosittuja palvelinpuolen kieliä ovat muun muassa PHP, Java, Ruby, Perl, sekä Windows-palvelimissa C# ja Visual Basic .NET. Node.js-ajoympäristön myötä myös JavaScriptia on voitu käyttää palvelinpuolen toteuttamiseen. Python oli minulle luontevin vaihtoehto aiemmasta sovellusprojektista saamani hyvän kokemuksen perusteella. Web-sovellus rakennetaan usein jonkin web-sovelluskehyksen (engl. web framework) päälle. Python-ohjelmointikieleen perustuvia web-sovelluskehyksiä on lukuisia, joista hyvin suosittuja ovat ainakin Django ja Flask. Kokeilin näistä molempia ja valitsin lopulta Flaskin opinnäytetyön perustaksi. Flask on puolestaan rakennettu Werkzeug-kirjaston päälle (3). Werkzeugiin sisältyy yksinkertainen web-palvelinohjelma, joten kehityskäyttöä varten muuta sellaista ei tarvitse erikseen asentaa (4).

2.3 Hypertekstin siirtoprotokolla

Hypertekstin siirtoprotokolla eli HTTP on merkittävin protokolla, jota käytetään selainten ja web-palvelinten välisessä kommunikoinnissa. Tämän raportin puitteissa HTTP:lla tarkoitetaan HTTP/1.1:tä, eli protokollan versiota 1.1, joka on määritelty Internet Engineering Task Forcen (IETF) RFC-dokumenteissa 7230 - 7235 (5).

Kaiken kaikkiaan HTTP on jokseenkin monimutkainen protokolla, mutta sen perustava idea on kuitenkin yksinkertainen. HTTP perustuu asiakas-palvelin-malliin, jossa asiakasohjelma lähettää pyyntöjä palvelinohjelmalle, ja palvelinohjelma lähettää niihin vastauksia. Kukin pyyntö-vastaus-pari (engl. request/response) muodostaa yhden HTTP-transaktion. HTTP on itsessään tilaton (engl. stateless) protokolla, mikä tarkoittaa sitä, että transaktiot ovat toisis-

taan riippumattomia. Tästä syystä kunkin HTTP-pyynnön täytyy sisältää kaikki ne tiedot, jotka palvelinohjelma tarvitsee pyynnön täyttämiseksi. Niin pyynnöt kuin vastaukset ovat viestejä, jotka voivat aloitusrivin (engl. start line) lisäksi sisältää otsikkorivejä (engl. headers) sekä sisältöosan (engl. body). Sisältö voi olla mitä tahansa dataa. (6.)

2.4 Staattiset resurssit

Staattisella resurssilla tarkoitetaan mitä tahansa tiedostoa, jonka web-palvelin toimittaa selaimelle suoraan sellaisena kuin se on, esimerkiksi kiintolevyllä tai web-palvelimen välimuistissa. Tyypillisiä staattisia resursseja ovat kuvatiedostot, CSS-tiedostot sekä JavaScript-tiedostot.

2.5 Dynaamiset resurssit

Dynaamisella resurssilla tarkoitetaan sisältöä, jonka web-sovellus muodostaa ennen kuin se toimitetaan selaimelle. On tavanomaista, että esimerkiksi HTML-tiedostot ovat dynaamisia resursseja, jotka muodostetaan "lennossa" mallipohjien (engl. templates) ja esimerkiksi tietokannasta haetun tiedon perusteella.

3 Projektin kulku

Sain tiedon asiakkaan tarpeesta alun perin huhtikuun alkupuolella 2015. Tiedustelin sovellukseen liittyviä vaatimuksia alustavasti 8. huhtikuuta pidetyssä Skype-palaverissa. Varsinainen määrittelypalaveri pidettiin Punkaharjun vaneritehtaan toimistolla 11. toukokuuta. Sain vielä saman päivän aikana kopion osaamismatriisista, jonka sovelluksen olisi tarkoitus korvata. Näiden tietojen myötä minun oli mahdollista ryhtyä suunnittelemaan sovelluksen toteuttamista. Palaverissa ei kuitenkaan oltu sovittu tarkemmin projektin aikataulusta; lähinnä minulle jäi käsitys, että sovelluksen demoversiota haluttaisiin kokeilla syksyn aikana. Projekti ei konkreettisesti edistynyt kesän aikana, koska suoritin tuolloin muita puuttuvia opintoja. Koetin kuitenkin kerätä tarvittavaa teknistä tietoa etukäteen, jotta pääsisin aloittaessani mahdollisimman nopeasti liikkeelle.

Käytännössä aloitin työn elokuun puolivälin jälkeen. Ryhdyin aluksi toteuttamaan ohjelman palvelinpuolta Djangoon perustuen, mutta noin kahden viikon kuluttua jouduin toteamaan, että Django on omaan makuuni liian monimutkainen sovelluskehys. Oli selvää, että Djangoa ei ole tarkoitettu pienten sovellusten perustaksi. Aloitin työn alusta Flask-sovelluskehyksen pohjalta. Toteutin vuoroin palvelinpuolta, vuoroin asiakaspuolta. Kaiken kaikkiaan työ edistyi odotettua hitaammin. En ollut aiemmin toteuttanut vuorovaikutteista selainkäyttöliittymää, ja erityisesti siihen liittyvät seikat veivät reilusti aikaa.

Lähetin ensimmäisen tilanneraportin asiakkaalle 5. lokakuuta, jolloin olin saanut jo aika paljon tehdyksi. Seuraavat tilanneraportit lähetin 13. ja 27. lokakuuta. Jälkimmäisessä ilmoitin, että työn jatkaminen ilman palautetta olisi hankalaa. Sovimme demopalaverin päivämääräksi 7. joulukuuta. Sovellus sai palaverissa myönteistä palautetta, mutta tässä yhteydessä esitettiin myös tukku lisätoiveita. Lisäksi heräsi toive siitä, että asiakas saisi kokeilla sovellusta itsekseen. Lähdin siitä eteenpäin toteuttamaan ensimmäistä julkaisuversiota. Toimitin version 1.0 asiakkaalle 25. tammikuuta 2016. Lähetin tämän jälkeen vielä kaksi versiopäivitystä, toisen 8. helmikuuta ja toisen 15. maaliskuuta.

Asiakas ei ole toistaiseksi ottanut sovellusta edes koekäyttöön eikä ole siten voinut myöskään antaa siitä palautetta.

En käyttänyt projektin aikana mitään erityistä ohjelmistokehitysmenetelmää. Alun perin olin ajatellut kokeilla testiohjattua kehittämistä, mutta en syksyllä 2015 kokenut ehtiväni perehtyä siihen kunnolla kaikkien muiden tehtävien ohessa.

4 Ohjelman toteutuksessa käytetyt teknologiat

4.1 Python 3

Python on tulkattava, dynaamisesti tyypitetty yleisohjelmointikieli, jonka ensimmäinen versio julkaistiin helmikuussa 1991. Python 2 julkaistiin alun perin lokakuussa 2000, ja tämän kehityshaaran versiot ovat vieläkin hyvin suosittuja. Ensi kerran joulukuussa 2008 julkaistu Python 3 on kuitenkin se kehityshaara, johon kaikkien Python-ohjelmoijien tulisi vaihtaa viimeistään vuoteen 2020 mennessä. Tätä nykyä Python 3 sisältää paljon sellaisia ominaisuuksia, joita ei löydy Python 2:sta. (7.)

Python on pääasiassa olio-ohjelmointikieli, mutta se tukee myös proseduraalista ohjelmointia. Pythonilla ohjelmointi on helppoa, ja kieli soveltuu hyvin käyttöjärjestelmäriippumattomien ohjelmien toteuttamiseen. Pythonia käytetään paljon myös komentosarjakielenä. Python-ohjelmat eivät kuitenkaan ole erityisen tehokkaita, mikä toki pätee muidenkin tulkattavien, dynaamisten ohjelmointikielten kohdalla.

4.2 SQLite

Valitsin sovelluksen tietokantaratkaisuksi SQLiten. Siinä missä monet muut tietokantaratkaisut ovat erillisiä palvelinohjelmistoja, on SQLite tietokantamoottori, joka liitetään osaksi sovellusta. SQLite sisältyy Python-asennukseen, joten sitä ei tarvitse erikseen asentaa. Pythonin standardikirjastoon kuuluva paketti (engl. package) nimeltä sqlite3 tarjoaa rajapinnan SQLiten hyödyntämiseen. Täten SQLite on helppo ottaa käyttöön Python-sovelluksessa. (8.)

SQLite-tietokanta on oletusarvoisesti yksittäinen tiedosto, joskin tietokantamoottori voi tietokantaa käytettäessä luoda väliaikaisia tiedostoja (9). Tietokantatiedoston voi helposti kopioida tai siirtää tietokoneelta toiselle. Sen voi tarvittaessa myös avata esimerkiksi DB Browser for SQLite -ohjelmalla tai sqlite3-komentorivisovelluksella.

SQLitesta on syytä tietää, että se ei oletusarvoisesti varmista viite-eheyttä. Viite-eheyden varmistamiseksi SQLitelle on annettava alla oleva komento aina, kun uusi yhteys tietokantaan avataan: (10.)

```
PRAGMA foreign_keys = ON;
```

Myös tiedon syöttäminen tietokantaan on oletusarvoisesti verrattain hidasta, koska SQLite pyrkii monin keinoin varmistamaan tietokantatiedoston rakenteellisen eheyden. Tiedon syöttäminen nopeutuu merkittävästi, jos SQLitelle annetaan alla oleva komento, kun yhteys tietokantaan on avattu:

```
PRAGMA synchronous = OFF;
```

Riskinä tässä on se, että tietokantatiedosto voi viottua, mikäli tietokone sammuu tai käyttöjärjestelmä kaatuu kesken kaiken. (10.)

4.3 SQLBuilder

Monen muun tietokantaratkaisun tavoin SQLite-tietokannan tietoja käsitellään SQL-komentokielellä. Esimerkiksi alla oleva SQL-kysely hakee taulusta nimeltä employee tiettyyn tiimiin kuuluvien työntekijöiden tiedot:

```
SELECT id, first_name, last_name
FROM employee
WHERE team_id = 1
ORDER BY last_name, first_name;
```

Python-ohjelmakoodissa vastaava haku voidaan suorittaa vaikkapa seuraavasti:

```
import sqlite3
with sqlite3.connect('app.db') as connection:
   cursor = connection.execute(
       'SELECT id, first_name, last_name ' +
       'FROM employee ' +
       'WHERE team_id = 1 ' +
       'ORDER BY last_name, first_name;'
   )
   print(cursor.fetchall())
   cursor.close()
```

Python-ohjelmakoodissa SQL-kyselyt ovat siis yksinkertaisesti merkkijonoja. Monesti SQL-kyselyt täytyy muodostaa ohjelmallisesti, ja tällöin niiden käsittely merkkijonoina on hieman kankeaa. SQLBuilder-paketin avulla SQL-kyselyt on mahdollista kirjoittaa Python-ohjelmakoodin muodossa, esimerkiksi seuraavasti:

```
from sqlbuilder.smartsql import (Query, Table, Field)
employee_id = Field.id
first_name = Field.first_name
last_name = Field.last_name
team_id = 1
query = Query(Table.employee)\
           .fields([employee_id, first_name, last_name])\
           .where(Field.team_id == team_id)\
           .order_by([last_name, first_name])
```

4.4 Flask

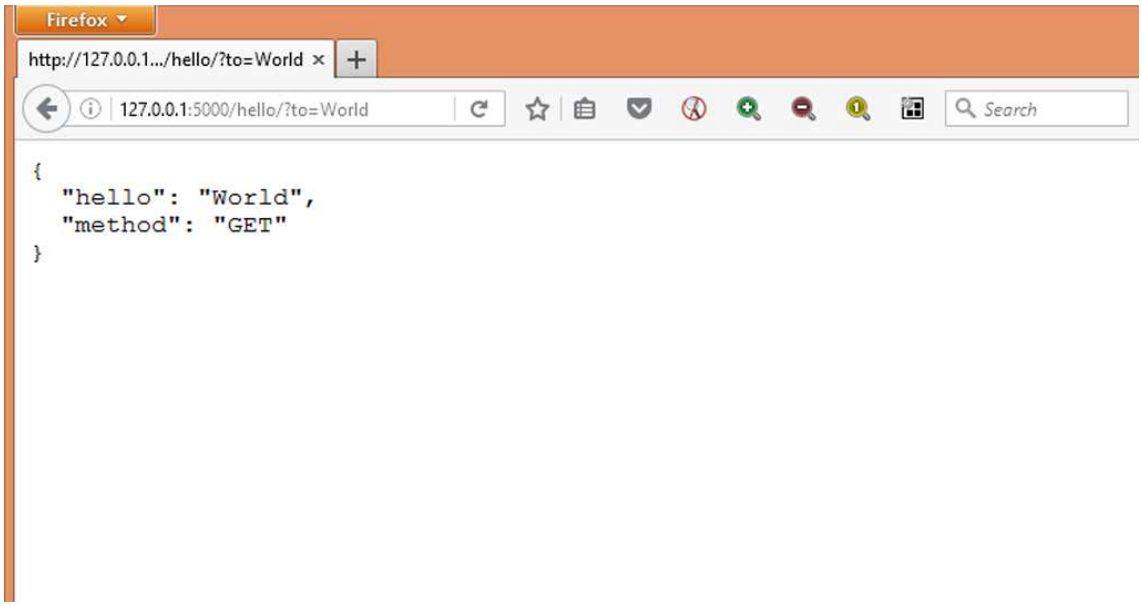
Jotakin laadukasta web-sovelluskehystä hyödyntäen niin yksinkertaisten kuin vähän monimutkaisempienkin web-sovellusten luonti on verrattain helppoa. Flask tarjoaa Python-kehittäjille helppokäyttöiset työkalut palvelinpuolen toteuttamiseen. Alla on koodiesimerkkinä ohjelma, joka antaa kuvassa 4.1 näkyvän vastauksen HTTP-pyyntöön "GET /hello/?to=World":

```
from flask import (Flask, request, jsonify)
app = Flask(__name__)

@app.route("/hello/")
def respond():
           return jsonify(
                       hello=request.args.get("to", ""),
                       method=request.method)

if __name__ == "__main__":
           app.run()
```

Esimerkki osoittaa, kuinka vähällä vaivalla selaimen lähettämään pyyntöön voidaan vastata Flaskia hyödyntämällä. Esimerkissä käytetyt ominaisuudet esitellään Flask-sivuston API-dokumentissa (11).



Kuva 4.1. Flask-sovelluksen lähettämä vastaus.

4.5 Flask-Classy

Flask-Classy (nykyisin Flask-Classful) on Flask-laajennus, jonka avulla web-sovelluksen HTTP-rajapinta, tai osia siitä, voidaan paketoida Python-luokiksi. Mielestäni tämä auttaa pitämään ohjelmakoodin selkeänä.

