VESA YLIKYLÄ

KUNNOSSAPITO-OHJELMISTON ENNAKKOHUOLTO-MODUULIN TOTEUTTAMINEN

TIETOJENKÄSITTELYN KOULUTUSOHJELMA

2014

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

KUNNOSSAPITO-OHJELMISTON ENNAKKOHUOLTO-MODUULIN TOTEUTTAMINEN

Ylikylä, Vesa
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Syyskuu 2014
Ohjaaja: Nieminen, Hans
Sivumäärä: 39
Liitteitä: -

Asiasanat: ohjelmointi, ohjelmistosuunnittelu, Python

Opinnäytetyössä käsitellään ohjelmistotuotteen moduuliin luomista osaksi olemassa olevaa tietojärjestelmää. Toteutettu moduuli on ennakkohuolto-moduuli, joka luotiin osaksi KP-ServicePartner Oy:n kunnossapito-ohjelmistoa Kuppia. KP-ServicePartner Oy on kunnossapitoalan yritys, joka on tuotteistanut käyttämänsä kunnossapito-ohjelmiston. Kuppi:n toteutuksessa on käytetty asiakas-palvelin-arkkitehtuuria.

Työssä selvitetään mitä vaiheita ohjelmistotuotteen moduulin kehittämiseen kuuluu. Tämän lisäksi selviää mitä hyötyä moduuli-rakenteisia sovelluksista on. Käsitellään mitä tietojärjestelmän määrittely tarkoittaa ja mitä vaiheita siihen kuuluu.

Moduulin toteutuksen palvelinpään ohjelmointiin käytettiin Python-ohjelmointikieltä. Kuppi käyttää palvelinpäässä Python-kielellä toteutettua Django-ohjelmistokehystä, joka on tarkoitettu web-sovellusten luomiseen. Tästä syystä ja asiakkaan vaatimuksesta johtuen moduulin toteutuksessa käytettiin Djangoa. Järjestelmän käyttäminen tapahtuu selaimella. Selainpuolen toteutuksessa käytettiin HTML-, CSS- ja JavaScript-tekniikoita. Toteutuksessa otettiin huomioon yrityksen vaatimukset sekä helppokäyttöisyys.

MAKING PRE-MAINTENANCE MODULE FOR MAINTENANCE SOFTWARE

Ylikylä, Vesa
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information Technologies
September 2014
Supervisor: Nieminen, Hans
Number of pages: 39
Appendices: -

Keywords: programming, software development, Python

This paper deals with making software module as part of existing data system. Module that is made is pre-maintenance module for Kuppi maintenance software. Kuppi is made by KP-ServicePartner Oy. KP-ServicePartner Oy works in the industrial maintenance trade. Kuppi uses client-server-architecture.

This paper explains what sort-of steps are included in developing software module. It also covers benefits of module structures in software engineering. This work gives brief description about data system specification process and steps that are part of specification process.

Python programming language is used on modules server-side execution. Kuppi uses Django Python web-framework. That and customer's requirement to use Django is why module uses this web-framework. System is used with web browser. Techniques used in user interface are HTML, CSS and JavaScript. In module development customer requirements were taken into account along with ease of use.

SISÄLLYS

1 JOHDANTO

Opinnäytetyössä tehdään ennakkohuolto-moduuli osaksi KP-ServicePartner Oy:n kunnossapitojärjestelmä Kuppia. Kuppi on tuotteistettu ja on tällä hetkellä käytössä kolmella teollisuusyrityksellä. Yrityksellä aiemmin käytössä olleessa järjestelmässä oli mahdollisuus luoda sääntöjä, joiden perusteella järjestelmä avasi työmääräimen määritettynä ajankohtana. Ominaisuutta käytettiin vanhassa järjestelmässä ja se vähensi työnjohtajien töiden avaamiseen kuluvaa aikaa. Tämän lisäksi työnjohtajien ei tarvinnut käydä papereita läpi ja huolehtia, että laitteiden huollot ja tarkastukset tulivat hoidettua ajoissa. Uudessa järjestelmässä ei tällaista ominaisuutta vielä ollut, joten sellainen tarvittiin.

KP-ServicePartner Oy on Suomessa toimiva keskisuuri yritys, jonka ydinliiketoimintaa on teollisuuden kunnossapitopalvelut. Pääpalveluina ovat tehdaslaitosten kokonaisvastuullinen kunnossapito, kunnossapidon yksittäispalvelut, koneistukset, pinnoitukset ja tuotantolinjojen modernisoinnit. Yrityksellä on seitsemän toimipistettä ympäri Suomea. Työntekijöinä yrityksen palveluksessa toimii noin 100 kunnossapidon ammattilaista. (KP-ServicePartner Oy:n www-sivut 2014)

Toiminnan kulmakivinä ovat asiakkaiden kanssa tehtävä pitkäaikainen työ laitosten tuotantovarmuuden ja kustannustehokkuuden kehittäminen avoimella yhteistyöllä ja kunnossapidon parantamisella. Oma mahdollisimman hyvä kustannustehokkuus varmistetaan yhteistyöpartneroinnilla ja matalalla organisaatiorakenteella. (KP-ServicePartner Oy:n www-sivut 2014)

2 KÄYTETTÄVÄT TEKNIIKAT

Kuppi käyttää asiakas-palvelin-arkkitehtuuria(engl. client-server). Palvelinpään toteutuksessa on käytetty Python-ohjelmointikieltä ja sitä käyttävää Django-nimistä ohjelmistokehystä (engl. Framework). Järjestelmän käyttäminen tapahtuu selaimen avulla. Käyttöliittymän toteutuksessa on käytetty HTML-, JavaScript- ja CSS-tekniikoita.

2.1 Asiakas-palvelin-arkkitehtuuri

Asiakas-palvelin-ohjelmistoarkkitehtuurilla tarkoitetaan sovellusta, jossa asiakas ottaa sovelluksella yhteyttä palvelimeen. Sovelluksen tiedot sijaitsevat palvelimella. Palvelin lähettää käyttäjälle käyttäjän pyytämät tiedot sekä tekee käyttäjän pyytämät muutokset tietoihin. Asiakkaat eivät ole tietoisia toisistaan vaan ovat ainoastaan yhteydessä palvelimeen. Asiakkaat sijaitsevat yleensä erillään palvelimesta tai vähintään eri säikeessä (engl. thread) (Laine, H. 2011).

Ratkaisun etuina käyttäjälle ovat yleensä asiakasohjelmiston (engl. client) pieni koko ja paikallista sovellusta pienemmät resurssivaatimukset. Käsiteltävät tiedot ovat aina ajan tasalla ja kaikille käyttäjille samat. Tietojen sijaitseminen palvelimella tarkoittaa myös sitä, että sovellusta käytettäessä on aina oltava palvelinyhteys. Käyttäjälle välitettävän tiedon määrästä ja käyttäjämäärästä riippuen järjestelmän siirtämä tiedonmäärä voi kasvaa suureksikin (Laine, H. 2011).

2.2 Python

Python on vuonna 1990 luotu ohjelmointikieli (Python Software Foundation. History and License). Sitä pidetään helposti opittavana, mutta tehokkaana ohjelmointikielenä. Se pitää sisällään korkean tason tietorakenteita (engl. High level data structure) ja on yksinkertainen, mutta tehokas olio-ohjelmointikieli. Korkean tason tietorakenteet tarkoittavat, että Pythonissa kaikki määriteltävät tietotyypit (engl. Data type) käyttävät sisäänrakennettuja tietorakenteita (engl. Data structure) (Python Software Foundation. What is Python? Executive Summary).

Pythonin toinen versio Python 2.0 julkaistiin 16.10.2000. Tällöin siihen lisättiin paljon uusia ominaisuuksia mm. roskienkeräys (engl. garbage collection) ja tuki Unicode-merkistölle. Roskienkeräyksellä tarkoitetaan tapaa, jolla koodinkääntäjä käsittelee ohjelman tarvitseman muistin käyttöä (Digi Wiki. Python Garbage Collection). Kehityksessä siirryttiin avoimempaan ja yhteisötukiseen kehitykseen. Pythonin kolmas versio, joka ei ole taaksepäin yhteensopiva, Python 3.0 julkaistiin joulukuussa 2008. Monet sen uudet ominaisuudet on nyt tehty myös Python 2.6 ja 2.7 versioihin. Tämä

sen taaksepäin yhteensopivuuden parantamiseksi (Python Software Foundation. What's New in Python).

Pythonin syntaksi on yksinkertainen, mutta tehokas. Koodia tarvitaan yleensä vähemmän kuin C, C++ tai Java ohjelmien vastaaviin toteutuksiin. Tämä johtuu mm. seuraavista asioista:

- Korkeantason tietorakenteet mahdollistavat monimutkaisten operaatioiden kirjoittamisen yhdellä lauseella.
- Komentorakenteiden ryhmittely on toteutettu sisennyksillä aloitus- ja päätösmerkkien sijaan.
- Muuttujien tyyppimäärittely ei ole pakollista.

(Python Software Foundation. Comparing Python to Other Languages)

2.3 Django

Django on Python-kieltä käyttävä korkean tason ohjelmistokehys (engl. Framework) web-järjestelmien toteuttamiseen. Korkean tason ohjelmistokehys sisältää HTTP-palvelinohjelman, toiminnon tiedon tallentamiseen esimerkiksi tietokannan, teemapohjamoottorin (engl. template engine), pyyntöjenkäsittelijän (engl.request dispatcher), todentamismoduulin (engl. authentication module) ja AJAX (Asynchronous JavaScript And XML) valmiudet (Python wiki. Web Frameworks). Django pyrkii mahdollistamaan nopean kehityksen kuitenkaan uhraamatta tehokkuutta. Tarkoituksena on ollut automatisoida mahdollisimman paljon ja noudattaa ”älä toista itseäsi”-periaatetta. Se soveltuu niin pieniin kuin isoihinkin toteutuksiin. Sitä käyttävät mm.

- Disqus, reaaliaikainen kommentointialusta käyttäjien verkkosivuilla
- Instagram, kuvien jakopalvelu
- Pinterest, virtuaalinen ilmoitustaulu

(Django Web Framework:n www-sivut)

Djangon kehittivät alun perin Adrian Holovaty ja Simon Willison työskennellessään web-ohjelmoijina Lawrance Journal-World lehdessä vuoden 2003 syksyllä. Ensimmäinen yleinen julkaisu tapahtui vuoden 2005 heinäkuussa. Django julkaistiin BSD lisenssin alaisena ja käyttää edelleen samaa lisenssiä. Nimensä ohjelmistokehys sai

ranskalaisen kitaristin Django Reinhardtin mukaan. Nykyään Djangon kehityksestä vastaa Django Software Foundation, joka perustettiin vuonna 2008 (The Django book. Chapter 1: Introduction to Django).

Django käyttää toiminnassaan osittain hyväksi MVC-arkkitehtuuria (engl. model-view-controller). Tällä erotellaan järjestelmän osat mallin, näkymän ja käsittelijän mukaan toisistaan lähes riippumattomiksi. Django:n käyttämät nimet ovat vähän erilaiset vaikka tarkoitus onkin sama. Tiedostotasolla jako näyttää seuraavan laiselta:

- models.py tiedostossa on tietokantataulujen määrittelyt, jotka on tehty käyttäen Python-luokaa (engl. class). Tämä on nimeltään malli.
- views.py tiedostossa on ohjelman toimintalogiikka. Tämä on nimeltään näkymä.
- urls.py tiedostossa määritellään käytettävät URL-osoitteet. Osoitteiden perusteella kutsutaan näkymän funktioita. Tämä toimii käsittelijänä.
- teeman_nimi.py on teemapohjatiedosto, jonka perusteella muotoillaan käyttäjälle näytettävä web-sivu.