4.6 Jinja2

Jinja2 on mallipohjamoottori (engl. template engine), joka tulee Flaskin mukana. Siihen sisältyy monipuolinen skriptikieli, jota voidaan hyödyntää mallipohjissa. Opinnäytetyöprojektissa käytin vain harvoja sen ominaisuuksista. Jinja2 auttaa dynaamisten sivujen luonnissa. Sitä kutsutaan funktiolla render_template(), jolle välitetään mallipohjan tiedostonimi sekä halutut muuttujat. Alla on tästä esimerkki:

```
render_template('employee.html', employee=employee)
```

Olettaen, että employee on muuttuja, joka sisältää kentät id, first_name ja last_name, nämä tiedot voidaan tulostaa mallipohjassa vaikkapa seuraavasti:

```
<html>
<body>
{{ employee.last_name }}, {{ employee.first_name }} ({{ employee.id }})
</body>
</html>
```

Yksi Jinja2:n hyödyllisistä ominaisuuksista on perintä. Kun useampi sivu sisältää samoja osia, niin on hyvä ajatus luoda yksi, yhteiset osat sisältävä perusmallipohja (engl. base template), ja periyttää sivujen varsinaiset mallipohjat (engl. child templates) siitä.

Yksinkertaisen perusmallipohjan (base.html) sisältö voi näyttää esimerkiksi seuraavanlaiselta:

```
<html>
<body>
<div id="logo"></div>
{% block content %}
{% endblock %}
</body>
</html>
```

Alla on esimerkki mallipohjasta (employee.html), joka periytyy edellä kuvatusta perusmallipohjasta:

```
{% extends "base.html" %}
{% block content %}
{{ employee.last_name }}, {{ employee.first_name }} ({{ employee.id }})
{% endblock %}
```

Funktion render_template() kutsuminen voisi tässä tapauksessa tuottaa esimerkiksi alla olevan tuloksen:

```
<html>
<body>
<div id="logo"></div>
Koskelainen, Juha (100)
</body>
</html>
```

Toinen Jinja2:n hyödyllinen ominaisuus on makrot. Makro on ikään kuin funktio; se palauttaa palan sisältöä mallipohjaan, josta sitä kutsutaan. Alla olevassa esimerkissä (macros.html) on määritelty imagebutton-niminen makro, ja sitä kutsutaan kahdesti:

```
{% macro imagebutton(name, text=None) %}
<button id="button_{{ name }}" type="button">
   <img src="images/{{ name }}.png">{{ text }}
</button>
{% endmacro %}

{{ imagebutton('ok', 'OK') }}
{{ imagebutton('cancel', 'Peruuta') }}
```

Funktion render_template() kutsuminen tuottaa seuraavanlaisen tuloksen:

```
<button id="button_ok" type="button">
   <img src="images/ok.png">OK
</button>
<button id="button_cancel" type="button">
   <img src="images/cancel.png">Peruuta
</button>
```

Makrot voi kirjoittaa omaan mallipohjatiedostoonsa. Tällöin makron voi liittää toiseen mallipohjaan seuraavasti:

```
{{ from 'macros.html' import imagebutton }}
```

Periaatteessa makro on sitä hyödyllisempi, mitä pidemmän kappaleen sisältöä se palauttaa, kunhan sitä voi hyödyntää useammin kuin kerran mallipohjissa.

Jinja2:ta hyödyntäen on mahdollista luoda monimutkaisia mallipohjatoteutuksia. Liiaksi sitä ei kuitenkaan kannata käyttää, sillä sovelluksen suorituskyky voi kärsiä (12).

4.7 OpenPyXL

Excelillä luodun osaamismatriisin tietojen tuonti sovellukseen on tärkeä toiminto, jota käsitellään luvussa 6.3.1. Se oli helppo toteuttaa OpenPyXL-kirjaston avulla. Seuraava Python-koodiesimerkki esittää yhden tavan, jolla xlsx-tyyppisen tiedoston tietty työsivu voidaan lukea ohjelman muistiin:

```
from openpyxl import load_workbook

wb = load_workbook('osaamismatriisi.xlsx', False)
ws = wb.get_sheet_by_name('Punkaharju koivu')
data = [list(row) for row in ws.iter_rows()]
```

4.8 jQuery

Alkuaan vuonna 2006 julkaistu jQuery on valtavan suosittu ilmainen JavaScript-kirjasto (13). Se tarjoaa web-sovelluskehittäjien käyttöön lukuisia helppokäyttöisiä toimintoja. Historiallisesti jQueryn hyödyntäminen on taannut sen, että sama ohjelmakoodi toimii kaikissa merkittävissä selaimissa. Tässä mielessä jQuery ei enää ole välttämättömyys, koska selainten väliset erot ovat nykyisin vähäisiä.

Tätä nykyä jQueryn vahvuus onkin niissä monen monissa käyttöliittymäkomponenteissa, jotka on toteutettu sen varaan.

4.9 jQuery UI

Monet jQueryn avulla toteutetut käyttöliittymäkomponentit tukevat jQuery UI -teemoja. Teeman avulla eri komponenteille voidaan helposti antaa verrattain yhtenäinen ulkoasu. Teemat ovat jQuery UI:n tärkein ominaisuus, mutta projekti kattaa myös muita enemmän tai vähemmän hyödyllisiä toimintoja komponenttien kehittäjille, sekä joitakin valmiita käyttöliittymäkomponentteja.

4.10 DataTables

Koska sovelluksesta tuli löytyä osaamismatriisia vastaava taulukkonäkymä sekä muita taulukoita, kävin läpi useita vaihtoehtoja etsiessäni sopivaa valmista taulukkokomponenttia. Valitsin lopulta DataTables-nimisen ratkaisun. Se on monipuolisista ominaisuuksistaan huolimatta helppokäyttöinen ja Editor-lisäosaa lukuunottamatta ilmainen komponentti. DataTables vaatii jQueryn toimiakseen.

4.11 Ajax

Verkkoselaimet tukevat monenlaisia eri teknologioita, jotka mahdollistavat monipuolisten web-sovellusten toteuttamisen. Yksi hyvin merkittävä joukko teknologioita tunnetaan käsitteellä Ajax. Ajax mahdollistaa sen, että selaimeen ladatun sivun JavaScript-koodi voi kommunikoida palvelimen kanssa asynkronisesti. Lisäksi sivun sisältöä voidaan päivittää ilman, että koko sivua tarvitsee ladata uudelleen. Ajaxia hyödyntäen sivulle voidaan ladata uusia tietoja tarpeen mukaan, esimerkiksi käyttäjän toiminnan seurauksena tai ajastetusti.

5 Ohjelman toteutuksessa käytetyt työkalut

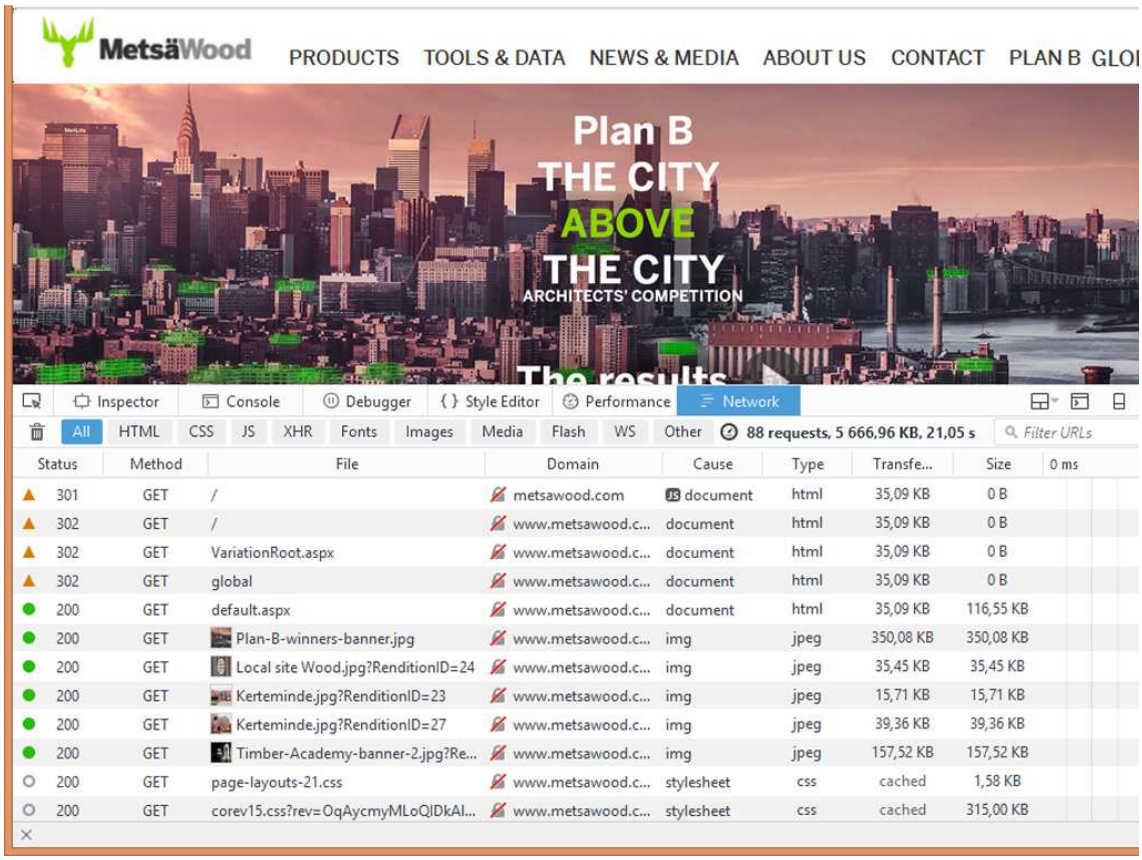
5.1 Mozilla Firefox

Yleisesti ottaen web-sovelluksen tulisi toimia selaimella kuin selaimella. Käytin sovelluksen kehittämisen aikana hyvin pitkälti vain Mozilla Firefoxia. Firefox ei useinkaan edusta aivan kehityksen kärkeä, joten jos sivusto tai sovellus toimii siinä, niin se todennäköisesti toimii ongelmitta myös muissa selaimissa.

Mozilla Firefoxissa kehittäjän työkalut tuodaan näkyviin näppäinyhdistelmällä Ctrl+Shift+I. Työkalupaneeli avautuu oletusarvoisesti selaimen aktiivisen välilehden alaosaan.

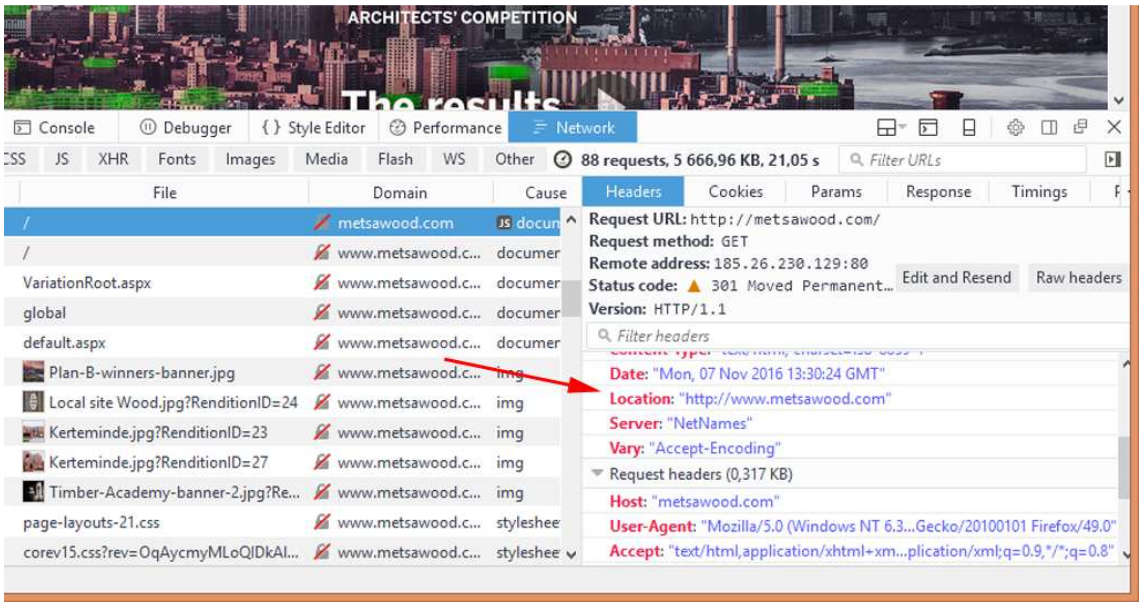
Työkalupaneeli sisältää kuusi pääasiallista välilehteä: Inspector, Console, Debugger, Style Editor, Performance ja Network. Kun Network-välilehti on aktiivisena ja selaimen osoiteriville syötetään esimerkiksi "metsawood.com", niin välilehdelle ilmestyy useita rivejä sisältävä taulukko. Kukin rivi sisältää valittuja tietoja yhdestä HTTP-transaktiosta. Transaktio on tapahtuma, joka käsittää kaksi osaa. Ensimmäinen osa on pyyntö (engl. request), jonka selain lähettää web-palvelimelle. Toinen osa on vastaus (engl. response), jonka web-palvelin lähettää selaimelle. Taulukko siis kertoo, että sivun lataamiseksi suoritetaan useita HTTP-transaktioita.

Taulukon sarakkeet Method ja File kertovat, millaisen pyynnön selain on lähettänyt, ja sarake Domain kertoo, minne pyyntö on lähetetty. Taulukon Status-sarake kertoo, minkä tilakoodin (engl. status code) web-palvelimen lähettämä vastaus sisältää. Kuvan 5.1 esimerkissä ensimmäinen tilakoodi on 301, jolla web-palvelin viestittää, että pyydetty resurssi on siirretty pysyvästi toiseen osoitteeseen. Tämän jälkeen selain on lähettänyt saman "GET /"-pyynnön uuteen osoitteeseen http://www.metsawood.com. Vastauksen tilakoodi on nyt 302, joka edelleen ohjaa selaimen pyytämään haluttua resurssia uudesta osoitteesta. Parin uudelleenohjauksen jälkeen selain saa vastauksen tilakoodilla 200, jolla web-palvelin viestittää, että pyyntö on käsitelty onnistuneesti.



Kuva 5.1. Firefoxin työkalupaneelin Network-välilehti.

Kun jokin riveistä aktivoidaan, niin taulukon vierelle ilmestyy lisätietopaneeli, jossa esitetään transaktion yksityiskohtaiset tiedot (kuva 5.2). Paneeli sisältää seitsemän välilehteä: Headers, Cookies, Params, Response, Timings, Security ja Preview. Kukin pyyntö ja vastaus on viesti, joka muodostuu otsikkotiedoista ja varsinaisesta sisällöstä. Headers-välilehti esittää valitusta transaktiosta niin pyynnön kuin vastauksen otsikkotiedot. Esimerkissä vastauksen otsikkotiedoista löytyy kenttä Location, jonka arvo on "http://www.metsawood.com/". Web-palvelin siis kertoo selaimelle osoitteen, josta haluttu resurssi löytyy, ja selain tekee tämän perusteella uuden pyynnön.



Kuva 5.2. Network-välilehden lisätietopaneeli.

Taulukon sarake Type kertoo, minkä tyyppisen resurssin vastaus sisältää. Nettisivut muodostuvat usein HTML-dokumentin lisäksi CSS-määrityksistä, kuvatiedostoista ja JavaScript-koodista. Esimerkkisivun lataaminen käsittää lukuisia HTTP-transaktioita, joissa selain ensin pyytää ja sitten vastaanottaa web-palvelimelta jonkin resurssin. HTML-tiedoston lisäksi siinä ladataan useita JavaScript-tiedostoja, CSS-tiedostoja sekä erityyppisiä kuvatiedostoja.

Nettisivun lataaminen selaimeen on tiettyjä sääntöjä noudattava prosessi. Kun osoiteriville syötetään esimerkin tyylinen osoite, niin selain pyytää web-palvelimelta ensisijaisesti HTML-dokumenttia. Selain viestittää tämän web-palvelimelle pyynnön otsikkotiedoissa kentällä Accept, jonka arvo alkaa tekstillä "text/html". Kun selain saa sitten vastauksen sisältönä HTML-dokumentin, niin selain lukee sen läpi ja pyytää myös siihen liittyvät resurssit web-palvelimelta. Usein selain tallentaa suuren osan resursseista omaan välimuistiinsa, ja kun sivu ladataan uudestaan, niin siihen liittyvät resurssit voidaan ladata sieltä web-palvelimen sijasta.

5.2 DB Browser for SQLite

DB Browser for SQLite on järjestelmäriippumaton työkaluohjelma, jolla voidaan luoda, tarkastella ja muokata SQLite-tietokantoja. Tietokannan suunnittelu oli projektin ensimmäisiä tehtäviä, ja DB Browser for SQLite oli siinä hyödyksi. Tie-

tokannan taulujen ja hakuindeksien luominen ja muokkaaminen, sekä taulujen sisältämien tietojen tarkastelu ja SQL-kyselyiden kokeileminen on usein helpompaa graafisen käyttöliittymän kautta kuin ohjelmallisesti tai komentorivisovelluksen kautta. Ohjelmalla on myös mahdollista viedä tietokannan tietoja tekstitiedostoon. Näin voidaan esimerkiksi tuottaa SQL-komentosarja, jolla tietokanta voidaan luoda uudelleen ohjelmallisesti.

6 Asiakkaalle luotu ohjelma

6.1 HTTP API

Sovelluksen palvelinpuoli tarjoaa määrätyn rajapinnan (HTTP API:n) tietokannan sisältämien tietojen käsittelyyn. Rajapinta koostuu lukuisista eri poluista, joihin asiakaspuoli voi tehdä HTTP-pyyntöjä. Voidaan ajatella, että palvelinpuoli muuntaa nämä pyynnöt tietokantakyselyiksi. Vastaavasti kyselyiden tulokset muunnetaan JSON-muotoisiksi vastausviesteiksi.

Rajapinnan kautta voidaan hakea, muokata, lisätä ja poistaa tietoja – tietenkin sisään kirjautuneen käyttäjän käyttöoikeuksien mukaisesti. Tietojen hakemiseen käytetään HTTP:n GET-pyyntöä, lisäämiseen POST-pyyntöä, poistamiseen DELETE-pyyntöä, ja muokkaamiseen tapauksesta riippuen joko PUT- tai PATCH-pyyntöä. Kaikki rajapinnan polut sisältävät /api-etuliitteen, joka siis erottaa rajapintaresurssit sovelluksen muista resursseista. Polkuja ovat esimerkiksi /api/plants/, /api/departments/ ja /api/employees/. Eri polut tukevat kuhunkin tapaukseen sopivia toimintoja. Yhteistä kaikille poluille on se, että ne tuottavat vastauksena JSON-taulukon, myös virhetilanteissa. Taulukon sisältö taas vaihtelee tilanteen mukaan.

Olettaen että sovellukseen on kirjauduttu sisään, pyyntö "GET /api/plants/" palauttaa taulukon niistä tehtaista, joihin käyttäjällä on vähintään lukuoikeus. Kullakin tehtaalla on tunniste (id) sekä nimi (description). Tietyn tehtaan osastot haetaan pyynnöllä "GET /api/departments/?plant_id=<id>", jossa <id> on tehtaan tunniste. Tässä tapauksessa plant_id on pakollinen argumentti. Jos rajapintaan tehdään pyyntö "GET /api/departments/", niin se tuottaa vastauksena HTTP-tilakoodin "400 BAD REQUEST" ja tyhjän taulukon.

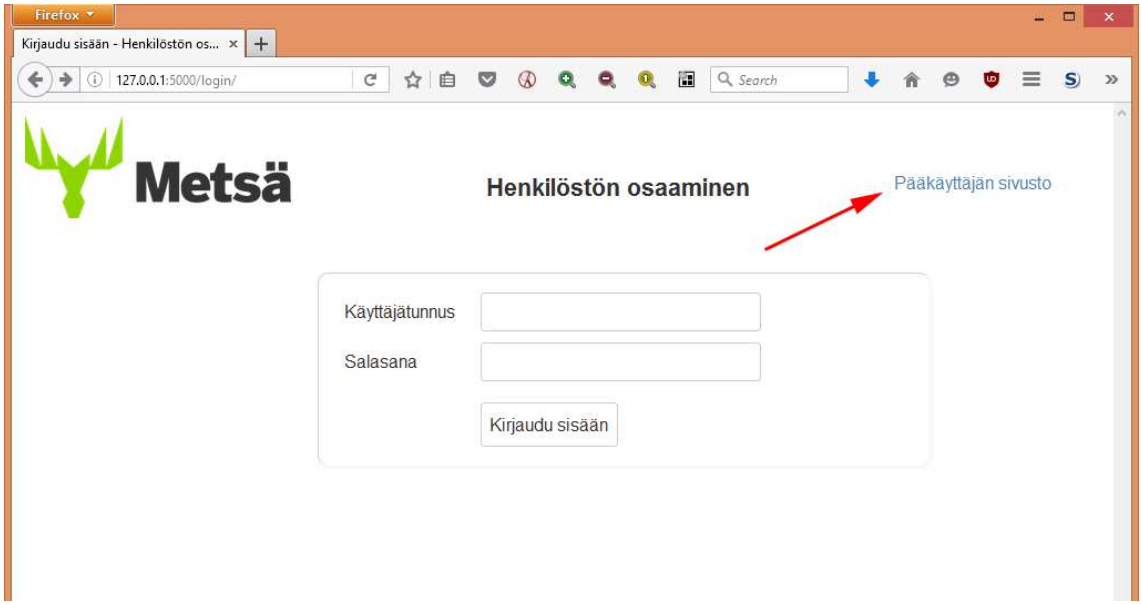
Tehtaan lisääminen tietokantaan tapahtuu pyynnöllä "POST /api/plants/". Tehtaan nimi välitetään tällöin description-nimisessä lomakemuuttujassa. Tehtaan voi lisätä vain pääkäyttäjän oikeuksilla. Kun lisääminen on onnistunut, niin rajapinta palauttaa HTTP-tilakoodin "201 CREATED" sekä taulukon, josta voidaan lukea tietokannan tehtaalle antama tunniste.

Yksittäisen tehtaan voi poistaa pyynnöllä "DELETE /api/plants/<id>". Tehtaan poistaminen myös poistaa kaikki siihen liittyvät tiedot tietokannasta. Poistaminen vaatii pääkäyttäjän oikeudet.

Tehtaan nimen muuttaminen onnistuu pyynnöllä "PUT /api/plants/<id>". Uusi nimi tulee välittää description-nimisessä lomakemuuttujassa. Tämäkin toiminto vaatii pääkäyttäjän oikeudet. Jos pääkäyttäjän oikeuksia ei ole, niin rajapinta palauttaa HTTP-tilakoodin "403 FORBIDDEN" ja tyhjän taulukon.

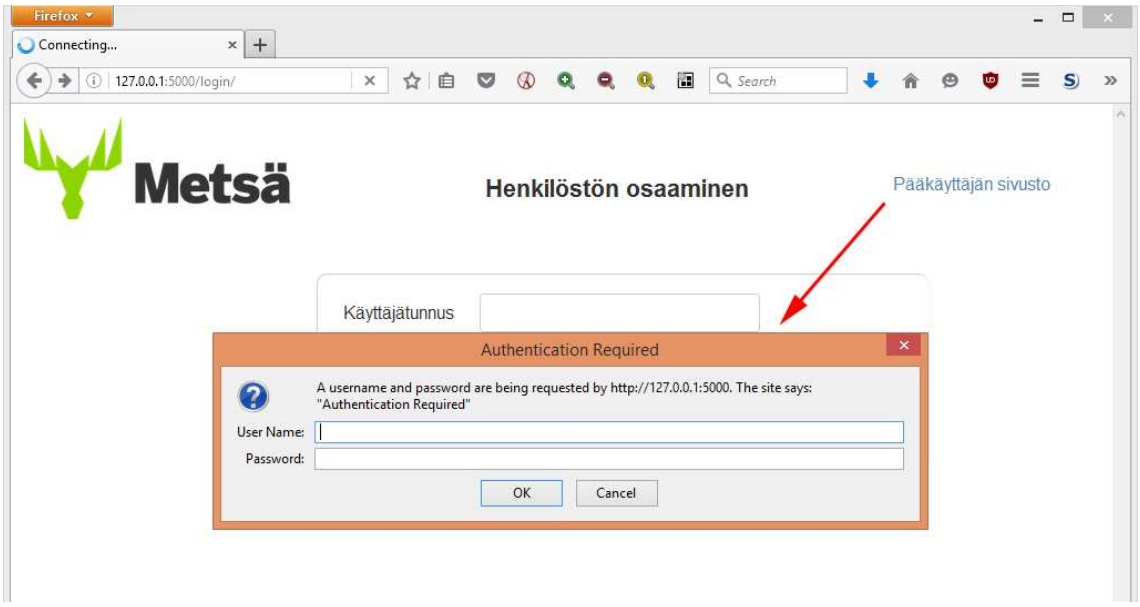
6.2 Sisäänkirjautuminen

Kun sovellus avataan selaimessa ensimmäistä kertaa, niin käyttäjälle esitetään sisäänkirjautumissivu (kuva 6.1). Sivulla on linkki pääkäyttäjän sivustoon, ja pääkäyttäjän tulee kirjautua sisään sen kautta.



Kuva 6.1. Sisäänkirjautumissivu.

Pääkäyttäjän sivusto -linkistä avautuu selaimen ponnahdusikkuna, johon syötetään käyttäjätunnus ja salasana (kuva 6.2). Pääkäyttäjän sivusto koostuu seitsemästä sivusta, joiden kautta hallitaan sovelluksen tietoja.

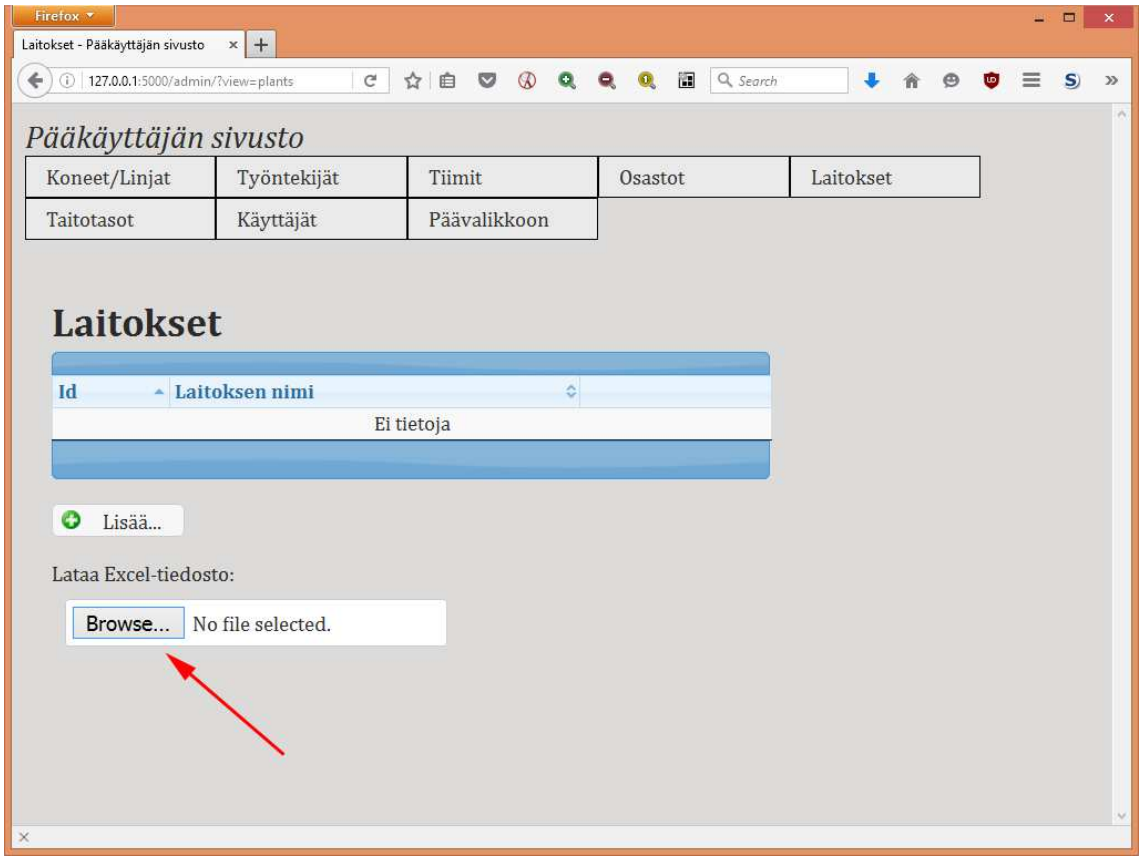


Kuva 6.2. Pääkäyttäjän sisäänkirjautuminen.