Selaimelta tulevat pyynnöt ohjautuvat http-palvelinohjelman kautta käsittelijälle. Käsittelijä ohjaa tiedot määritetyn näkymän funktiolle. Näkymän funktiossa voidaan käyttää mallin tietoja. Lopuksi näkymän funktio palauttaa tiedot käsittelijälle käyttäen hyväksi teemapohjatiedostoa tai muulla tapaa muotoiltuna. Käsittelijä siirtää luodut tiedot http-palvelinohjelmalle, joka puolestaan lähettää tiedot käyttäjän selaimelle.
(The Django book. Chapter 1: Introduction to Django)

Djangossa on monipuolinen ORM (engl. Object-relation mapper), joka mahdollistaa tietomallien määrittelyn täysin pythonilla. Näiden määrittelyjen perustella luodaan tietokantaan tarvittavat rakenteet. Tietomallien tietokantakyselyitä varten Djangossa on oma ohjelmistorajapintansa. Tämän avulla tietokantakyselyt ovat helppoja ja nopeita tehdä, eikä tarvetta SQL:n osaamiselle ole. Kyseisellä ominaisuudella on kuitenkin rajoitteensa, mutta rajanpinnan kautta on mahdollista tehdä kyselyjä käyttäen SQL-lauseita. Tietoa haettaessa käyttäen SQL-lauseita on mahdollista palauttaa samanlaisia objekteja kuin rajapinnan omaa syntaksia käytettäessä. Tämä mahdollistaa

objektien käyttämisen samalla tavalla tietojenhakutavasta riippumatta (Django Web Framework:n www-sivut).

Tietokannan tietoja pääsee hallitsemaan helposti ylläpidon hallintapaneelista. Ylläpidon hallintapaneeli luodaan automaattisesti tietomallien pohjalta. Tietomallien näkyminen hallintapaneelissa on muokattavissa moduulin sisällä olevan admin.py tiedoston avulla. Sen avulla on mahdollista valita esimerkiksi mitkä tietomallin osat näytetään hallinnassa ja millaisena tietojenlistaus näkyy. Hallinnasta on myös mahdollista hallita kaikkia järjestelmässä olevia käyttäjiä, käyttäjäryhmiä, sekä niiden käyttöoikeuksia. Näiden käyttäjä ja käyttöryhmä määrittelyjen avulla sovelluksen käyttöoikeuksien hallinta on helppoa ja nopeaa. Oletuksena käyttöoikeudet mahdollistavat käytön rajaamisen vain tietomallien tietojen lukemiseen, lisäämiseen, muuttamiseen ja poistamiseen. Uusia oikeuksia pystyy lisäämään näkymän funktioiden yhteydessä. Näiden avulla pystytään rajaamaan käyttöoikeuksia tarkemmin. (Django Web Framework:n www-sivut)

URL-osoitteiden hallintaa varten on erillinen sisäänrakennettu käsittelijä. Tämä mahdollistaa juuri sellaisten URL-osoitteiden käyttämisen kuin sovellus tarvitsee. Osoitteet voidaan määritellä koko sovelluksen laajuisesti tai jokaisella moduulilla erikseen. Osoitteiden muodon oikeellisuus tarkastetaan samalla kun pyyntö ja sen mukana tulleet tiedot ohjataan oikean näkymän oikealle funktiolle. Osoitteiden määrittelyssä käytetään pythonia. (Django Web Framework:n www-sivut)

Tiedon käyttäjälle esittämisessä käytetään Djangon teemapohjakieltä. Kieltä voidaan käyttää minkä tahansa muun kielen tai tekstin lisänä. Tämä mahdollistaa muun muassa komentorakenteiden käyttämisen, sekä tiedon muotoiluun. Teemapohjalle tiedot välitetään näkymän funktiolta. Kieltä on mahdollista laajentaa Pythonin avulla. Pohjia voidaan yhdistellä keskenään, niin että jokainen sivu näyttää yleisesti samalta, mutta jotkin alueet tuodaan eri teemapohjalta. (Django Web Framework:n www-sivut)

Django tukee erilaisten välimuistitapojen käyttämistä:

- memcached, muistipohjainen välimuistitapa

- Tietokannan käyttäminen välimuistina
- Paikallinen välimuistipalvelu

Välimuisti asetukset voidaan määrittää koskemaan kaikkea Djangon sisältöä tai rajata vain valittuihin näkymiin. (Django Web Framework:n www-sivut)

Djangossa on myös mahdollisuus lokalisoinnin määrittelyyn. Tämä mahdollistaa monikielisten ja alueellisten sovelluksien tekemisen mahdollisimman vaivattomasti. Tämä on hoidettu kaksiosaisesti. Kehittäjät määrittävät mitkä osat sovelluksesta käyttävät kieli- ja aluemäärittelyjä. Kääntäjät voivat tämän jälkeen kääntää nämä osat erillisiin tiedostoihin. Näin uusien käännöksien lisääminen on helppoa. (Django Web Framework:n www-sivut)

2.4 HTML

HTML (HyperText Markup Language) on hypertekstin merkintäkieli. Sitä käytetään pääasiassa verkkosivujen määrittelyyn. HTML on elementteihin perustuva ns. merkkauskieli (engl. markup language). Koodi sisältää sisäkkäisiä ja perättäisiä elementtejä. HTML suoritetaan selaimessa. Yleisimpiä käytössä olevia HTML-standardeja ovat HTML 4.0, XHTML 1.1 ja HTML5. HTML5-standardi ei ole vielä täysin valmis, mutta suuri osa selaimista tukee ainakin osia siitä.(w3schools. HTML Introduction) (W3C. HTML & CSS.)

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <h1>Hei lukija</h1>
5  <p>Tämä on HTML sivu.</p>
6  </body>
7  </html>
```

Kuva 1. Esimerkki HTML-koodista

Yllä oleva koodi(Kuva 1.) käyttää HTML5-standardia. Ensimmäisellä rivillä määritellään käytettäväksi standardiksi HTML5-standardi. Rivillä kaksi on html-elementin aloituselementti. Päätöselementti löytyy riviltä seitsemän. Näiden elementtien väliin

tulee kaikki sivulle tuleva sisältö. Sivun sisältö jaetaan kahteen osaan otsikko (engl. head(header)) ja sisältö (engl. body). Otsikko-osassa määritellään sivun metatiedot ja sisältö-osaan tulee käyttäjälle näytettävä sisältö. Esimerkkikoodissa ei ole määriteltynä mitään metatietoa. Rivillä kolme aloitetaan sisällön osuus. Neljännellä rivillä on h1-elementti, joka on pääotsikko. Pääotsikkoa seuraavalla rivillä viisi on ”p”-elementti, joka on tekstikappale. HTML:ssä elementtien käyttötarkoitukset ovat helposti johdettavissa niiden nimistä. Pääotsikon elementin ”h” nimi tulee englanninkielisestä sanasta heading ja koska kyseessä on pääotsikko niin numero yksi. H2-elementti on ensimmäisen aliotsikon. Tekstikappaleen elementti ”p” tulee englanninkielisestä sanasta paragraph. Sisältö-osuus päätetään rivillä kuusi. Selaimella katsottaessa yllä oleva koodi luo seuraavan laisen (Kuva 2.) verkkosivun.

Kuva 2. Esimerkki HTM-sivu

2.5 CSS

CSS(Cascading Style Sheets) on pääasiassa www-sivustojen ulkoasun määrittelemiseen käytettävä tyyliohje. Se helpottaa www-sivuston ja yksittäisen sivun ulkoasun hallintaa. Tyylimäärittelyt voidaan tehdä elementtityypin mukaan eikä tarvitse määritellä samoja asioita jokaiselle elementille. Tämä myös pienentää sivujen kokoa. Määrittelyn kohdentamiseen on olemassa suuri määrä erilaisia valitsimia. Valitsimien avulla pystytään tekemään yksinkertaisista yhden elementin valintoja. Myös paljon monipuolisempien valintojen tekeminen onnistuu. Esimerkiksi voidaan muuttaa kolmannen sisäkkäisen listan viidennen rivin ensimmäinen kirjain isommaksi (Kuva 3.). (w3schools. CSS Introduction.)