6.3 Pääkäyttäjän sivusto

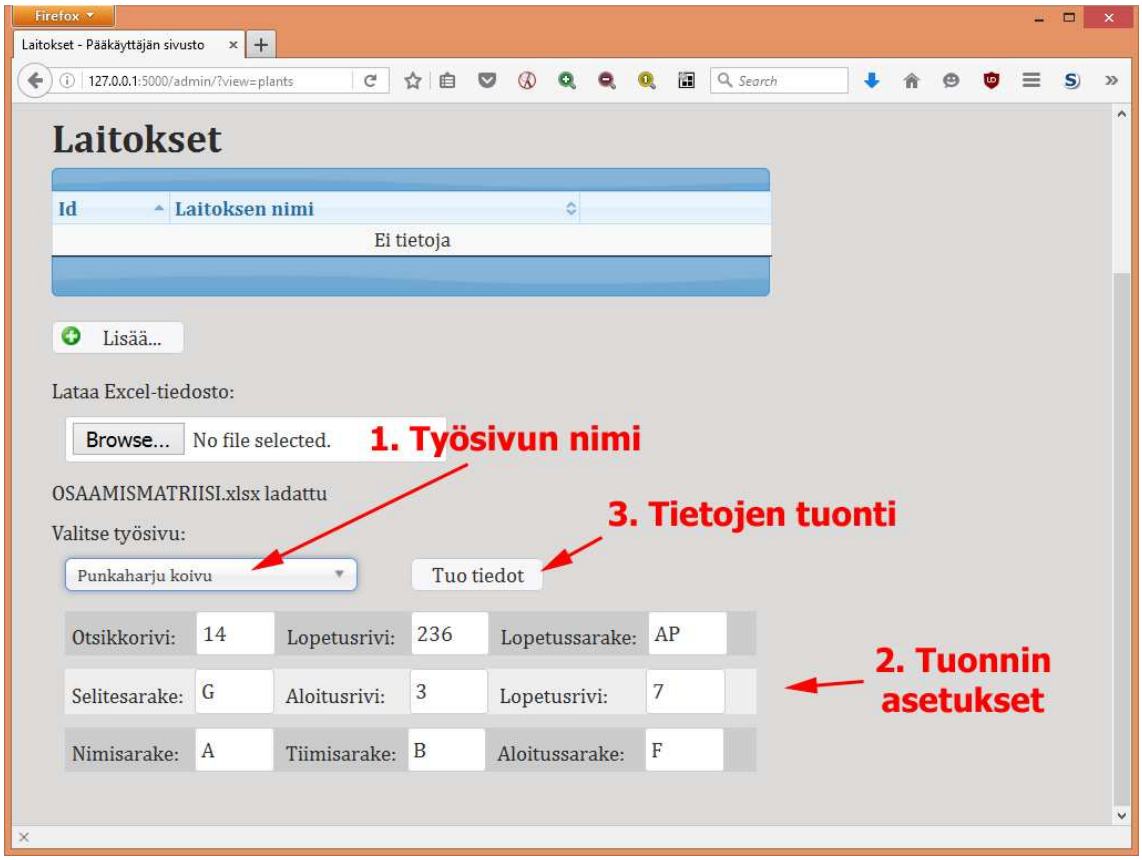
6.3.1 Osaamismatriisin tuonti

Pääkäyttäjä voi tuoda osaamismatriisin tiedot sovellukseen Laitokset-sivulta tekstin "Lataa Excel-tiedosto” alta löytyvällä toiminnolla. Kuvassa 6.3 näkyvästä Browse-painikkeesta avautuu selaimen tiedostonvalintaikkuna, ja kun pääkäyttäjä on valinnut tiedoston, niin se ladataan automaattisesti palvelimelle.



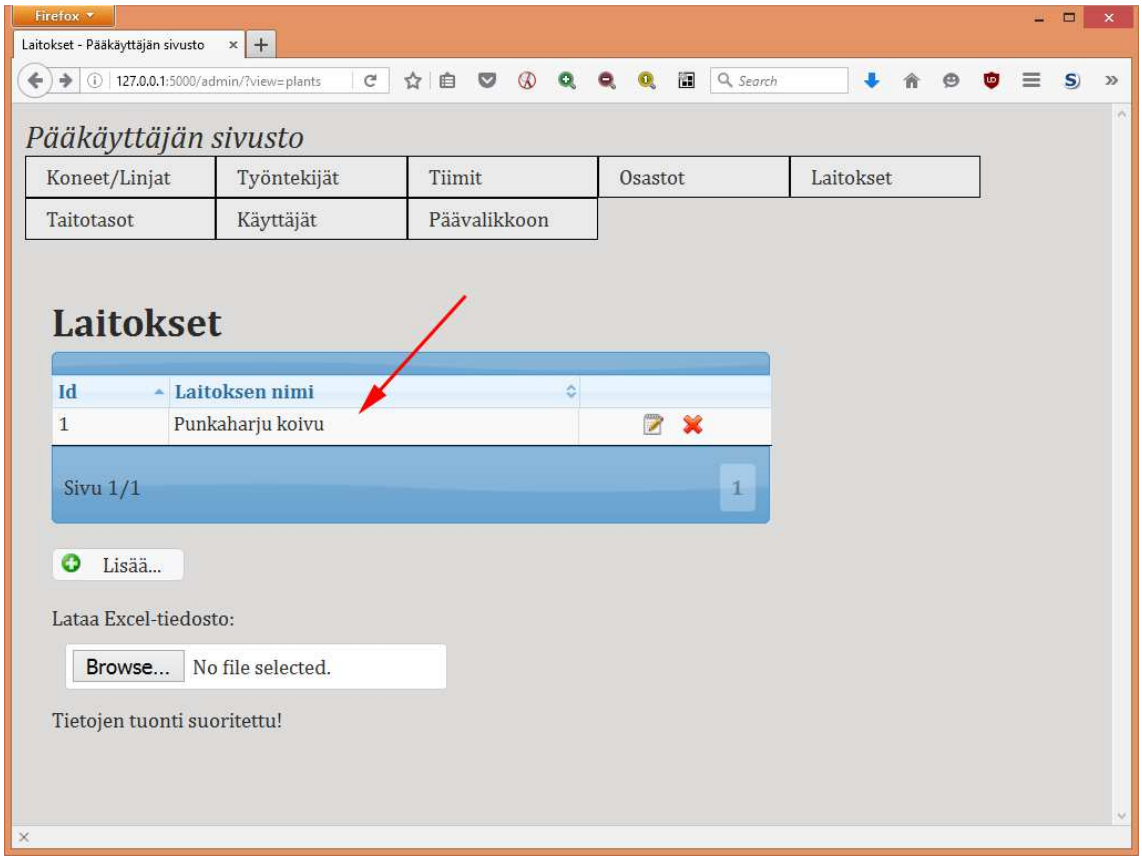
Kuva 6.3. Excel-tiedoston lataus.

Kun tiedosto on ladattu, niin sivulle tulee näkyviin tuonnin käyttöliittymä (kuva 6.4). Pudotusvalikosta tulee valita Excel-taulukon työsivu, jolta tiedot tuodaan. Tässä oletetaan, että työsivun nimi on laitoksen nimi. Pääkäyttäjän tulee myös tarkistaa, että tuonnin asetukset ovat kohdallaan. Tuo tiedot -painikkeesta sovellus koettaa tuoda valitut tiedot



Kuva 6.4. Tietojen tuonti.

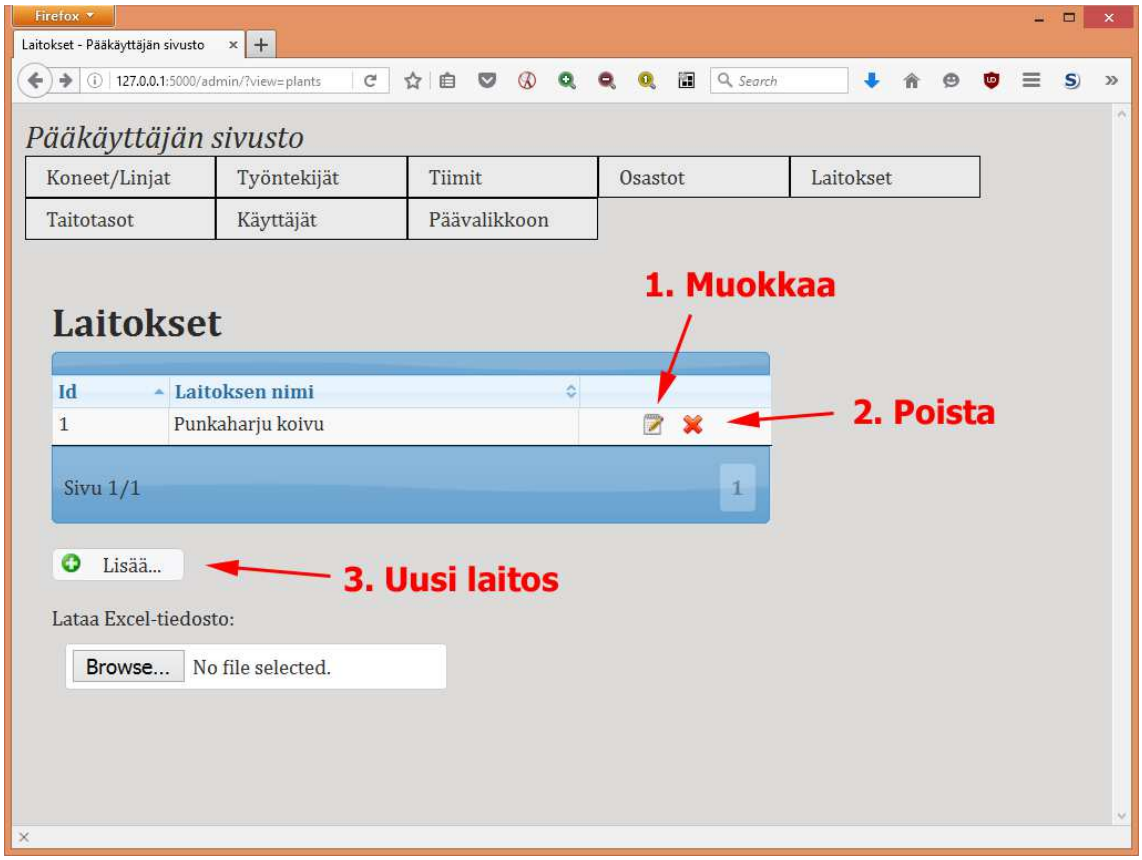
Jos tuontioperaatio onnistuu, niin laitos ilmestyy sivulla näkyvään taulukkoon (kuva 6.5).



Kuva 6.5. Pääkäyttäjän lisäämä laitos.

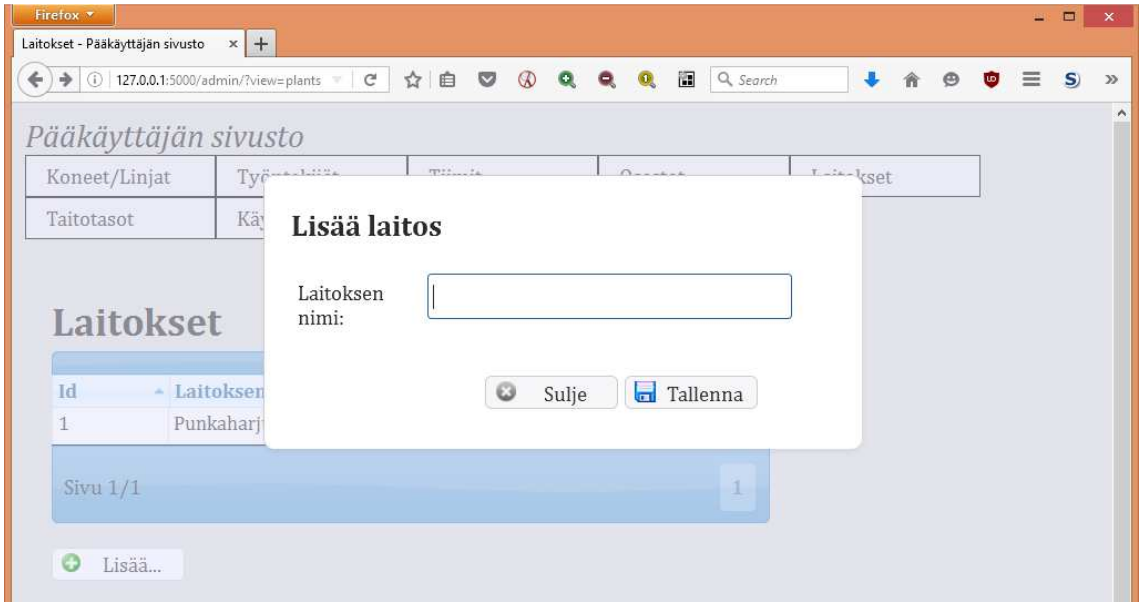
6.3.2 Laitokset-sivu

Kuvaan 6.6 on merkitty laitokset-sivun muut toiminnot. Pääkäyttäjä voi tarvittaessa nimetä laitoksen uudelleen taulukosta löytyvän Muokkaa-toiminnon kautta. Taulukosta löytyy myös Poista-toiminto, jolla voidaan poistaa laitos ja samalla kaikki siihen liittyvät tiedot. Lisää-painiketta napsauttamalla on mahdollista lisätä sovellukseen uusi laitos, mutta tällöin kaikki laitokseen liittyvät tiedot on vastaavasti syötettävä käsin.



Kuva 6.6. Laitokset-sivun toiminnot.

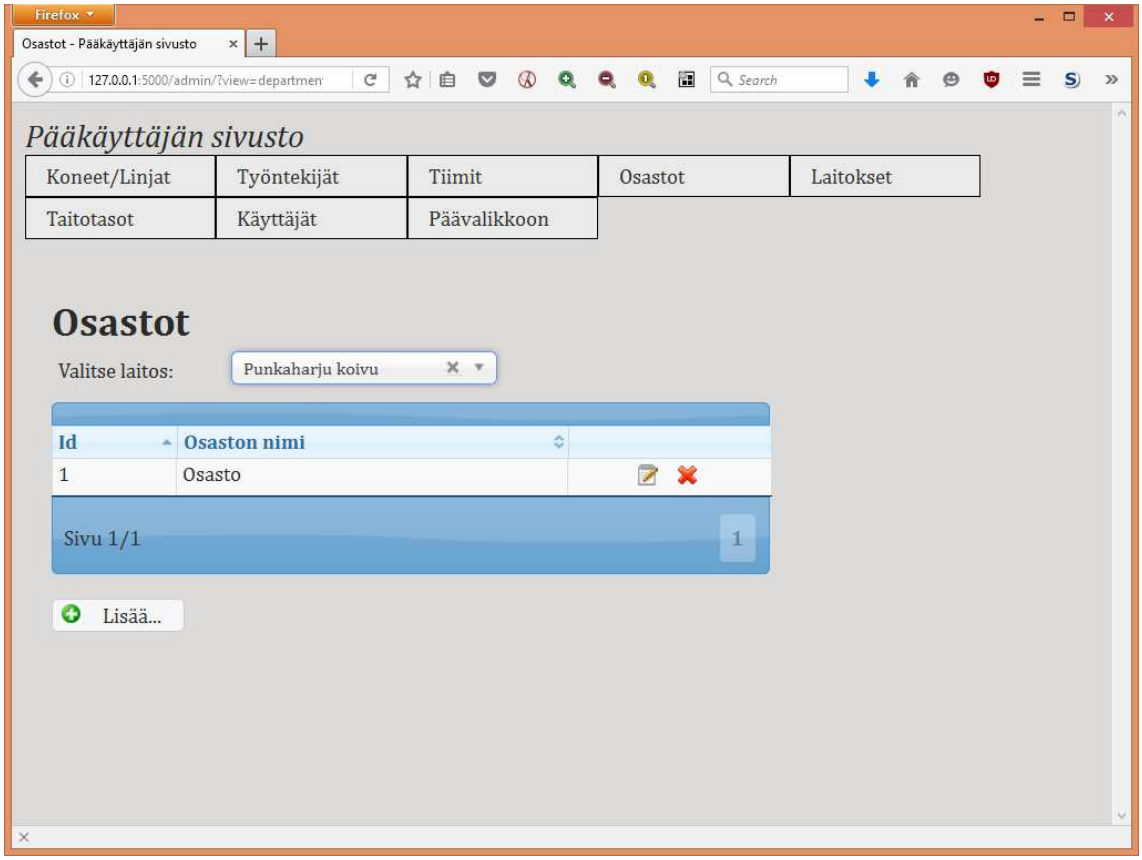
Aluksi lisää-toiminnon kautta luodulle laitokselle annetaan vain nimi (kuva 6.7).



Kuva 6.7. Laitoksen lisääminen.

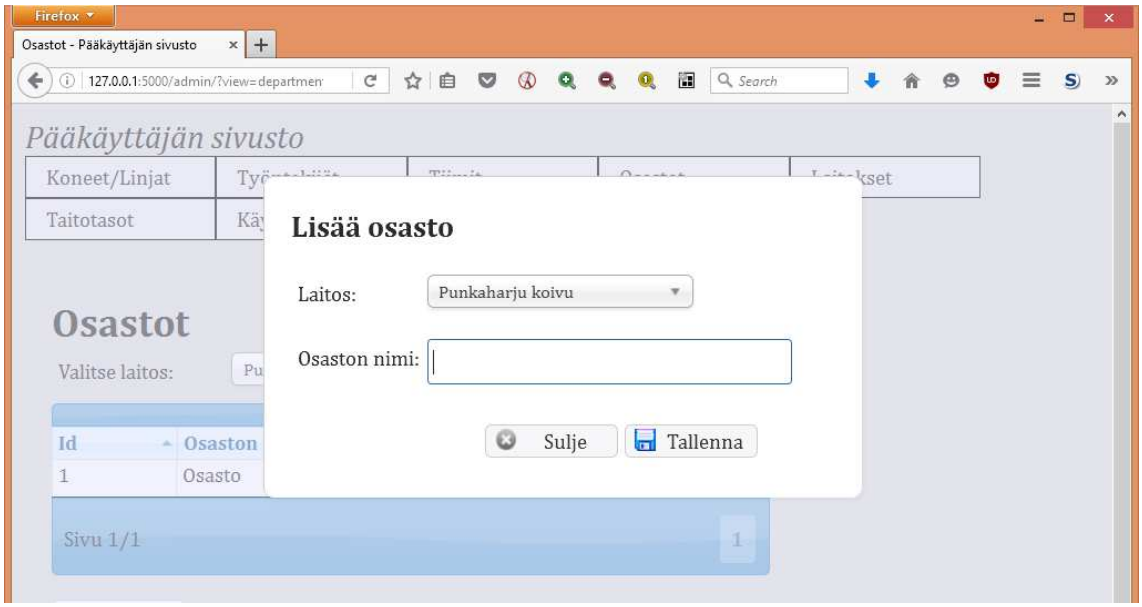
6.3.3 Osastot-sivu

Kun osaamismatriisi tuodaan sovellukseen, niin sovellus lisää laitokseen osaston, jonka nimi on yksinkertaisesti Osasto. Tämä johtuu siitä, että osaamismatriisissa ei erikseen määritellä osastoja. Pääkäyttäjä voi kuitenkin tarvittaessa lisätä laitoksiin uusia osastoja kuvassa 6.8 näkyvän Osastot-sivun kautta. Osaston voi myös nimetä uudelleen. Osaston poistaminen poistaa kaikki siihen liittyvät tiedot.



Kuva 6.8. Osastot-sivu.

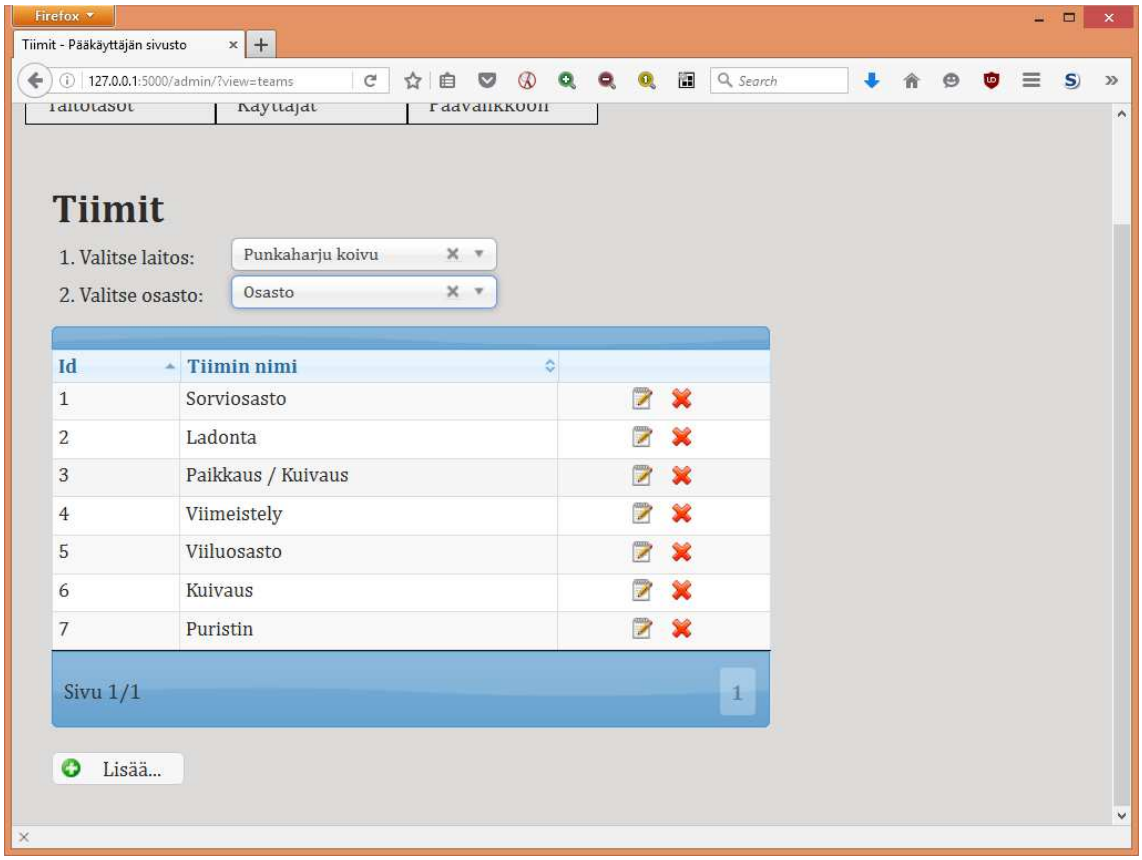
Osasto kuuluu aina johonkin laitokseen. Uutta osastoa lisättäessä osastolle annetaan nimi, ja laitoksen voi vaihtaa pudotusvalikosta (kuva 6.9).



Kuva 6.9. Osaston lisääminen.

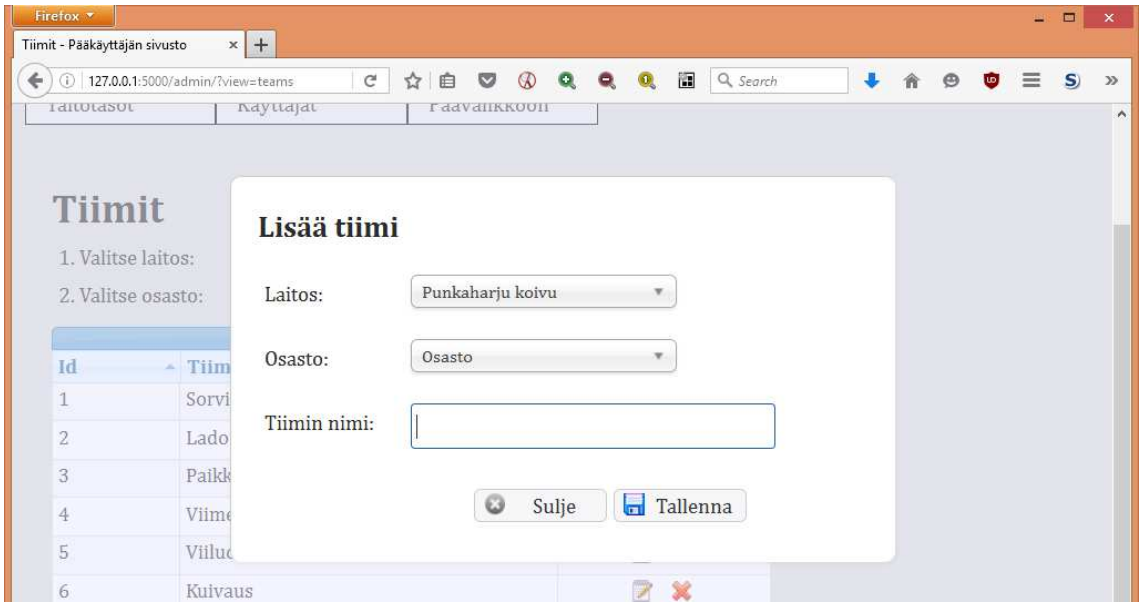
6.3.4 Tiimit-sivu

Tiimi liittyy aina johonkin osastoon. Tiimit-sivun kautta pääkäyttäjä voi tarkastella, muokata, lisätä ja poistaa tiimejä (kuva 6.10).



Kuva 6.10. Tiimit-sivu.

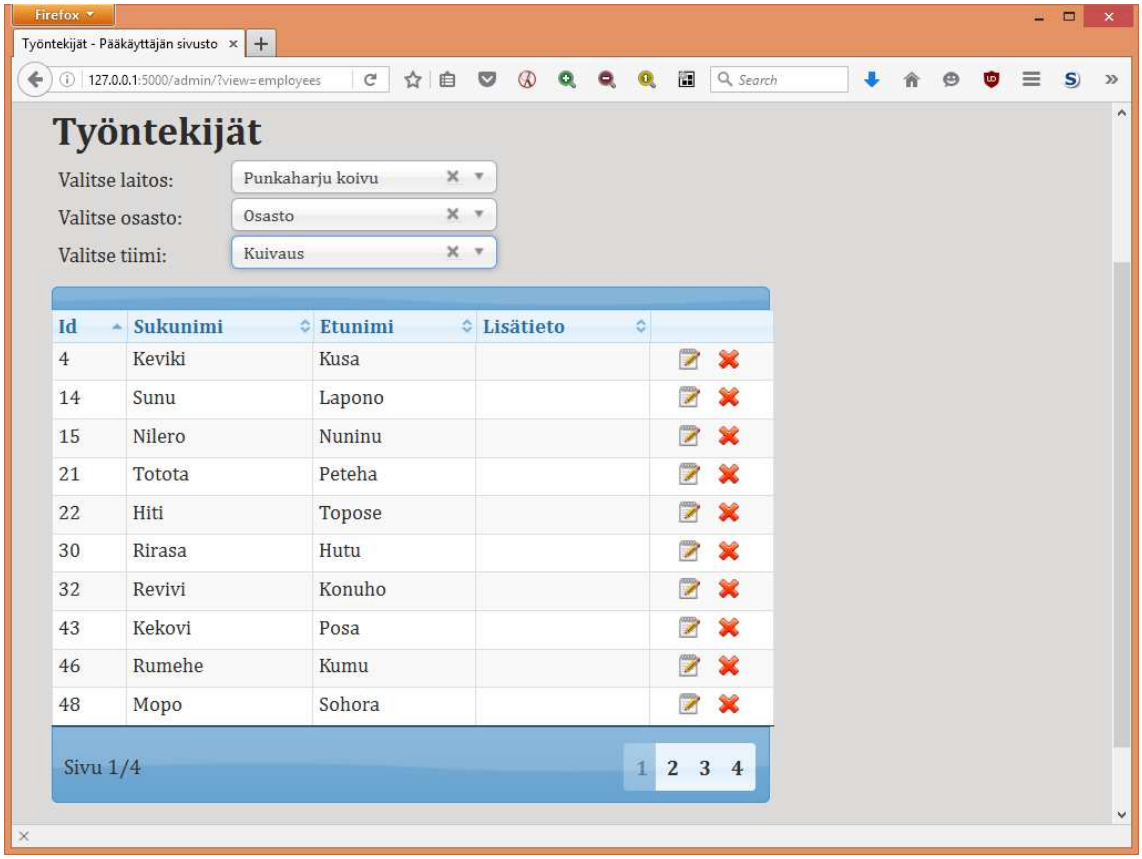
Uutta tiimiä lisättäessä tiimille annetaan nimi, ja sekä laitoksen että osaston voi vaihtaa pudotusvalikosta (kuva 6.11).



Kuva 6.11. Tiimin lisääminen.

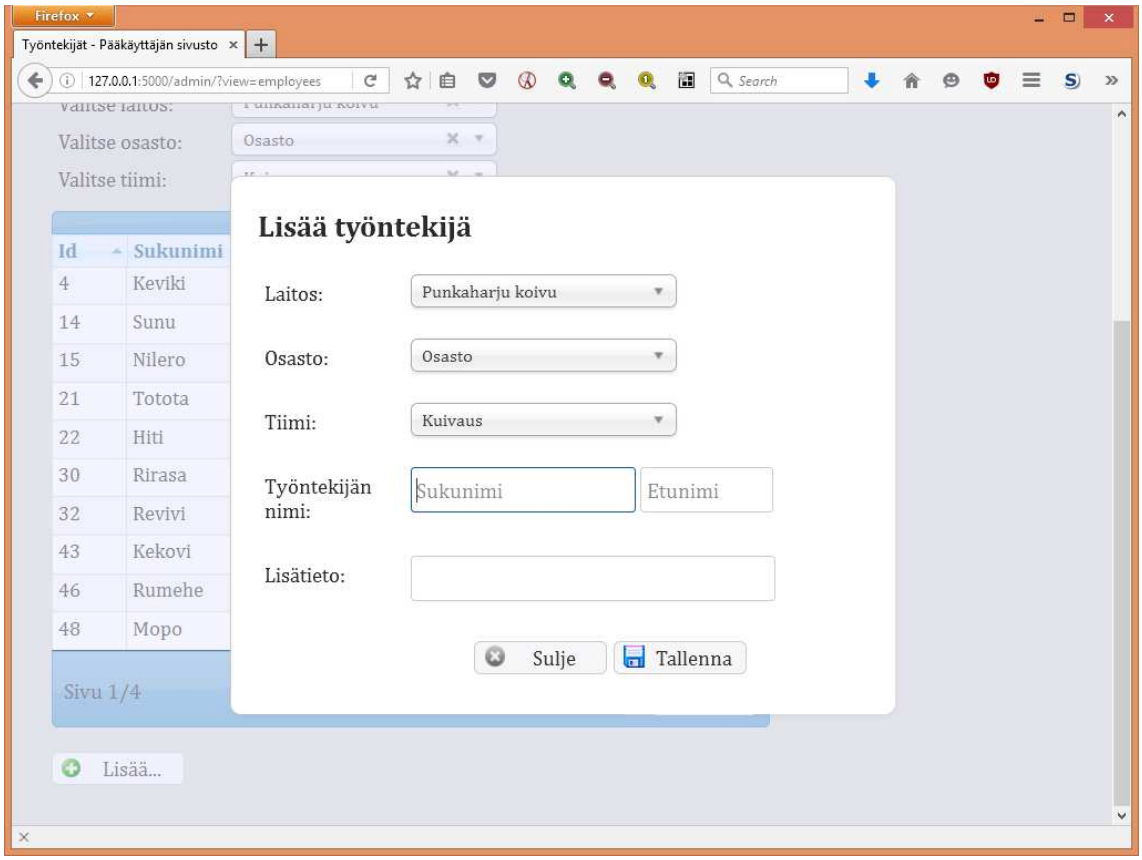
6.3.5 Työntekijät-sivu

Työntekijä kuuluu aina johonkin tiimiin. Työntekijät-sivun kautta pääkäyttäjä voi tarkastella, muokata, lisätä ja poistaa työntekijöitä (kuva 6.12). Pääkäyttäjän sivustolta ei kuitenkaan voi nähdä saati muokata työntekijän osaamiseen liittyviä tietoja.