```
p {
    color: red;
}

ul ul ul li:nth-child(5)::first-letter {
    font-size: 50px;
}
```

Kuva 3. Esimerkki CSS-koodista

2.6 JavaScript

On yleisin web-sivuilla käytettävä skriptikieli, jota käytetään web-sivujen dynaamisuuden toteuttamiseen. JavaScript-koodi suoritetaan käyttäjän selaimessa. Yleisimmin sillä toteutetaan käyttäjän kanssa kommunikointia, selaimen ohjaamista, tahdistamatonta kommunikointia (engl. asynchronous communication) ja näytettävän dokumentin tietojen muokkausta. JavaScriptiä käytetään myös palvelinpään ohjelmoinnissa, pelinkehityksessä, sekä mobiili-sovelluksien tekemisessä. (w3schools. JavaScript Introduction.)

Nimestään huolimatta JavaScriptillä ei ole Javan kanssa muuta yhteistä kuin nimi. JavaScript-syntaksi on ottanut vaikutteita C-kielestä. Pääsuunnitteluperiaatteisiin ovat vaikuttaneet Self ja Schema. JavaScript on dynaamisesti tyypitetty ja objekti perusteinen. (Douglas Crockford. 2001. JavaScript: The World's Most Misunderstood Programming Language.)(Java. How is JavaScript different from Java?)

2.7 jQuery

jQuery on monipuolinen JavaScript-kirjasto. Se helpottaa yleisimpiä web-sivuilla tehtäviä JavaScript-operaatioita kuten

- HTML-dokumentin muokkaaminen
- tapahtumakäsittelyt
- animaatiot
- AJAX-operaatiot. Sisältää useassa selaimessa toimivan rajapinnan.

(jQuery:n www-sivut)

3 MODUULIN MÄÄRITTELY

3.1 Moduuli

Ohjelmistotuotteen moduuli on järjestelmän mahdollisimman itsenäinen osa. Moduulit voivat vaatia toimiakseen toisia moduuleita tai toimia itsenäisesti. Moduulien välistä vuorovaikutusta varten moduulille tarvitsee luoda rajapinta. Rajapinnan kautta moduulit voivat käyttää toisessa moduulissa määriteltyjä asioita. Rajapintaa luodessa päätetään mitkä asiat ovat sellaisia joihin muista moduuleista on tarvetta päästä käsiksi.

Järjestelmän jakaminen pienempiin osiin eli moduuleihin helpottaa järjestelmän ylläpitoa. Tällöin on mahdollista toteuttaa kaikki samaan asiaan liittyvät toiminnot yhdessä paikassa. Tämä pitää koodin paljon helpommin hallittavana, kun osat ovat selkeästi eroteltuina toisistaan.

Tuotteen näkökulmasta moduulit mahdollistavat järjestelmässä käytettävän vain tarvittavia moduuleita. Esimerkiksi videovuokraamon tietojärjestelmä tarvitsee varastonhallintaan moduulin, mutta ei työtilaukset moduulia. Ohjelmistoa myyvä taho antaa asiakkaan valita tarvitsemansa moduulit, eikä tarjota järjestelmää ainoastaan kokonaisena. Tällöin asiakas voi säästää järjestelmän kustannuksissa kun ei tarvitse maksaa ominaisuuksista joita ei käytetä. Järjestelmän ollessa kustannustehokkaampi asiakkaan kynnys siirtyä käyttämään järjestelmää laskee. Vain tarvittavien moduulien pyörittämiseen tarvitaan yleensä vähemmän resursseja kuin suuren kaiken kattavan järjestelmän. Tästä on etua myös sovellusta tarjoavalle taholle, jos he ajavat asiakkaiden sovelluksia omilla palvelimillaan.

3.2 Tietojärjestelmän määrittely

Ennen tietojärjestelmän tai sen osan määrittelyä tarvitsee asiakkaalla olla tarve kyseiselle toiminnalle. Tarpeen ilmettyä selvitetään voidaanko kyseinen tarve hoitaa päivittämällä vanhaa järjestelmää ja onko tämä kustannustehokas vaihtoehto. Esimerkiksi nykyisen järjestelmän ollessa elinkaarensa lopussa sen kehittäminen ei välttä-

mättä kannata vaan on parempi siirtyä uuteen järjestelmään. Asiakkaan selvitettyä mahdollisuudet vanhan järjestelmän kehittämiseen tai uuden järjestelmän käyttöönottamiseen voidaan siirtyä vaatimusmäärittelyyn. (Johdatus tietojärjestelmiin.)

Vaatimusmäärittelyssä kerätään tietoa tulevan järjestelmän vaatimuksista eri sidosryhmiltä. Vaatimusmäärittelyssä ei oteta kantaa miten järjestelmä teknisesti toteutetaan vaan siihen mitä järjestelmältä vaaditaan. Vaatimukset voidaan jakaa toiminnallisiin ja ei toiminnallisiin vaatimuksiin. Vaatimuksien hahmottamiseen voidaan käyttää esimerkiksi käyttötapauskaavioita. Toiminnalliset vaatimukset kuvaavat toimintoja joita järjestelmän tulisi pystyä tekemään. Esimerkiksi työmääräimelle on pystyttävä lisäämään liitteitä. ”Ei toiminnalliset”- vaatimukset ovat reunaehtoja järjestelmälle. Esimerkkinä reunaehdoista on järjestelmän käyttämä arkkitehtuuri ja käyttäjälle määriteltyjen töiden maksimi määrä. Kaikki vaatimukset priorisoidaan sen mukaan mitä järjestelmässä on oltava heti. Priorisoitaessa vaatimuksia voidaan huomata vaatimusten olevan ristiriidassa keskenään. Osa vaatimuksista saattaa osoittautua tarpeettomaksi ja jäädä pois kokonaan. Ominaisuuksia, joita ei tarvita järjestelmään heti, voidaan sopia toteutettavan myöhemmin. (Johdatus tietojärjestelmiin)(Paakki J. 2011)

Vaatimusmäärittelyn pohjalta luodaan vaatimusmäärittelydokumentaatio. Dokumentaatioon tulisi kirjata asiat mahdollisimman tarkasti myöhempien epäselvyyksien välttämiseksi. Dokumentaatioon tulisi kirjata ainakin hankeen toimeksianto, jokaisen vaatimuksen (toiminnallisen ja ei toiminnallisen) tarkka kuvaus sekä rajoitteet. Vaatimukset olisi hyvä kirjata priorisoituina. Lisäksi olisi hyvä olla kuvaus järjestelmän nykytilasta sekä sille asetetuista tavoitteista. (Johdatus tietojärjestelmiin)(Paakki J. 2011)

Vaatimusmäärittelyn perusteella aloitetaan järjestelmän määrittely eli järjestelmäanalyysi. Määrittelyssä pyritään luomaan toiminnallinen määrittely vaatimusmäärittelydokumentaation perusteella. Toiminnallisessa määrittelyssä määritellään mitä järjestelmän tulee tehdä. Määrittely koostuu jokaisen toiminnon ja rajapintojen kuvauksista sekä tietojen ja tietokantojen kuvauksista. Tässäkin tapauksessa mahdollisimman laaja ja selkeä dokumentointi on tärkeää. (Johdatus tietojärjestelmiin)

3.3 Ennakkohuolto moduulin vaatimukset

Moduuli tulee osaksi käytössä olevaa Kuppi kunnossapitojärjestelmää. Moduuliin on pystyttävä määrittämään ennakkohuoltoja. Ennakkohuollot ovat määriteltynä aikana ja aikavälillä tehtäviä töitä. Ennakkohuoltojen tietojen perusteella järjestelmän pitää pystyä luomaan uusia työmääräimiä. Ennakkohuoltojen hallitsemista varten järjestelmään tarvitaan näkymä, jossa näkyvät kaikki järjestelmässä olevat ennakkohuollot. Ennakkohuoltoja on myös pystyttävä lisäämään ja muokkaamaan helposti. Ennakkohuolloissa suoritettavat vaiheet ovat samantyyppisillä laitteilla samat, mutta jotkin laitteiden ominaisuudet voivat vaikuttaa vaiheisiin. Esimerkiksi nostureissa nostokapasiteetti vaikuttaa huollon vaiheisiin. Vaiheet ovat ennakkohuollon tyypistä riippuen erilaiset. Vaiheiden lisäämistä, muokkaamista ja hallitsemista varten tarvitaan näkymät. Ennakkohuoltoja on oltava mahdollista ketjuttaa reiteiksi. Reittien lisäämiseen, muokkaamiseen ja hallitsemiseen tarvitaan näkymät.

Ennakkohuolloille kirjattavien tietojen on oltava sellaiset, että niiden perusteella on mahdollista avata työmääräimiä. Yhdelle ennakkohuollolle voi kohdistua vain yksi vaiheryhmä. Ennakkohuoltojen työmääräimeksi muodostumista varten tarvitaan seuraavia tietoja:

- Deadline, päivämäärä johon mennessä työ on oltava tehty
- Ennakkoaika, kuinka paljon ennen deadlinea luodaan työmääräin
- Intervalli, huoltojen välinen aika
- Intervallin laskentatapa. Lasketaanko edellisestä deadline:sta vai edellisen samantyyppisen huollon valmistumisesta.
- Laite ja asiakas, työn kohdistamista varten
- Vaiheryhmä
- Laskutukseen liittyvät tiedot. Asiakkaalla olevien oletustietojen muuttaminen.
- Tiimi tai henkilö. Työn kohdistamiseksi tiimille tai työntekijälle.
- Päätyö, avautuvien töiden liittämiseksi päätyölle.

Luotavien työmääräimien tilaksi tulee ”odottaa hyväksyntää”-merkintä.

4 TOTEUTETTU MODUULI

4.1 Reitit

Reittien tarkoitus on liittää yhteen ja järjestää samalla kertaa tehtäviä ennakkohuoltoja. Yhteen voidaan liittää saman yrityksen ennakkohuoltoja. Reittien käytöllä pyritään pääsemään mahdollisimman optimaaliseen töiden suoritusjärjestykseen. Käytännössä tämä tarkoittaa sitä, ettei töissä tarvittavia työkaluja ja apulaitteita tarvitsisi siirrellä edestakaisin rakennuksen sisällä tai rakennuksien välillä. Reittejä ei kuitenkaan voida käyttää kaikissa tapauksissa, koska kaikkia samalla alueella tehtäviä huoltoja ei voida suorittaa samalla kertaa. Työn sopiva ajankohta riippuu asiakkaan tuotannon tarpeista, huolloille ja tarkastuksille säädetyistä väleistä. Tästä johtuen töitä, joilla ei ole niin tarkka toteuttamisajankohta toteutetaan kun se parhaiten sopii asiakkaan ja huoltajien aikatauluun. Reitit eivät graafisesti näytä reittiä kartalla vaan listana, joka kertoo missä järjestyksessä työt tulisi tehdä.

Reittien lisäämistä varten järjestelmään tarvitaan lisäys- ja listausnäkymä. Reittejä on myös pystyttävä muokkaamaan, joten tarvitaan muokkausnäkymä. Lisäksi reittien sisällön hallitsemiseen tarvitaan näkymä.

Reittejä varten tarvitaan kaksi erillistä tietomallia (engl. model). Ensimmäinen tietomalli nimeltään reitti (kuva 4.) pitää sisällään reitin tiedot: nimen, koodin ja asiakkaan. Django luo tietokantatauluille automaattisesti pääavaimen (engl. Primary key) jos sellaista ei ole tietomallissa määritelty. Oletuksena pääavain on nimeltään tunniste (engl. id). Reitti tietomalliin myös määritellään ”monta moneen”-tyyppinen kenttä (engl. many-to-many). Django luo ”monta moneen”- kentän perusteella tietokantaan aputaulun. Tämä aputaulu sisältää ainoastaan kaksi viiteavainta (engl. Foreign key). Ensimmäinen viiteavaimista viittaa tauluun jossa kenttä on määritelty. Toinen viiteavain on kentän määrittelyssä annettuun tauluun. Aputaulun voi määrittää itse ja määrittää kentän määrittelyissä, että käytetään sitä aputauluna. Tämä antaa mahdollisuuden lisätä aputauluun muuta sisältöä viiteavaimien lisäksi.

```
class reitti(models.Model):
    nimi            = models.CharField(max_length=255)
    koodi           = models.CharField(max_length=50)
    asiakas         = models.ForeignKey(asiakas)
    ennakkohuollot  = models.ManyToManyField(ennakkohuolto, blank=True, null=True, related_name="reittiennakkohuolto", through="reittiennakkohuollot")

class reitinennakkohuollot(models.Model):
    reitti        = models.ForeignKey(reitti)
    ennakkohuolto = models.ForeignKey(ennakkohuolto)
    jarnum        = models.PositiveIntegerField()
```

Kuva 4. Tietomallit reitti ja reittiennakkohuollot.

4.1.1 Listausnäkymä

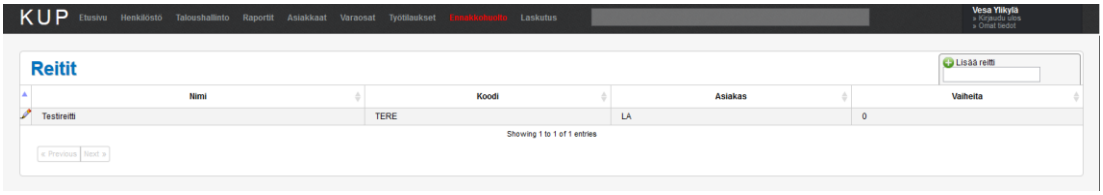
Listausnäkymän tarkoitus on nimensä mukaisesti listata järjestelmään lisätyt reitit. Listauksessa näytetään kaikki tiedot mitä reitistä on järjestelmässä: nimi, koodi ja asiakas. Näiden lisäksi näytetään reittiin kuuluvien huoltojen määrä.

Näkymässä tarkastetaan aluksi, että käyttäjällä on oikeudet nähdä reittilistaus. Reittejä tietokannasta haettaessa haetaan vain niiden asiakkaiden reitit, joita käyttäjän on lupa nähdä. Tietojen noutamisen jälkeen lasketaan jokaiseen reitin vaiheiden määrä (Kuva 5.). Näkymässä välitetään teemapohjalle reitti-objektit sekä käyttäjän tiedot.