Kuva 6.12. Työntekijät-sivu.

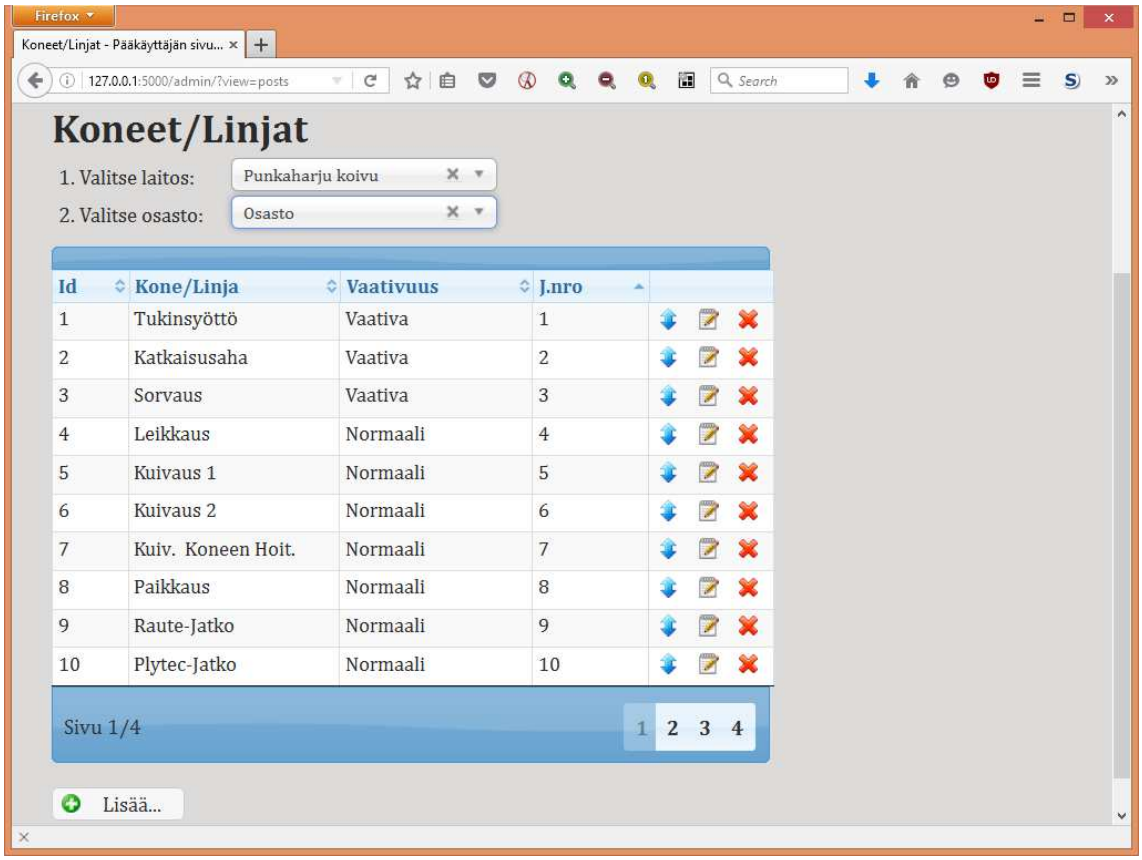
Kuvassa 6.13 on dialogi, joka avautuu sivun lisää-painikkeesta. Uudelle työntekijälle syötetään sukunimi, etunimi, ja mahdollisesti jokin lisätieto. Myös tiimin, osaston ja laitoksen, joihin työntekijä kuuluu, voi vaihtaa pudotusvalikoiden kautta.



Kuva 6.13. Työntekijän lisääminen.

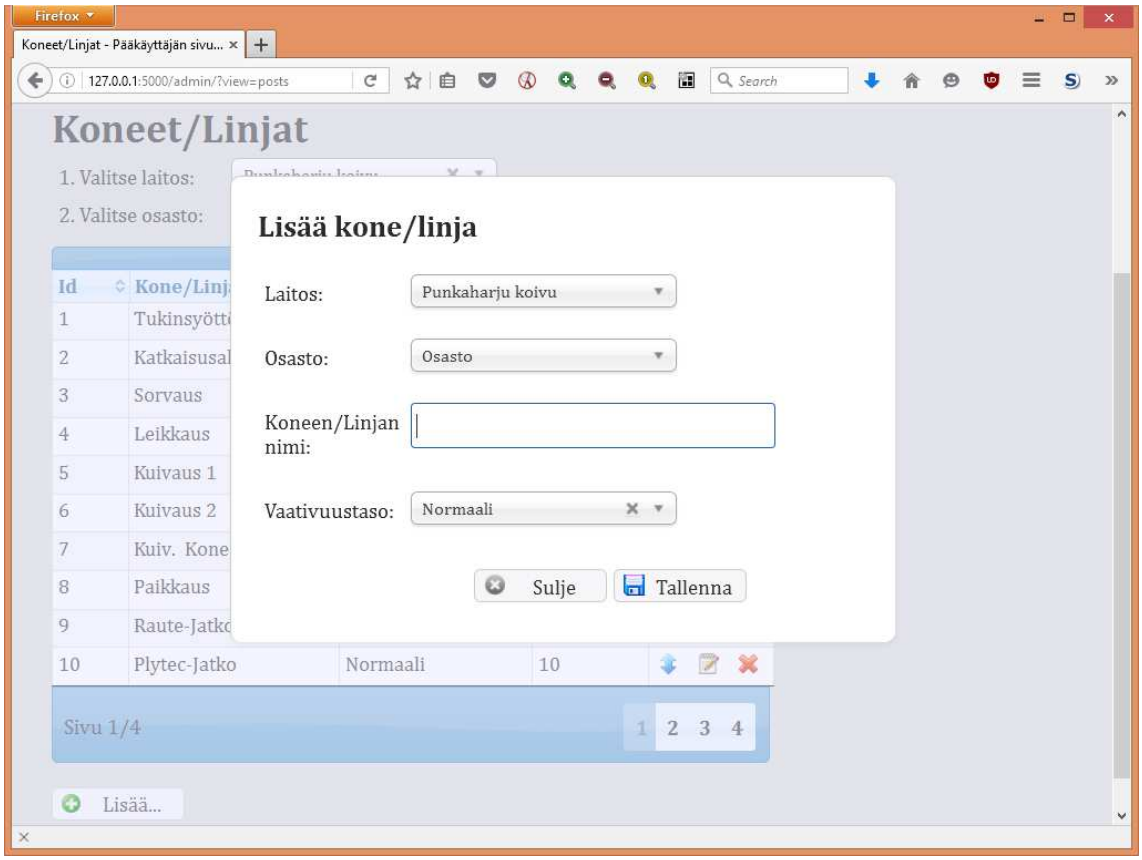
6.3.6 Koneet/linjat-sivu

Kuten tiimi, yksittäinen kone tai linja liittyy ainakin johonkin osastoon. Kuvassa 6.14 esiintyvän Koneet/Linjat-sivun kautta pääkäyttäjä voi tarkastella, muokata, lisätä ja poistaa koneita ja linjoja. Kullakin koneella ja linjalla on nimi, vaativuustaso, sekä järjestysnumero, joka määrittää sen paikan osastonäkymän taulukossa (luku 6.6).



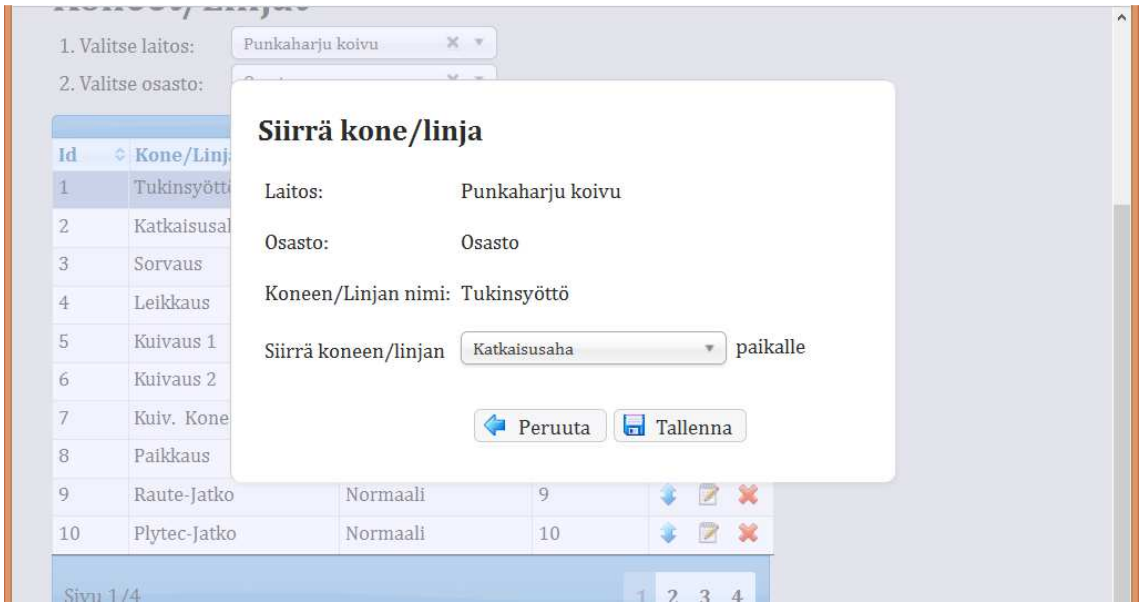
Kuva 6.14. Koneet/linjat-sivu.

Pääkäyttäjä voi lisätä uuden koneen tai linjan klikkaamalla Lisää-painiketta, jolloin selaimeen avautuu kuvassa 6.15 näkyvä dialogi. Koneelle tai linjalle syötetään nimi, ja sille valitaan vaativuustaso. Lisäksi laitoksen ja osaston voi tarvittaessa vaihtaa.



Kuva 6.15. Koneen tai linjan lisääminen.

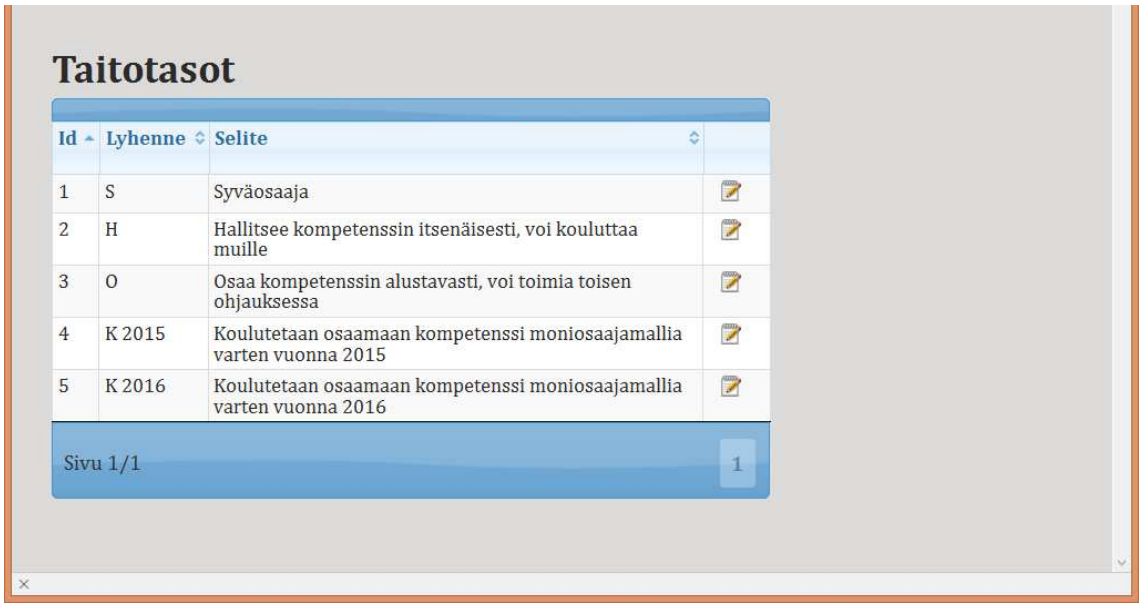
Uusi kone tai linja saa automaattisesti sillä hetkellä suurimman järjestysnumeron. Järjestysnumeroa ei voi muokata suoraan, mutta muokkaa- ja poista-toimintojen lisäksi Koneet/Linjat-sivun taulukossa on siirrä-toiminto. Sitä klikkaamalla avautuu kuvan 6.16 mukainen dialogi, jolla valitun koneen tai linjan voi siirtää jonkin toisen paikalle. Järjestysnumerot päivittyvät automaattisesti siirtämisen myötä.



Kuva 6.16. Koneen tai linjan siirtäminen.

6.3.7 Taitotasot-sivu

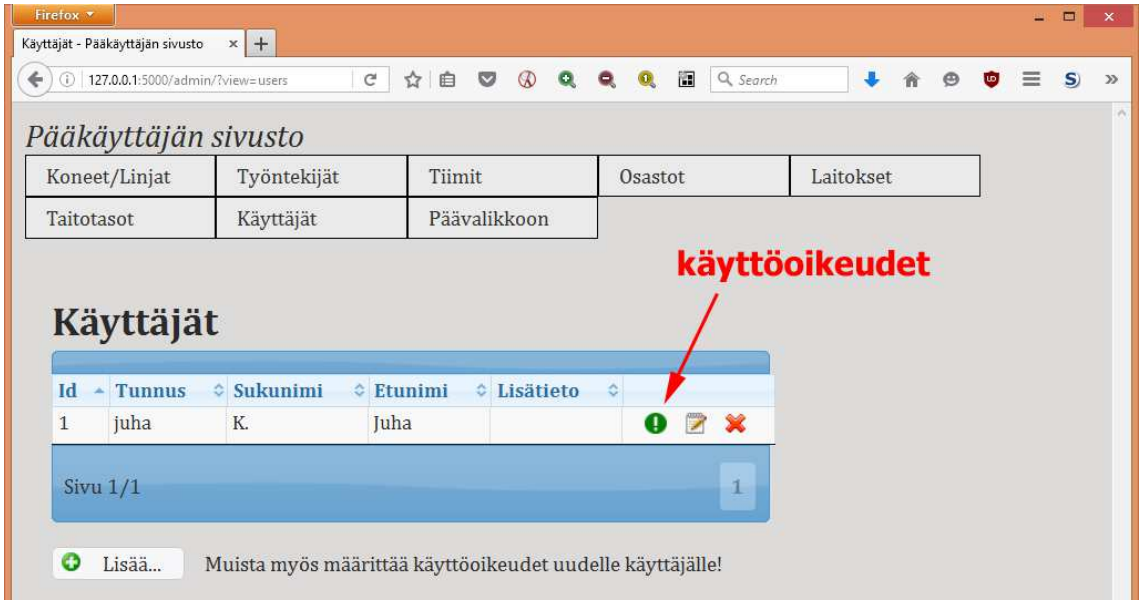
Taitotasot-sivulla esitetään taitotasoja vastaavat lyhenteet ja niiden selitteet (kuva 6.17). Tämän sivun toteutus jäi kesken, mistä johtuen taitotasoja ei voi muokata sovelluksen kautta.