```
for reittiobjekti in reittiobjektit:
    reittiobjekti.vaiheet = reittiobjekti.ennakkohuollot.all().count()
```

Kuva 5. Reittien vaiheiden laskeminen.

Teemapohjassa (engl. template) käytetään for-komentorakennetta muodostamaan HTML-taulukon rivit reitti-objekteista. Tämä taulukko alustetaan käyttäen Datatables jQuery-lisäosaa. Taulukon riveille lisätään linkki tietojen muokkaukseen siirtymiseen ja tapahtumankäsittelijä rivin painamiseen(engl. onClick), joka siirtää käyttäjän reitin hallintanäkymään. Näkymän toiminnot-valikkoon lisätään linkki uusien reittien lisäämiseen. Käyttäjän selaimessa sivu näyttää seuraavanlaiselta (Kuva 6.).



Kuva 6. Reittien listausnäkymä.

4.1.2 Lisäys- ja muokkausnäkymä

Reitin tietojen lisäämiseen ja muokkaamiseen käytetään samaa näkymä (engl. view) -funktiota sekä samaa teemapohjaa. Tietojen syöttämistä varten tarvitaan lomake. Lomakkeen luomisessa käytetään avuksi Djangon ModelForm-ominaisuutta. ModelForm-luokka luo tietomallin perustella automaattisesti lomakkeen, jonka kentät määräytyvät tietomallin kenttien tyypin perusteella. Esimerkiksi tietomallissa oleva viiteavainkenttä saa lomakekentäkseen pudotusvalikon. Tässä tapauksessa automaattisesti luotavat kentät ovat tyypeiltään oikeanlaiset. Kenttien nimet vaihdetaan määrityksissä suomenkielisiksi ja poistetaan ennakkohuollot kenttä lomakkeelta. Poistettu kenttä ei näy lomakkeella eikä sen tietoja tarkasteta kun lomakkeen tietoja tarkastetaan. Sille on mahdollista syöttää tietoja näkymä-funktiossa, mutta ei lomakkeen avulla.

```
class reittiLomake(ModelForm):
    def __init__(self, *args, **kwargs):
        super (reittiLomake, self ).__init__(*args,**kwargs)

        self.fields['nimi'].label       = "Nimi"
        self.fields['koodi'].label      = "Koodi"
        self.fields['asiakas'].label    = "Asiakas"

    class Meta():
        model = reitti
        exclude = ['ennakkohuollot']
```

Kuva 7. Reitin lisäys- ja muokkauslomakkeen määrittely

Näkymä-funktiossa tarvitaan käsittelytavat kaikille mahdollisuuksille

- Uuden reitin lisääminen
- Uuden reitin tallentaminen
- Olemassa olevan reitin muokkaaminen
- Olemassa olevan reitin tallentaminen

Tapauksien erittely koodissa käy helposti, kun tiedetään, että selain suorittaa GET-operaation kun käyttäjä tulee sivulle linkin kautta tai suoraan osoitteella. Tallennettaessa tietoa on lomakkeen lähettämiseen määritelty käytettäväksi POST-operaatiota. Lisäyksen ja muokkauksen erottaa toisistaan URL-osoitteiden rakenne. Muokattaessa osoitteen perään lisätään muokattavan reitin tunniste. Djangon URL-käsittelijä etsii

pyydetystä osoitteesta tunnistetta. Mikäli URL- käsittelijään määritettyihin ehtoihin sopiva tunniste löytyy, lähetetään se näkymän funktiolle. Näiden tietojen avulla saadaan eriteltyä kaikki neljä tapahtumaa toisistaan (Kuva 8.).

```
def lisaaReitti(request, tunniste=False):
    profile = request.user.get_profile()
    reittiobjekti = None

    if request.method == 'POST' and tunniste:
        # Olemassa olevan reitin tallentaminen
        reittiobjekti = get_object_or_404(reitti, id=tunniste)
        reittilomake = reittiLomake(request.POST, instance=reittiobjekti)
        if reittilomake.is_valid():
            reittilomake.save()
            return HttpResponseRedirect("/reitienlistaus/")
    elif request.method == 'POST':
        # Uuden reitin tallentaminen
        reittilomake = reittiLomake(request.POST, instance=reittiobjekti)
        if reittilomake.is_valid():
            reittilomake.save()
            return HttpResponseRedirect("/reittienlistaus/")
    elif tunniste:
        # Olemassa olevan reitin muokkauslomakkeen luominen
        reittiobjekti = get_object_or_404(reitti, id=tunniste)
        reittilomake = reittiLomake(instance=reittiobjekti)
    else:
        # Uuden reitin muokakuslomakkeen luominen
        reittilomake = reittiLomake()

    t = loader.get_template('lisaareitti.html')
    c = Context({
        'profile': profile,
        'lomake': reittilomake,
        'obj': reittiobjekti,
    })
    return HttpResponse(t.render(RequestContext(request, c)))
```

Kuva 8. Näkymä-funktio lisaaReitti

Käyttäjän halutessa luoda uusi reitti, tarvitaan ainoastaan tyhjä lomake sekä tietojen lähetys-painike. Lomake saadaan näkymään pyytämällä se tietomallista. Tämän jälkeen lähetetään tyhjä lomake ja määrittelemätön reittiobjekti teemapohjalle. Siellä sivun tekstit ja lomakkeen lähetysosoite muotoillaan lisäykseen sopivaksi, koska reitti-objektia ei ole määritelty (Kuva 9).

```
{%if obj%}<form action="/reitinlisays/{{obj.id}}/" method="POST">
{%else%}<form action="/reitinlisays/" method="POST">{%endif%}
...
{%if obj%}<input type="submit" value="Muokkaa" class="button">
{%else%}<input type="submit" value="Lisää" class="button">{%endif%}
```

Kuva 9. Osia lisaareitti.html teemapohjasta

Käyttäjän täytettyä tiedot ja painettua lähetä-painiketta, lähettää selain lomakkeessa olevat tiedot palvelimelle. Siellä palvelinohjelmisto ohjaa tiedot Djangon URL-käsittelijälle. Se siirtää tiedot lisäykseen käytettäväksi merkitylle näkymän lisaareitti-funktiolle. Koska selain lähetti lomakkeen on kyseessä POST-operaatio eikä osoitteessa ollut mukana reitin-tunnistetta. Näiden tietojen perusteella ehtorakenne ohjaa suorituksen oikeaan paikkaan.

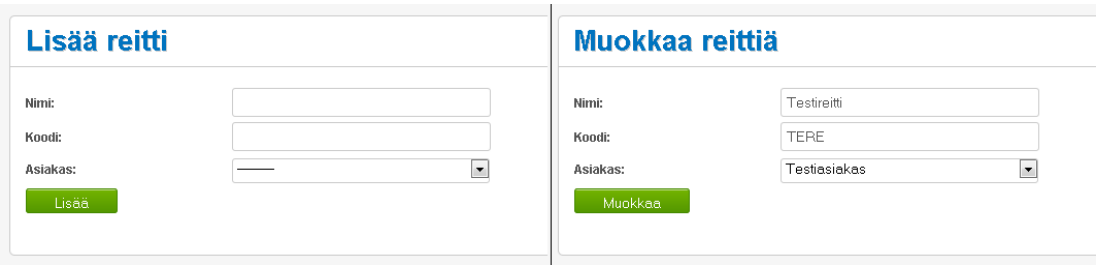
Aluksi luodaan reittiLomake-objekti, jolle annetaan parametrina lomakkeelta tulleet tiedot. Tämän jälkeen käytetään reittiLomake-objektilla olevaa is_valid-funktiota. Funktio tarkastaa lomakkeelle syötettyjen tietojen oikeellisuuden. Oletuksena oikeellisuus tarkistetaan tietomallille määritettyjen ehtojen perusteella. On myös mahdollista määritellä omia ehtoja oikeellisuuden tarkistukseen. Tässä tapauksessa lomakkeelta tulevien tietojen oikeellisuuden tarkistamiseen riittää oletustarkistus. Esimerkiksi viiteavainkentän arvon on vastattava tietokannasta löytyvää arvoa. Tämän jälkeen tiedot tallennetaan tietokantaan käyttäen reittiLomake-objektin tallennus (save)-funktiota, jos tiedot olivat oikeanlaisia. Tallentamisen jälkeen käyttäjä ohjataan takaisin reittien listaukseen.

Jos syötetyt tiedot eivät ole oikeanlaisia luo Django automaattisesti virheilmoitukset kentille, joiden tiedot eivät ole oikein. Virheilmoitukset kertovat miksi käyttäjän syöttämä arvo ei kelvannut. Tämän lisäksi ajetaan järjestelmään itse tehty addFormErrorMessages-funktio, joka lisää virheitä sisältäviin kenttiin CSS-luokan error. Error-luokka muuttaa kentän punaiseksi, jotta käyttäjän on helpompi huomata virheelliset kentät. Käsitelty lomake lähetään käyttäjälle takaisin uuden puhtaan lomakkeen sijaan. Käsitellyssä lomakkeessa on käyttäjän aiemmin syöttämät tiedot ja virheet.

Käyttäjän valitessa reitin muokkauksen reittilistauksesta tai hallintanäkymästä siirrytään samaan URL-osoitteeseen kuin reittiä luotaessa, mutta osoitteeseen lisätään reitti-tunniste. Näkymässä tarkistetaan, että valitulla tunnisteella on olemassa reitti joka käyttäjän on lupa nähdä. Tarkistaminen tapahtuu hakemalla reitti-objekti käyttämällä Djangon oikopolkufunktiota get_object_or_404. Funktiolle syötetään parametreina halutun objektin nimi sekä hakuehdot queryset-säännöillä. Tässä tapauksessa ha-

kuehtoina on reitti-tunniste sekä reitille määritellyn asiakaan kustannuspaikan kuuluminen käyttäjälle sallittuihin kustannuspaikkoihin. Funktion löytäessä ehtoja vastaavan objektin palauttaa se tämän objektin. Jos ei löydetä yhtään ehtoja vastaavaa objektia tai löydetään useampi kuin yksi objekti, laukaisee (engl. raise) funktio 404-virheen. Tällöin käyttäjä ohjataan suoraan 404-sivulle.

Tässä tapauksessa tietokannasta ei kuitenkaan voi löytyä kuin yksi tai ei yhtään vastaavuutta. Tästä pitää huolen tietokanta ja ehdoissa määritelty tunniste. Tunnistekenttä on tietokannassa määritelty uniikiksi (engl. unique). Tarkoittaen ettei kyseisessä taulussa voi olla kuin yksi rivi jokaista tunnisteen arvoa kohden. Jos funktio löytää objektin, jatketaan luomalla lomake, jolle annetaan parametrina haettu reitti-objekti. Tällöin lomake täytetään objektin tiedoilla. Tämä lomake lähetetään teemapohjalle, mutta tällä kertaa lomake-objektin lisäksi välitettävä reitti-objekti ei ole määrittelemätön(engl. None). Teemapohjassa reitti-objektin perusteella tiedetään, että kyseessä on muokkaus tilanne ja muotoillaan sivu muokkausta varten. Muokkauta varten vaihdetaan tekstejä sekä lisätään lomakkeen lähetysosoitteeseen reitti-objektin tunniste. Tällöin lomakkeen tietojen saapuessa palvelimelle tiedetään, että halutaan muokata kyseistä objektia eikä luoda uutta objektia.

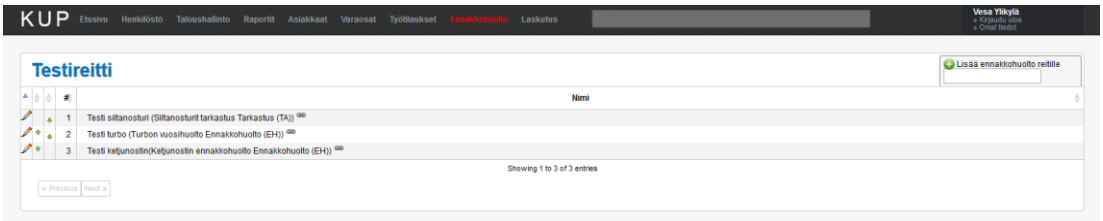


Kuva 10. Reitin lisäyksen ja muokkauksen eroavaisuudet selaimessa

Muokkauslomakkeen tietojen palvelimelle saapumisen jälkeen tapahtuu samat asiat kuin lisäyslomaketta tallennettaessa. Näkymän funktiossa ehtorakenne ohjaa suorituksen eri paikkaan. Ennen muokkauksen tallentamista varmistetaan, että käyttäjällä on oikeus muokata reittejä. Tallennettaessa tarkistetaan vielä uudelleen, että muokattava reitti on sellainen jonka näkemiseen käyttäjällä on oikeudet. Tallentamisen jälkeen käyttäjä ohjataan takaisin reittien listaukseen. Virheellisten tietojen käsittely tapahtuu samalla tavalla kuin uutta reittiä tallennettaessa.

4.1.3 Reitin hallintanäkymä

Reittien hallintanäkymässä (Kuva 11.) näytetään reittiin liittyvät ennakkohuollot oikeassa järjestyksessä. Reittiin kuuluvat ennakkohuollot näytetään HTML-taulukkona. Näytettäviä tietoja ovat järjestysnumero ja huollon nimi. Tietojen lisäksi taulukossa on painikkeet huoltojen järjestyksen vaihtamiseen sekä linkki huollon muokkausnäkymään. Näkymän toiminnot laatikossa on lisää ennakkohuolto reitille-painike, josta siirrytään ennakkohuollon reittiin lisääminen-näkymään.



Kuva 11. Reitin hallintanäkymä

Ennakkohuollon lisääminen reitille on toteutettu samalla tavalla kuin reittien lisääminen. Käytetään yhtä funktiota näkymässä, jossa hoidetaan lisääminen ja muokkaaminen. Molemmissa tapauksissa sivulle tultaessa on osoitteessa oltava mukana reitti-tunniste. Reitti-tunniste tarvitaan, jotta tiedetään mihin reittiin halutaan liittää ennakkohuolto. Muokattaessa reitti-tunnisteella ei ole samanlaista merkitystä, koska reittiennakkohuollot-objektilla on tieto mille reitille se kuuluu. Reitti-tunniste pidetään mukana osoitteiden yhtenäisenä pitämisen vuoksi. Muussa tapauksessa olisi jouduttu käyttämään kahta eri URL-osoitetta sekä kahta erillistä näkymän funktiota. Tämä johtuu reittiennakkohuollot-objektien lisäämiseen käytettävän funktiolle määritetyistä parametreista: pyyntö (engl. request), reitti-tunniste ja reittiennakkohuollot-tunniste. Pyyntö on Djangon sisäänrakennettu objekti, jonka URL-käsittelijä automaattisesti lähettää näkymän funktiolle. Se sisältää selaimelta tulevat tiedot sekä Djangon tietoja, muun muassa kirjautuneen käyttäjän. Vaihtoehtona olisi käyttää vain yhtä parametria pyynnön lisäksi ja yhteistä osoitetta. Tällaisessa ratkaisussa ei kuitenkaan pystyttäisi funktiossa tietämään onko kyseessä reitti-tunniste vai reittiennakkohuollot-tunniste.

Funktion alussa tarkastetaan reitti-tunnisteen oikeellisuus käyttämällä get_object_or_404-funktiota. Uutta reittiennakkohuollot-objektia luotaessa tämä

määrittää mihin reittiin uusi objekti liitetään. Muokattaessa reittiennakkohuollot-objektia reitti-tunnisteella varmistetaan, että käyttäjä tulee oikealta reitiltä muokkaukseen. Sen lisäksi reitti-tunnisteen perusteella osataan ohjata käyttäjä onnistuneen tallentamisen jälkeen oikean reitin hallintanäkymään takaisin.

Uutta ennakkohuoltoa reitille lisättäessä haetaan kaikista reittiin liitetyistä huolloista suurin järjestysnumero. Haettuun järjestysnumeroon lisätään yksi ja saadaan uudelle liitettävälle huollolle järjestysnumero, joka ei ole päällekkäinen minkään muun kanssa. Jos reitillä ei ole yhtään liitettyä ennakkohuoltoa niin asetetaan järjestysnumeroksi 1 (Kuva 12.).

```
maxobj = reittiennakkohuollot.objects.filter(reitti=reittiobjekti).aggregate(Max('jarnum'))
seurnum = 1
if maxobj.get('jarnum__max', 0) is not None:
    seurnum = 1 + maxobj.get('jarnum__max', 0)
```

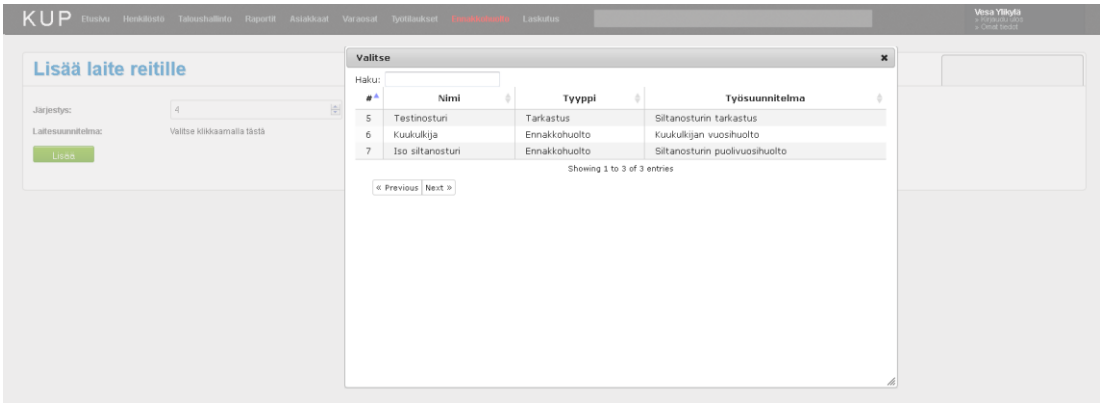
Kuva 12. Liitoksen seuraavan järjestysnumeron selvittäminen

Reittiennakkohuollot-mallissa on kolme kenttää: reitti, ennakkohuolto ja järjestysnumero. Näistä ennakkohuolto ja järjestysnumero näkyvät lomakkeella. Reitti on määritelty pois lomakkeelta lomaketta määriteltäessä. Lomaketta luotaessa sille annetaan parametrina reitti-objekti ja seuraava järjestysnumero.

Ennakkohuolto-kentän lomaketyypiksi määritellään datatableSelect-kenttä. Tämä on järjestelmään tehty lomakkeen kenttätyppi, joka luo sivulle jQuery UI(User Interface)-kirjaston modal-elementin. Modal-elementti on sivun sisällön päälle tuleva ponnahdusikkunan kaltainen elementti. Sen sisään voidaan laittaa kaikenlaista sisältöä. Modal-elementtiin tulee sisällöksi HTML-taulukko, joka määritellään käyttämään Datatables jQuery-lisäosaa (Kuva 14.). Taulukosta käyttäjä voi etsiä ja valita haluamansa vaihtoehdon. Tämä helpottaa oikean valinnan löytämistä verrattuna perinteiseen pudotusvalikko-elementtiin verrattuna. Taulukkoon tulee nimen lisäksi huollon tyyppi ja työsuunnitelma. DatatableSelect-kentälle haetaan vaihtoehdoiksi ainoastaan valitun reitin asiakkaan laitteiden huollot (Kuva 13.). Tämä nopeuttaa valinnan tekemistä entisestään kun valittavana eivät ole kaikki ennakkohuollot. DatatableSelect-kenttä määritellään näkymän funktiossa. Lopuksi lähetään profiili-ja reitti-objektit, lomake ja reittiennakkohuollot-objekti teemapohjalle.