Kuva 6.17. Taitotasot-sivu.

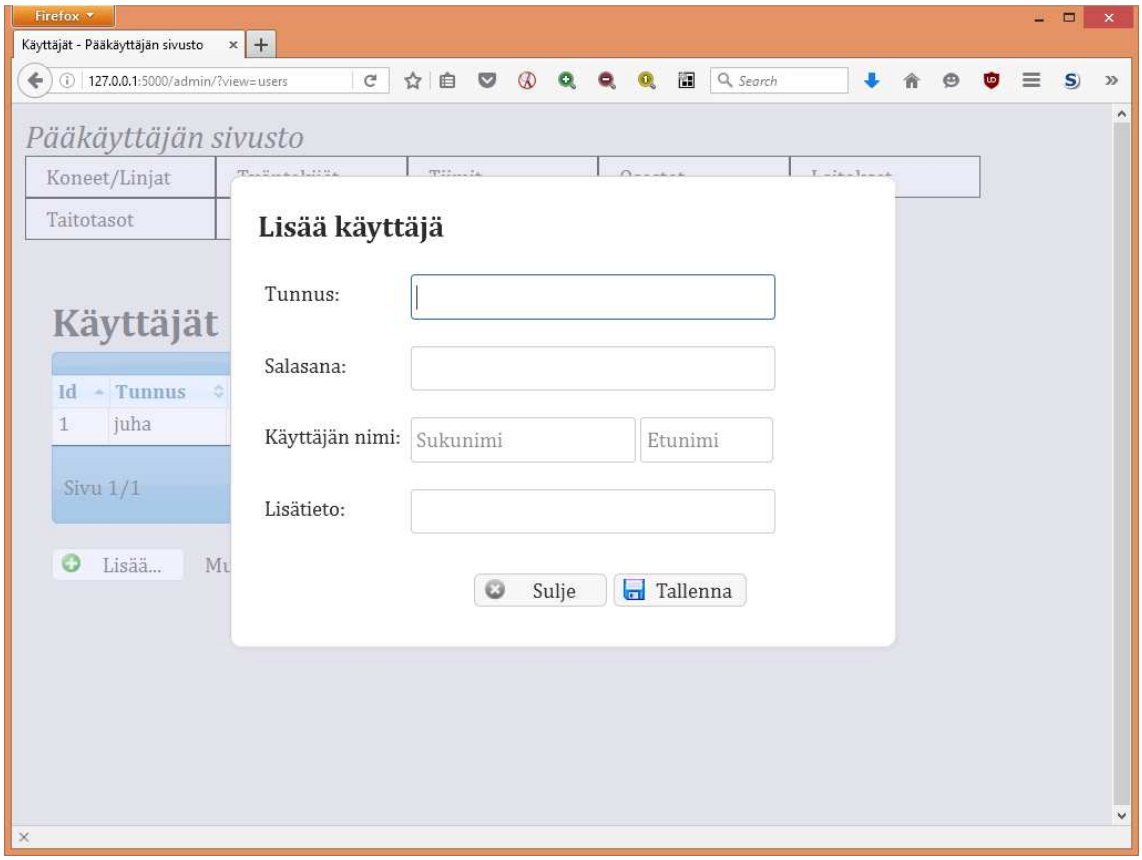
6.3.8 Käyttäjät-sivu

Käyttäjät-sivulta käsin pääkäyttäjä voi hallita sovelluksen käyttäjiä ja heidän käyttöoikeuksiaan (kuva 6.18).



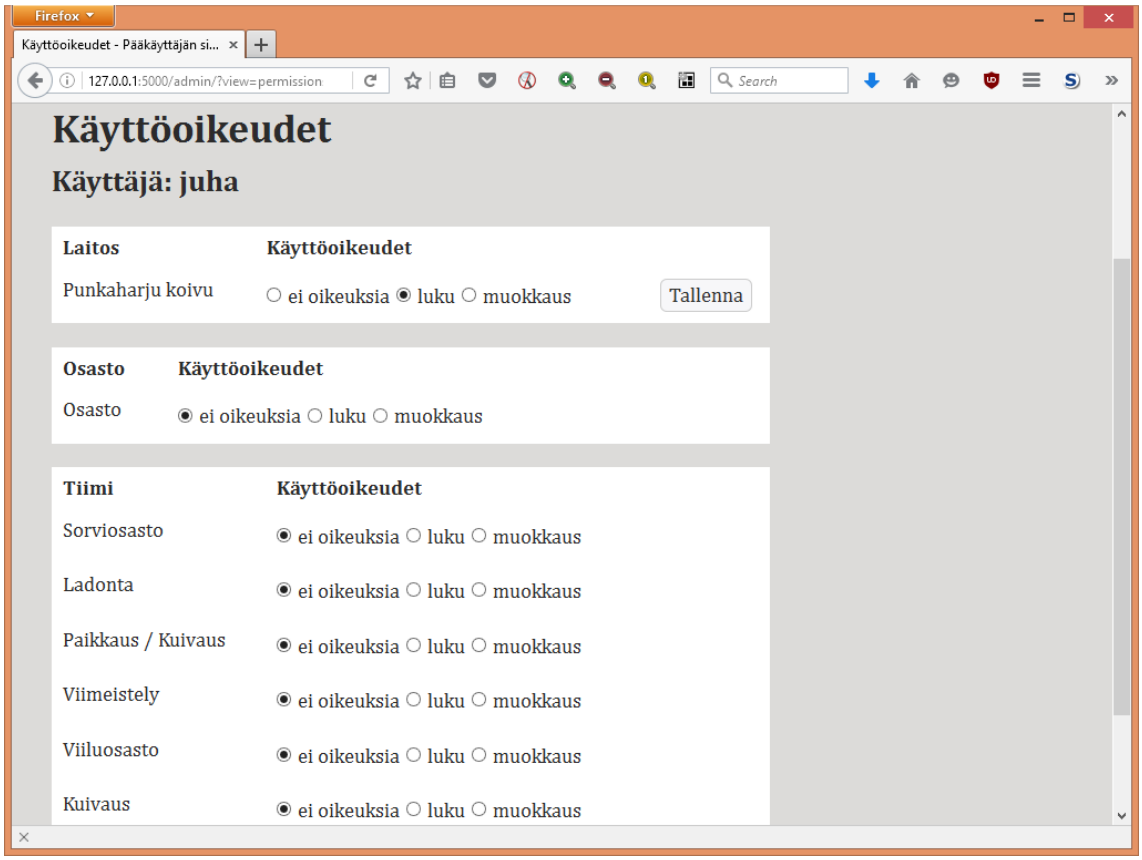
Kuva 6.18. Käyttäjät-sivu.

Lisää-painiketta napsauttamalla voidaan lisätä uusi käyttäjä (kuva 6.19). Käyttäjän tietoja ovat käyttäjätunnus, salasana, nimi ja mahdollinen lisätieto. Oletusarvoisesti uudella käyttäjällä ei ole minkäänlaisia käyttöoikeuksia. Tällainen käyttäjä voi kirjautua sisään sovellukseen, mutta ei voi käytännössä tehdä mitään.



Kuva 6.19. Käyttäjän lisääminen.

Pääkäyttäjä voi myöntää luku- ja muokkausoikeuksia yksittäiselle käyttäjälle laitos-, osasto- ja tiimitasolla, mikä esitetään kuvassa 6.20. Käyttöoikeudet ovat riippumattomia toisistaan, eli ne eivät "periydy" laitostasolta osastotasolle tai osastotasolta tiimitasolle.

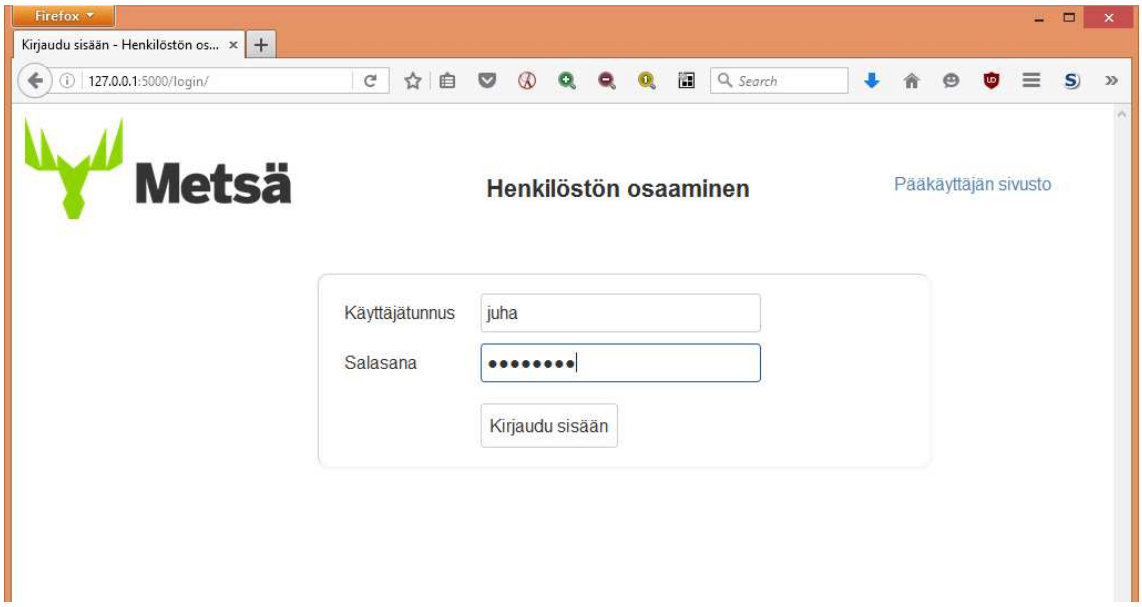


Kuva 6.20. Käyttöoikeusnäkymä.

Pääkäyttäjän sivustolta on myös linkki sovelluksen päävalikkoon. Pääkäyttäjä voi käyttää sovellusta kuten tavallinen käyttäjä, jolle on myönnetty täydet käyttöoikeudet. Ainoa ero on, että pääkäyttäjän salasanaa ei voi muuttaa sovelluksen käyttöliittymän kautta.

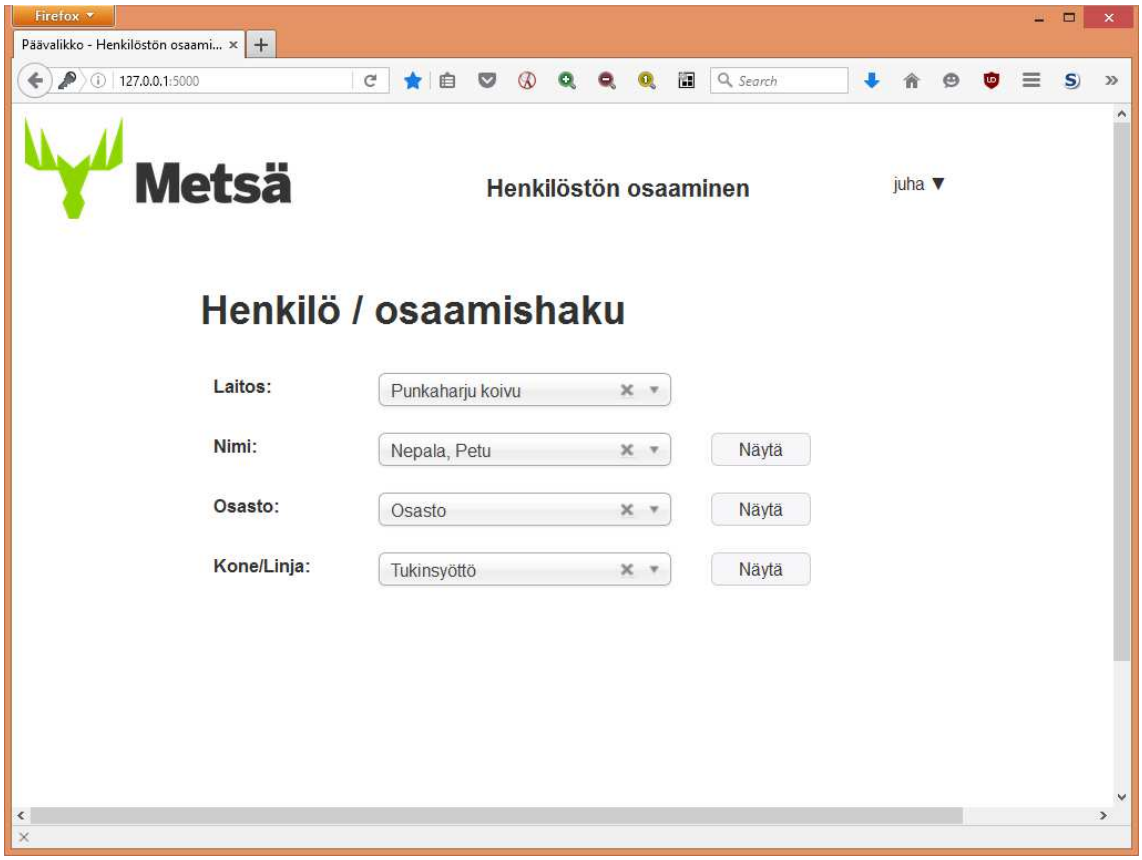
6.4 Sovelluksen päävalikko

Tavallinen käyttäjä kirjautuu sisään sovellukseen syöttämällä käyttäjätunnuksensa ja salasanansa sisäänkirjautumissivun lomakkeeseen (kuva 6.21). Sisäänkirjautumisen onnistuttua käyttäjälle esitetään sovelluksen päävalikko.



Kuva 6.21. Tavallisen käyttäjän sisäänkirjautuminen.

Päävalikko koostuu neljästä pudotusvalikosta ja kolmesta painikkeesta (kuva 6.22). Ensimmäisestä pudotusvalikosta valitaan haluttu laitos. Käyttäjä näkee tässä valikossa vain ne laitokset, joihin hänellä on vähintään lukuoikeus. Toisesta pudotusvalikosta käyttäjä voi valita nimen perusteella tietyn työntekijän. Tiimitason käyttöoikeudet vaikuttavat tämän valikon sisältöön. Valikon viereisestä Näytä-painikkeesta käyttäjä pääsee työntekijäsivulle.