```
ehobjektit = ennakkohuolto.objects.filter(asiakas=asiakasobjekti)
ehotsikot = ["#","Nimi","Tyyppi",u"Työsuunnitelma"]
ehvaihtoehdot = []
if ehobjektit:
    for i in ehobjektit:
        ehvaihtoehdot.append((i.id, i.nimi, i.tyyppi, i.tyosuunnitelma))
reittiennakkohuollotLomake.fields['ennakkohuolto'].widget = DatatableSelect(attrs={'header': ehotsikot})
reittiennakkohuollotLomake.fields['ennakkohuolto'].choices = ehvaihtoehdot
```

Kuva 13. Ennakkohuolto-kentän määrittäminen



Kuva 14. Modal-elementti laitteen liitoksessa reitille

Reittiennakkohuolto-objektin tallentaminen toimii samalla periaatteella kuin reitin lisäämisessäkin. Tarkistetaan reitti-tunniste ja annetaan se ja lomakkeelta tulevat tiedot rettiennakkohuollotLomake-luokalle lomakkeen luontia varten. Varmistetaan että tiedot ovat oikeanlaisia. Tässäkin tapauksessa tietomalliin syötettyjen tietojen perusteella pystytään tekemään riittävä tarkastus. Riittävä tarkastus tässä tapauksessa on että ennakkohuolto on tietokannassa olemassa ja järjestysnumero on todellakin numero eikä esimerkiksi kirjain. Tulevaisuudessa voitaisiin lisätä tarkastus, ettei kyseistä huoltoa ja tai järjestysnumeroa ole jo reitillä. Järjestelmän toiminnan kannalta järjestysnumeroiden päällekkäisyydellä ei ole mitään merkitystä. Myöskään saman huollon esiintyminen kahdesti reitillä ei vaikuta järjestelmän toimintaan. Reittiin liitetyt ennakkohuollot näkyvät vain käyttöliittymässä eikä niiden perusteella luoda mitään. Jos syötetyt tiedot ovat oikeanlaisia, tallennetaan objekti. Tallentamisen jälkeen siirretään käyttäjä takaisin reitin hallintanäkymään. Virheiden käsitteleminen tapahtuu samalla tavalla kuin lisaaReitti-funktiossa.

Reittiennakkohuollot-objektin muokkaukseen tultaessa annetaan funktiolle reitti-tunnisteen lisäksi myös muokattavan objektin-tunniste parametrina. Näiden molempien olemassa oleminen tarkastetaan tietokannasta get_object_or_404-funktiota apuna käyttäen. Muokattavan objektin hakuehdoissa on lisäksi reitti-objekti. Tämän jälkeen luodaan lomake muokattavan objektin tietojen perusteella. Tämän jälkeen pro-

sessi etenee samalla tavalla kuin reittiennakkohuollot-objektin luomisessa. Ennen muutosten tallentamista tarkistetaan annettujen parametrien ja lomakkeelta tulleiden tietojen oikeellisuus.

Reittien järjestystä on mahdollista vaihtaa käyttöliittymässä käyttämättä reittiennakkohuoltojen muokkausta. Järjestyksen vaihtamista varten on näkymään (engl. view) tehty oma funktio. Parametreina sille syötetään pyynnön (engl. request) lisäksi reitti-tunniste, reittiennakkohuollot-tunniste ja siirtosuunta. Reittiennakkohuollot-objektista selviää reitti-tunniste, mutta syöttämällä tunniste erikseen varmistetaan, että reittiennakkohuollot-objekti varmasti kuuluu reittiin jota ollaan muuttamassa. Funktion aluksi tarkastetaan tunnisteiden perusteella reitti- ja reittiennakkohuolto-objektien olemassa oleminen. Mikäli molemmat tunnisteet eivät ole oikein ohjataan käyttäjä 404-sivulle. Jos valittuna siirtosuuntana on ylöspäin, niin pienennetään järjestysnumeroa. Tällöin tarkastetaan, ettei käyttäjä yritä muuttaa järjestysnumeroa pienemmäksi kuin numero yksi. Siirtosuunnan ollessa alaspäin kasvatetaan järjestysnumeroa yhdellä. Kun uusi järjestysnumero on selvillä, haetaan reitiltä reittiennakkohuolto-objekti, jolla on kyseinen numero käytössä. Mikäli objekti löytyy, vaihdetaan objektien järjestysnumerot keskenään ristiin. Objektin puuttuminen saattaa johtua kahdesta syystä: käyttäjä on poistanut keskeltä reittiä ennakkohuollon tai lisännyt reitille ennakkohuollon suuremmalla numerolla kuin suurin edellinen lisättynä yhdellä. Tällöin vain siirrettävän objektin numero päivitetään uudeksi. Objektien järjestysnumeroista voitaisiin poistaa tyhjät välit objekteja poistettaessa ja lisättäessä. Reittejä käyttävien laitteiden sijainneissa tapahtuu erittäin vähän muutoksia. Jonkin laitteen päivittäminen uuteen ja uuden laitteen huollon liittäminen reittiin saadaan hoidettua muokkaamalla vanhaa reittiennakkohuollot-objektia.

Reittienakkohuollot-objektin poistaminen onnistuu painamalla käyttöliittymän reittinäkymästä roskakorin kuvaa. Painaminen aktivoi JavaScript-funktion, joka muodostaa kysymyslaatikon käyttäjälle. Tällä laatikolla varmistetaan onko käyttäjä varma, että haluaa poistaa reitin liitoksen. Käyttäjän vastatessa myöntävästi siirretään käyttäjä osoitteeseen, joka ohjautuu liitosten poisto-funktioon. Parametreina funktiolle annetaan pyyntö ja poistettavan reittiennakkohuollot-tunniste sekä reitti-tunniste. Ennen funktion suoritusta käytetään Djangon permission_required-decorator-ominaisuutta tarkastamaan käyttäjän oikeuksien riittävyys kyseisen funktion käyttä-

miseen (Kuva 15.). Tämän jälkeen jatketaan samoin kuin reittiennakkohuollot-objekteja siirrettäessä eli tarkastetaan reitti- ja reittiennakkohuollot-objektin olemassaolo sekä yhteenkuuluvuus. Molempien ollessa kunnossa suoritetaan reittiennakkohuollot-objektin poisto (engl. delete)-funktio. Poistamisen jälkeen käyttäjä ohjataan takaisin reittinäkymään.

```
@permission_required('ennakkohuolto.delete_reittiennakkohuollot')
def poistaReittiEnnakkohuolto(request, reittitunniste, reittiennakkohuollottunniste):
        profile = request.user.get_profile()
```

Kuva 15. Permission_required decorator

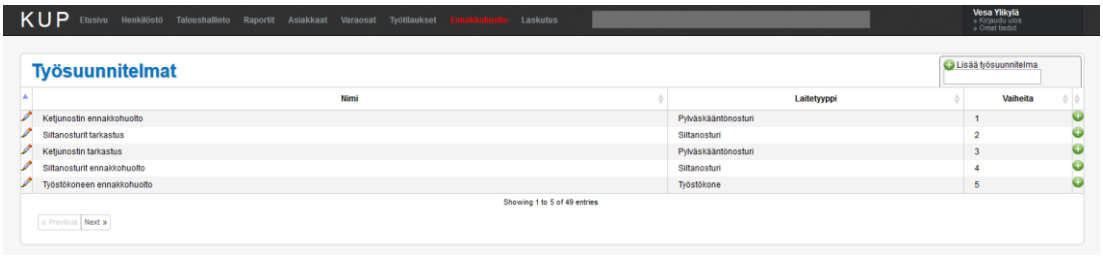
4.2 Vakiotyösuunnitelma

Vakiotyösuunnitelmat ovat ryhmä työvaiheita, jotka suoritetaan määritellyssä järjestyksessä. Työvaiheiden lisäksi vakiotyösuunnitelmiin kuuluvat työssä käytettävät osat. Vakiotyösuunnitelmia varten tarvitaan listaus-, lisäys-, muokkaus- ja hallintanäkymät. Vakiotyösuunnitelman tietorakenteessa tarvitaan viittaus laitetyyppiin, koska erityyppisten laitteiden huollot eroavat toisistaan. Lisäksi laitetyypin perusteella osataan tarjota jo aikaisemmin tehtyä huoltosuunnitelmaa käytettäväksi kun käyttäjä luo ennakkohuoltoa tai työmääräintä samantyyppiselle laitteelle. Laitetyyppi viittauksen lisäksi tarvitaan viittaus työtyyppiin. Työtyyppi kertoo onko kyseessä esimerkiksi ennakkohuolto vai tarkastus. Työtyyppien ja laitetyyppien hallinta tapahtuu järjestelmässä muualta, joten niiden hallintaan ei moduulissa tarvita erillistä ominaisuutta. Näiden viittausten lisäksi tietorakenteessa on nimi ja koodi joilla erotellaan saman työtyypin ja laitetyypin omaavat suunnitelmat toisistaan.

4.2.1 Listausnäkymä

Suunnitelmien listaaminen toteutetaan samalla tapaa kuin reittien listaaminen. Luodaan yksinkertainen HTML-taulukko joka määritellään käyttämään Datatables jQuery-lisäosaa. Taulukkoon tulevat vakiotyösuunnitelman nimi, laitetyyppi sekä painike työsuunnitelman muokkaamiseen siirtymiseen ja toinen painike joka luo uuden työmääräimen käyttäen työsuunnitelmaa. Näiden lisäksi taulukon riveihin lisä-

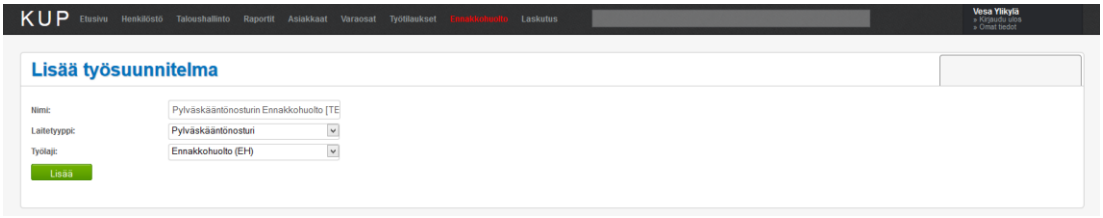
tään painettaessa (onclick)-tapahtuma joka siirtää käyttäjän työsuunnitelman hallintanäkymään (Kuva 16.).



Kuva 16. Työsuunnitelmien listausnäkymä

4.2.2 Lisäys- ja muokkausnäkymä

Suunnitelmien lisäys- ja muokkausnäkymä ei myöskään eroa suuresti reittien vastaavasta. Lomakkeella näytetään kaikki tietorakenteessa olevat kentät. Kaikki kentät ovat myös oletustyyppisiä. Nimi on tekstikenttä. Työ- ja laitetyyppien määrät ovat sen verran pieniä, että niiden valintaan riittää pudotusvalikko. Näkymän funktion rakenne on samanlainen kuin reittien vastaavassa käytetään. Suunnitelmien funktiossa ei ole kuitenkaan tarvetta muokata lomakkeen kenttiä. Samoin automaattiset tietojen oikeellisuuden tarkastukset sopivat suoraan (Kuva 17.).



Kuva 17. Työsuunnitelmien lisäysnäkymä

4.2.3 Hallintanäkymä

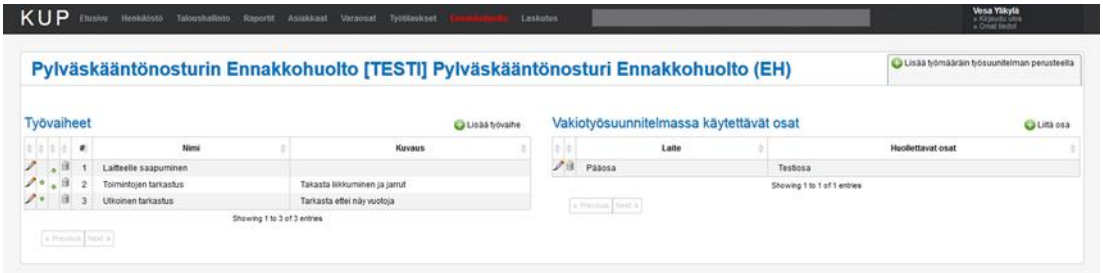
Suunnitelman hallintanäkymä on jaettu kahteen osaan. Vasemmanpuoleisesta sarakkeesta löytyvät työvaiheet. Työvaiheiden listaamista varten luodaan HTML-taulukko, joka määritetään käyttämään Datatables jQuery-lisäosaa (Kuva 18.). Taulukkoon syötetään työvaiheen järjestysnumero, nimi ja kuvaus. Tietojen lisäksi taulukkoon luodaan painikkeet vaiheiden järjestyksen vaihtamiseen, poistamiseen ja

muokkaamiseen. Taulukon yläpuolelta löytyy painike jonka kautta pääsee vaiheiden lisäyslomakkeeseen.

```
<script>
$(document).ready(function() {
    $('.datatable').dataTable( {
        "bSort": false,
         "bFilter": false,
         "bLengthChange": false,
         "bAutoWidth": false,
        "bStateSave": true,
    } );
} );
</script>
```

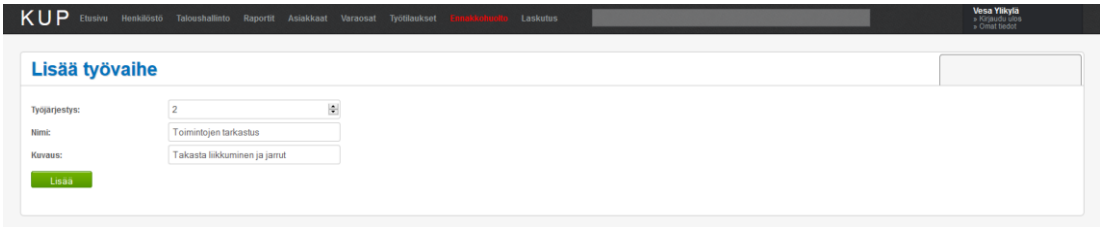
Kuva 18. Datatables jQuery-lisäosan määritteleminen

Oikean puoleisessa sarakkeessa ovat työsuunnitelmaan kuuluvat osat. Joissakin huolloissa tai tarkastuksissa vaihdetaan aina samat osat. Jolloin tämän avulla saadaan siirrettyä nämä osat suoraan työmääräimelle. Tämä vähentää työntekijän kirjausten tekemiseen kuluvaa aikaa. Osien HTML-taulukko määritellään myös käyttämään Datatables jQuery-lisäosaa. Taulukossa on painikkeet liitettyjen osien poistamiseen työsuunnitelmalta ja muokkaamiseen. Näiden lisänä ovat sarakkeet laitteelle ja vaihdettavalle osalle. Laiteella tarkoitetaan eri laitteista koostuvan laitteen osaa. Esimerkiksi nosturin työsuunnitelmalla laite voi olla nosturin moottori ja laitteen huollettava osa moottorin käämit. Liitoksessa on oltava määriteltynä vähintään laite. Jos ainoastaan laite on valittuna tarkoittaa se, että huollossa vaihdetaan koko laite. Osien lisääminen työsuunnitelmaan tapahtuu taulukon yläpuolella olevasta painikkeesta. Sivun toiminnot-alueella on painike, joka aloittaa uuden työmääräimen luomisen käyttäen valittua työsuunnitelmaa.



Kuva 19. Työsuunnitelmien hallintanäkymä

Työvaiheiden lisääminen, poistaminen ja hallinta tapahtuvat samaan tapaan kuin reitin vaiheiden vastaavat toiminnot. Käytettävät funktiot ovat erilaiset, mutta ainoana eroavaisuutena niissä on käytettävät tietomallit, muuttujien nimet ja käytettävä teemapohja. Funktioiden rakenne ja toimintatapa ovat molemmissa samanlaiset (Kuva 20.).



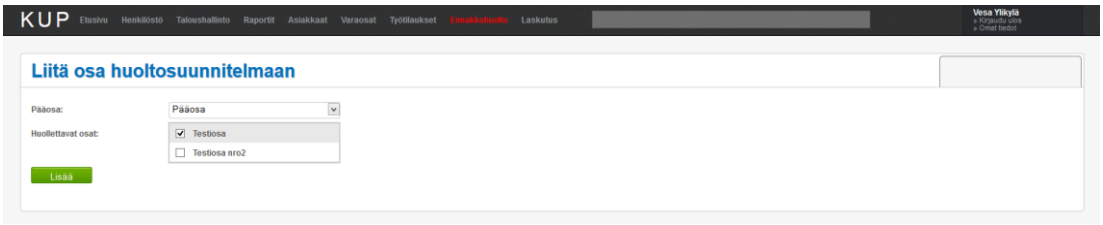
Kuva 20. Työvaiheen lisäysnäkymä

Osien liittämisen ja muokkaamiseen käytettävä lomake eroaa muista lomakkeista hieman toiminnollisuudeltaan. Näkymässä olevan funktion rakenne ja sisältö on samanlainen kuin muissakin tähän asti käsitellyissä. Käyttäjän pyytäessä lisäyslomaketta luodaan lomake lomake-luokan kautta samoin kuin aikaisemminkin. Työsuunnitelman tunniste saadaan URL-osoitteesta, jonka vuoksi kyseinen kenttä on määritelty pois lomakkeelta. Aliosien valintaan käytettävä pudotusvalikko-elementti on määritelty monivalinta-kentäksi. Tämä kenttä muotoillaan selaimessa JavaScriptin avulla uusiksi.

Käyttäjän valittua pudotusvalikosta haluamansa pääosan, aktivoituu JavaScript-tapahtumakäsittelijä kyseiselle toiminnalle. Tapahtumakäsittelijä luo Ajax-pyynnön palvelimelle. Pyynnön mukana lähetään pääosan tunniste. Mikäli pääosan tunniste löytyy tietokannasta palauttaa palvelin kaikki pääosaan kuuluvat osat. Jos pääosalta ei löydy yhtään osaa palauttaa palvelin tyhjän vastauksen. Tunnisteen ollessa virheellinen palauttaa palvelin virheilmoituksen: ”Valittua pääosaa ei löydy järjestelmästä”. Vastauksen sisältäessä osia näytetään ne monivalintalaatikossa. Laatikon valintojen arvoksi annetaan osan tunniste ja näytettäväksi tekstiksi osan nimi sekä osanumero, jos sellainen on määritelty (Kuva 21.)

Valintalaatikoista käyttäjä valitsee haluamansa ja tallentaa lomakkeen. Muokattaessa suunnitelmaan liitettyä osaa, luodaan aliosien monivalintalaatikko teemapohjalla.

Näin vältytään yhdeltä turhalta Ajax-pyynnöltä. Tallennettaessa käytetään Djangon automaattisesti tietomallin perusteella luomia tarkistusehtoja. Ehdot ovat erittäin yksinkertaiset: työsuunnitelma ja pääosa ovat pakollisia ja niiden on oltava olemassa tietokannassa. Aliosat ovat vapaaehtoisia, mutta valittujen alaosien tulee olla olemassa.



Kuva 21. Osien liittäminen työsuunnitelmaan

4.3 Ennakkohuollot

Ennakkohuollot ovat tämän moduulin keskeisin osa. Niiden avulla luodaan uusia työmääräimiä määriteltyjen ehtojen perustella. Nimestä huolimatta ennakkohuoltoja voi käyttää myös muun tyyppisten töiden tekemiseen. Toinen huollon lisäksi paljon käytetty työtyyppi on tarkastukset.

4.3.1 Tietomalli

Ennakkohuoltojen malli koostuu monista kentistä. Pakollisia ja aina tarvittavia tietoja ovat työsuunnitelma, laite, deadline, ennakkojakso, huoltojen intervalli ja huoltojen laskentatapa. Laskentatapoja on kaksi erilaista. Oletuksena lasketaan seuraavan huollon ajankohta edellisen huollon deadlinen perusteella. Toinen vaihtoehto on laskea seuraavan huollon ajankohta käyttämällä hyväksi edellisen huollon valmistumispäivää. Työsuunnitelman perusteella tiedetään minkä tyyppinen työ on kyseessä sekä liitetään työn vaiheet ja tarvittavat materiaalit työmääräimelle. Laitetietoa tarvitaan huollon kohdistamiseen laitteelle. Järjestelmässä on laitenäkymä, jossa näkyvät laitteen tiedot, tehdyt/keskeneräiset työt ja liitetyt osat. Laitetiedon avulla saadaan laitenäkymään näkyviin laitteelle kohdistetut ennakkohuollot. Laitteen avulla saadaan työmääräimelle asiakastieto. Deadline on päivämäärä johon mennessä huollon tulee olla suoritettu. Deadlinen, intervallin ja seuraavan huollon laskentatavan perusteella

lasketaan huoltojen työmääräimeksi nostamispäivämäärät. Ennakkojakso määrittää, montako päivää ennen deadlinea ennakkohuollosta luodaan työmääräin.

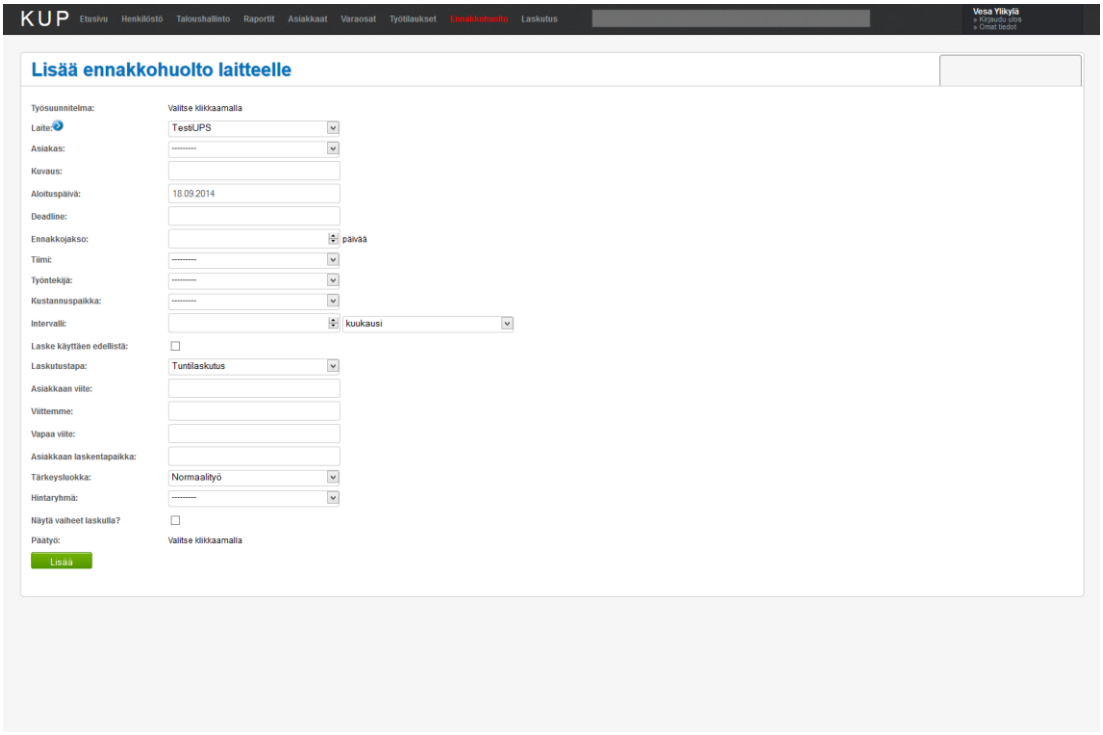
Pakollisten kenttien lisäksi on vapaaehtoisia kenttiä joiden tiedot ylittävät muualta tulevat tiedot. Muualta tulevan tiedon ylittäviä kenttiä ovat laskuntyyppi, kustannuspaikka, hintaryhmä ja viitetiedot. Laskun tyyppi määrittää työn laskutustavan. Hintaryhmä vaikuttaa työlle tulevien hintojen syntymiseen. Kustannuspaikka kohdistaa laskun yrityksen toimintopaikalle. Viitetiedot ovat laskulle tulevia viitetietoja. Kaikki edellä mainitut kentät saavat oletuksena arvonsa laitteen asiakas-objektilta.

Tiimi, työntekijä, tärkeysaste ja päätyö ovat työmääräimen vapaaehtoisia kenttiä. Tiimin ja työntekijän valinnat vaikuttavat ennakkohuoltojen näkymiseen niissä valituille käyttäjille. Ennakkohuollot näkyvät kohdistetulle käyttäjille korostettuina. Luotavat työmääräimet tulevat automaattisesti näkyviin käyttäjien etusivulle ja ovat korostettuja töiden listauksessa. Tärkeysasteella voidaan nostaa ennakkohuolto tehtäväksi ennen muita. Esimerkiksi asiakkaan tuotanto on vuodessa viikon poissa toiminnasta, joten niiden laitteiden huollot täytyy saada sen viikon aikana tehtyä. Päätyö on työmääräin, johon avattavat työmääräimet kuuluvat. Esimerkiksi on voitu tehdä asiakkaan kanssa sopimus, että kaikki heidän laitteidensa tarkastukset laskutetaan kerralla. Tällöin kaikki asiakkaan laitteiden tarkastukset liitettäisiin samaan päätyöhön. Käyttäjälle näkyvien kenttien lisäksi kerätään järjestelmää varten tietoja objektin luojasta ja muokkaajasta sekä näiden tapahtumien ajankohdista.

4.3.2 Lisäys- ja muokkausnäkymä

Ennakkohuoltojen lisäämiseen ja muokkaamiseen käytettävän näkymä funktion rakenne on samanlainen kuin muissakin lomakkeiden käsittelyissä. Näkymän funktiossa määritetään laite-kenttä datatableSelect-kentäksi. Tämän lisäksi teemapohjassa (engl.template) luodaan tarvittavat jQuery-rakenteet työsuunnitelmien laitteelle hakemista varten. Tällä varmistetaan, ettei käyttäjä valitse työsuunnitelmaa, joka ei sovellu valitulle laitteelle. Mallin perusteella luotavat tietojen automaattiset tarkistukset riittävät. Vaihtoehtona olisi luoda mallille itse tarkastukset, mutta erilaisten kombinaatioiden määrän huomioiden, ei siinä olisi järkeä. Automaattiset säännöt riittävät

pitämään huolen, ettei käyttäjä pysty sekoittamaan järjestelmää. Väärin tai huonosti täytetyistä ennakkohuolloista ei avaudu työmääräimiä ollenkaan tai ne ovat täysin käyttökelvottomia työmääräimiä. Avautuvat työmääräimet saavat tilakseen ”odottaa hyväksyntää”-merkinnän. Työnjohtaja muuttaa työt ”hyväksytty”- tilaan, jos tiedot ovat oikein. Jos työmääräimen tiedot ovat väärin niin työnjohtajan tehtäväksi tulee korjata virheet ennakkohuollosta. Alta löytyy kuva ennakkohuoltojen lisäämiseen ja muokkaamiseen käytettävästä lomakkeesta (Kuva 22.).



Kuva 22. Ennakkohuoltojen lisäyslomake

4.3.3 Kalenterinäkymä

Tulevien ennakkohuoltojen tarkastelemista varten on kalenterinäkymä. Kalenteri näkymässä näytetään tulevat ennakkohuollot vuodeksi eteenpäin. Tietojen näyttämiseen käyttöliittymässä käytetään Gantt Chart jQuery-lisäosaa. Näkymän funktiossa haetaan kaikki aktiiviset ennakkohuollot, jotka käyttäjällä on oikeus nähdä. Näille ennakkohuolloille lasketaan ajankohdat.

Ennakkohuoltojen toteutumisaikoja laskemista varten haetaan kaikki aktiiviset ennakkohuollot, jotka käyttäjällä on oikeus nähdä. Ennakkohuollot käydään läpi yksi

kerrallaan. Ensimmäisenä tarkistetaan lasketaanko seuraavan huollon ajankohta käyttäen hyväksi edellisen huollon päättymisajankohtaa. Tällaisessa tapauksessa etsitään tietokannasta kaikki työmääräimet, joka perustuvat kyseiseen ennakkohuoltoon. Jos työmääräintä ei löydy käytetään ennakkohuollolle syötettyä deadline päivämäärää. Deadlinen päivämäärän puuttuminen johtaa huollon ajankohdan laskemisen epäonnistumiseen. Mikäli järjestelmästä löytyy yksi tai useampi valmis ennakkohuoltoa käyttävä työmääräin niin käytetään viimeisimpänä valmistuneen työmääräimen valmistumisajankohtaa apuna seuraavan huollon laskemiseen. Deadline saadaan lisäämällä edellisen työmääräimen valmistumispäivämäärään ennakkohuollon intervalli. Intervallin puuttuminen johtaa myös laskemisen epäonnistumiseen. Ennakkohuollot jotka eivät käytä edellistä valmistumisajankohtaa laskemiseen käyttävät deadlinea suoraan seuraavan huollon deadlinena. Deadline-tiedon puuttuminen johtaa laskemisen epäonnistumiseen.

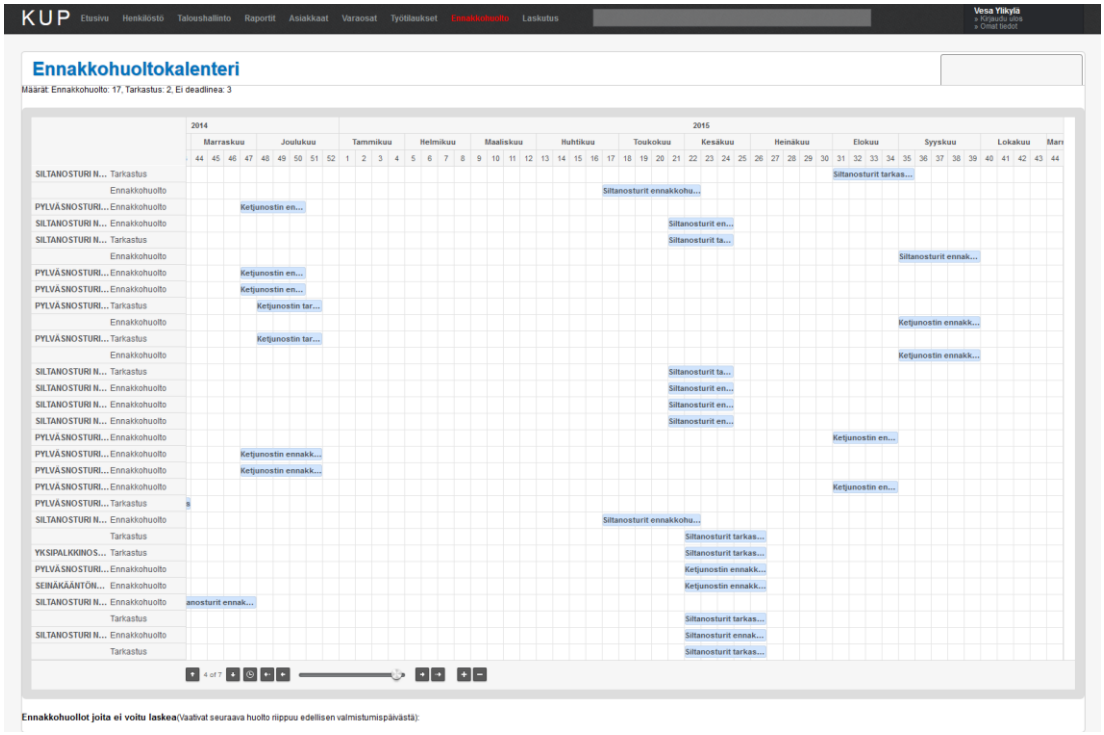
Kun seuraavalle ennakkohuollon työmääräimelle on saatu deadline, tarkastetaan sen kuuluminen vuoden aikajanalle. Kalenterissa näytetään tulevat ennakkohuollot vuoden verran eteenpäin. Jos deadline mahtuu aikajanalle, lasketaan huollolle nousemispäivämäärä vähentämällä deadlinesta ennakkoaika. Ennakkoajan puuttuminen johtaa laskemisen epäonnistumiseen. Deadline ja nousemispäivämäärä muutetaan sopivaan formaattiin Gantt Chart jQuery-lisäosaa varten ja laitetaan taulukkoon (engl. dict) ennakkohuolto-objektin kanssa (Kuva 23.). Aina lisättäessä uusi ennakkohuolto onnistuneet taulukkoon kasvatetaan myös muuttujan arvoa joka pitää kirjaa kuinka monta minkäkin tyyppistä huoltoa on laskettu.