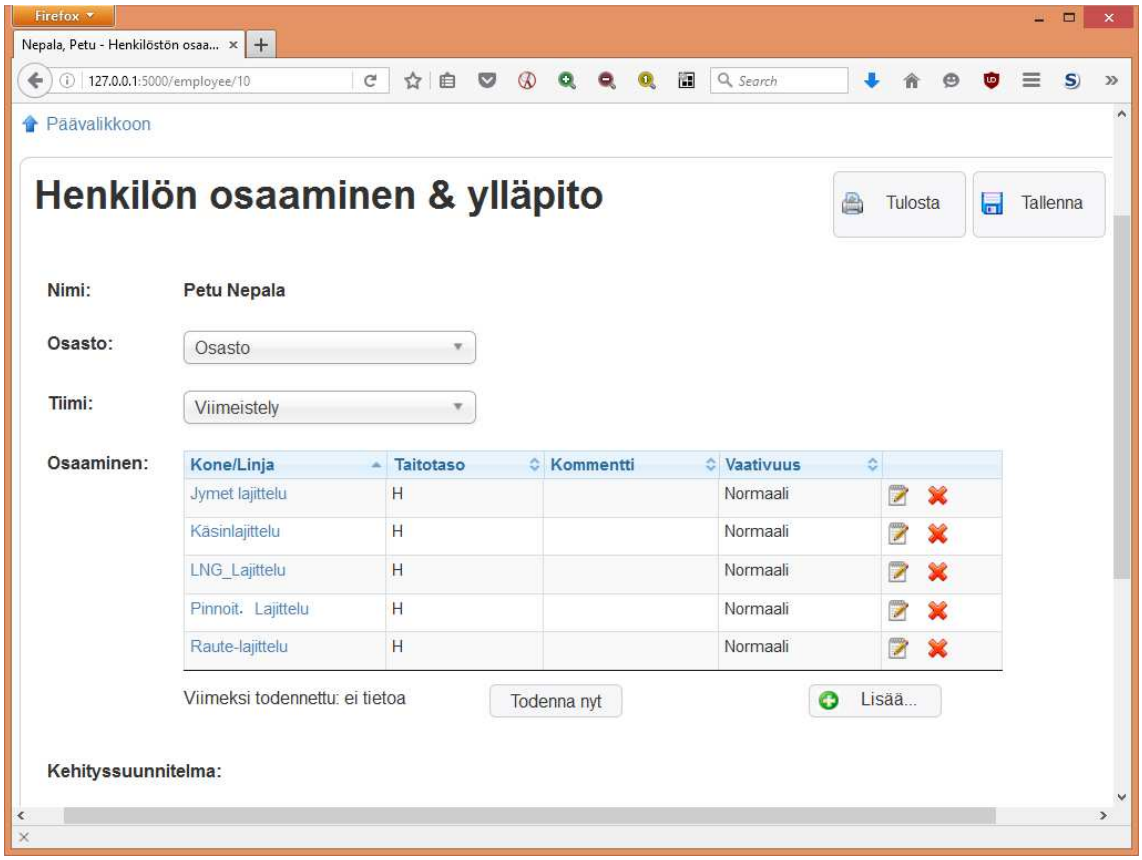
Kuva 6.22. Sovelluksen päävalikko.

6.5 Työntekijänäkymä

Työntekijänäkymässä (kuva 6.23) näytetään työntekijän nimi, osasto, tiimi, osaaminen, kehityssuunnitelma sekä huomautukset. Työntekijän nimeä ei voi tältä sivulta käsin muuttaa; se onnistuu vain pääkäyttäjän sivuston kautta. Työntekijän voi siirtää toiselle osastolle ja toiseen tiimiin, jos käyttäjällä on riittävät käyttöoikeudet. Muussa tapauksessa sovellus ei anna käyttäjän tallentaa muokattuja tietoja, mistä ilmoitetaan viestillä: "Tallentaminen estetty". Sama periaate pätee muidenkin tietojen muokkaamiseen.

Työntekijän osaaminen muodostuu niin nykyisistä kompetensseista kuin myös koulutusaikomuksista. Kompetenssin taitotaso kertoo kummasta on kyse. Kullekin yksittäiselle kompetenssille voidaan syöttää vapaamuotoinen kommentti. Työntekijän osaaminen voidaan todentaa Todenna nyt -painiketta napsauttamalla.

Sekä kehityssuunnitelma että huomautukset ovat vapaamuotoisia tekstikenttiä.

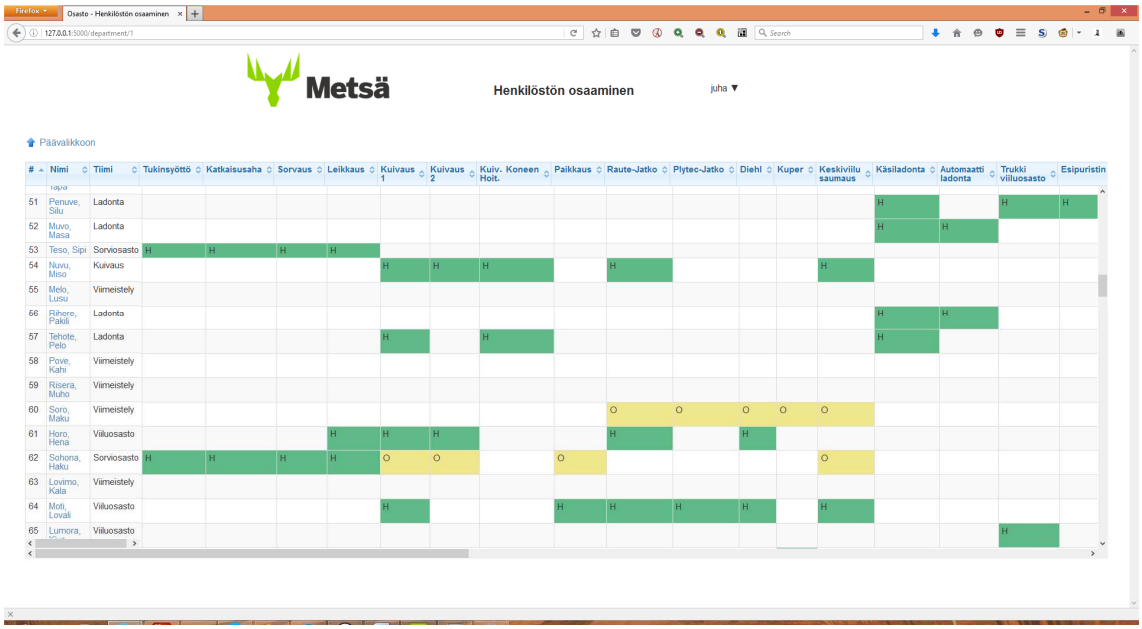


Kuva 6.23. Työntekijänäkymä.

6.6 Osastonäkymä

Päävalikon kolmannesta pudotusvalikosta löytyvät laitoksen osastot, joihin käyttäjällä on vähintään lukuoikeus. Näytä-painikkeesta avautuu taulukkonäkymä, jossa sovellukseen syötetyt tiedot esitetään työn pohjana toimineen osaamismatriisin tyyliin. Tässä taulukossa osaamiset on merkitty eri värein taitotason mukaan. Kuvassa 6.24 nähdään vihreitä ja keltaisia soluja, jotka siis vastaavat kahden eri taitotason kompetensseja. Jos kompetenssiin on liitetty kommentti, niin solun oikeassa yläreunassa on siitä merkkinä pieni punainen neliö. Kun hiiren osoittimen vie sellaisen solun kohdalle, niin kommentti tulee näytölle näkyviin.

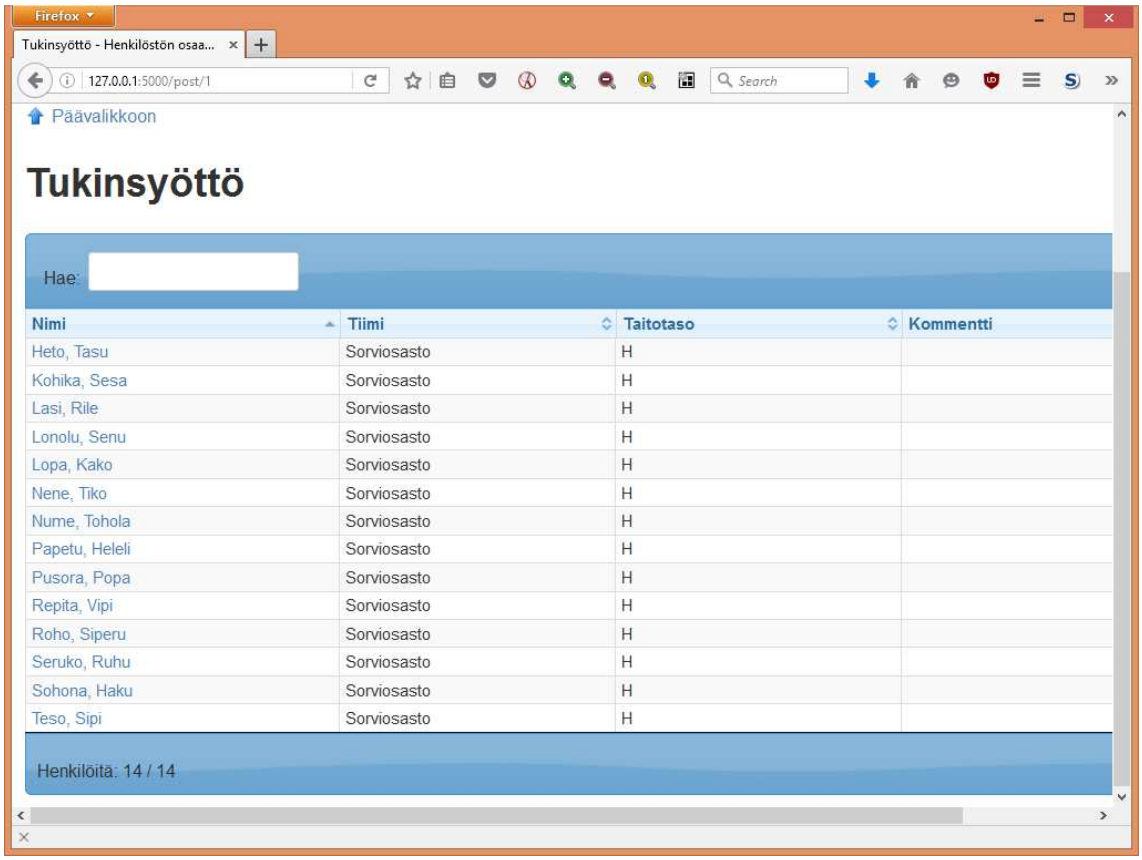
Osaamistietoja ei voi lisätä, muokata tai poistaa osastonäkymästä käsin, mutta kukin nimi on linkki työntekijäsivulle, jossa nämä toimenpiteet onnistuvat. Taulukon rivit voi lajitella minkä tahansa sarakkeen mukaisesti klikkaamalla sarakkeen otsikkoa. Sarakkeiden järjestyksen muuttaminen onnistuu vain pääkäyttäjän sivuston kautta.



Kuva 6.24. Osastonäkymä.

6.7 Kone- tai linjanäkymä

Päävalikon neljäs pudotusvalikko sisältää laitoksen koneet ja linjat. Näytä-painikkeesta avautuvalla sivulla esitetään ne työntekijät, joilla on osaamista tai koulutusaikomus valitun koneen tai linjan suhteen (kuva 6.25). Kunkin työntekijän nimi on linkki työntekijäsivulle. Sivulla olevasta taulukosta nähdään nimen lisäksi tiimi, johon työntekijä kuuluu, taitotaso kyseisellä koneella tai linjalla sekä kommentti tai huomautus, joka mahdollisesti löytyy työntekijän tiedoista.



Kuva 6.25. Kone- tai linjanäkymä.

7 Yhteenveto ja pohdinta

Projektin tarkoituksena oli tuottaa asiakkaan tarpeeseen mukautettu osaamisenhallinnan sovellus, jolla voitaisiin korvata olemassa oleva laskentataulukkoon perustuva ratkaisu. Suunnittelun viidestä pääasiallisesta tavoitteesta neljä voidaan katsoa toteutuneeksi. Ohjelma tukee useita samanaikaisia käyttäjiä, joista kukin saa käyttöoikeuksiensa mukaisen näkymän osaamistietoihin. Ohjelman asentaminen palvelimelle on melko yksinkertaista; Python 3.4 -asennuksen lisäksi tarvitaan vain muutaman paketin lataaminen PyPI- eli Python Package Index -palvelusta. Ohjelman omat tiedostot puretaan zip-paketista. Päivittäminen tapahtuu tyypillisesti purkamalla uudet tiedostot vanhojen päälle, minkä jälkeen ohjelma käynnistetään uudelleen. Ohjelman käyttöliittymä on mielestäni selkeä ja yksinkertainen, joskin viimeistelemättömän oloinen. Osaamistietoja on mahdollista käsitellä myös ohjelmallisesti HTTP API:n

kautta. Visuaalisten raporttien osalta merkittävin puute on se, että tulostustoiminnot jäivät toteuttamatta.

Jokainen projekti on mahdollisuus kehittää ja laajentaa omaa osaamista. Sain projektista hyvää oppia niin JavaScript- kuin Python-ohjelmointikielestä, ja Flask-sovelluskehys tuli tietysti tutuksi. Olin ennen projektin aloittamista tutustunut joihinkin vaihtoehtoisiin ohjelmointikieliin ja tekniikoihin, mutta valitsin tarkoituksella vakiintuneemmat, mahdollisimman suoraviivaiset ratkaisut, jotta aikataulu ei entisestään venyisi. Jälkikäteen kaduttaa lähinnä se, että en hyödyntänyt testiohjatun kehittämisen menetelmiä, jotka olisivat hyvinkin voineet parantaa projektin hallittavuutta. Olisin myös mielelläni käyttänyt enemmän aikaa ohjelman käyttökokemuksen ja työnkulun hiomiseen, jos se vain olisi ollut mahdollista.

Kuvat

Lähteet

1. Vihavainen, A. n.d. Web-palvelinohjelmointi. http://wepa.mooc.fi/index.html. Luettu 11.7.2016.

2. Wikipedia. n.d. Client–server model. https://en.wikipedia.org/wiki/Client%E2%80%93server_model. Luettu 16.5.2016.

3. Grinberg, M. 2014. Flask Web Development. Sebastopol, California: O'Reilly Media, Inc.

4. The Werkzeug Team. n.d. Serving WSGI Applications. http://werkzeug.pocoo.org/docs/0.11/serving/. Luettu 11.9.2015.

5. IETF HTTP Working Group. n.d. HTTP Documentation. http://httpwg.org/specs/.

6. Gourley, D., Totty, B., Sayer, M., Reddy, S. & Aggarwal, A. 2002. HTTP: The Definitive Guide. Sebastopol, California: O'Reilly Media, Inc.

7. Wikipedia. n.d. History of Python. https://en.wikipedia.org/wiki/History_of_Python. Luettu 6.9.2016.

8. Python Software Foundation. n.d. 12.6. sqlite3 — DB-API 2.0 interface for SQLite databases. https://docs.python.org/3/library/sqlite3.html. Luettu 22.9.2015.

9. Hipp, R. n.d. Temporary Files Used By SQLite. https://sqlite.org/tempfiles.html. Luettu 24.09.2015.

10. Hipp, R. n.d. PRAGMA Statements. https://www.sqlite.org/pragma.html.

11. Ronacher, A. 2013. API — Flask Documentation (0.10). http://flask.pocoo.org/docs/0.10/api/. Luettu 3.9.2015.

12. Ronacher, A. 2008. Frequently Asked Questions — Jinja2 Documentation. http://jinja.pocoo.org/docs/dev/faq/.

13. The jQuery Foundation. n.d. History | jQuery Foundation. https://jquery.org/history/. Luettu 11.9.2016.