```
tiedot['eh'] = eh
dl = str(int(str(time.mktime(eh.deadline.timetuple()))[:-2] + '000'))
np = str(int(str(time.mktime(datetime.datetime.combine((eh.deadline-timedelta(days=eh.ennakkoaika)), datetime.time(23,59,59)).timetuple()))[:-2] + '000'))
tiedot['ajat'] = [[dl, np, eh.deadline, eh.deadline-timedelta(days=eh.ennakkoaika)],]
```

Kuva 23. Ennakkohuollon ja ajankohdan lisääminen

Seuraavaksi toistetaan uuden deadlinen laskeminen käyttäen hyväksi aiemmin laskettua deadlinea. Jos uusi laskettu deadline osuu myös aikajanalle, lasketaan sille nousemispäivä ja lisätään myös nämä tiedot taulukkoon. Tämä prosessi toistetaan kunnes uusi deadline ei enää mahdu aikajanalle. Tämän jälkeen siirrytään seuraavaan ennakkohuoltoon ja aloitetaan prosessi alusta. Mikäli ennakkohuollon ensimmäisen ilmestymisen laskeminen epäonnistuu, lisätään ennakkohuollo-objekti ja epäonnistumisen

syy epäonnistuneet-taulukkoon (engl. array). Näkymän funktiolta teemapohjalla viedään onnistuneesti lasketut, epäonnistuneet laskemiset ja ennakkohuoltojen määrät. Ganttin Chart-elementin alapuolelle tulostetaan listaksi epäonnistuneet luonnit. Epäonnistuneista näkyviin tulostetaan nimi ja syy. Epäonnistuneen ennakkohuollon nimi toimii linkkinä kyseisen ennakkohuollon muokkaukseen. Näin käyttäjä pääsee helposti korjaamaan tarvittavat virheet.



Kuva 24. Ennakkohuoltokalenteri

4.3.4 Työmääräimien automaattinen luominen

Ennakkohuolloista tulevat uudet työmääräimet luodaan automaattisesti vuorokauden vaihtuessa. Tämä tapahtuu palvelimen ajoitettujen tehtävien (engl. cron) avulla, joka ajaa erillisen Python tiedoston. Tätä tiedostoa käyttäjä ei pysty suorittamaan käyttöliittymästä. Tiedostoa ei myöskään ajateta Djangon kautta. Tiedoston alussa kuitenkin tuodaan Djangon asetukset ja tarvittavat moduulit.

Työmääräimien luominen tapahtuu osittain samalla tavalla kuin kalenterinäkymän luominenkin. Aluksi haetaan kaikki aktiiviset ennakkohuollot. Nämä käydään kaikki yksitellen läpi. Ennakkohuollon nousemispäivä ja deadline lasketaan samalla tavalla kuin kalenterinäkymää luotaessa. Kun ennakkohuollolle on saatu aloituspäivämäärä

ja deadline, tarkastetaan että avoimena ei ole yhtään samaa ennakkohuoltoa käyttävää työmääräintä. Toinen tarkistettava asia on työmääräimen avauspäivämäärä, jonka on oltava pienempi tai yhtä suuri kuin nykyinen päivämäärä. Kun edellä mainitut ehdot täyttyvät, niin siirrytään eteenpäin luomaan työmääräintä.

Työmääräintä varten haetaan käyttäjä-objekti joka on luotu ennakkohuoltojen luomista varten. Työmääräimen avaajaksi merkitään tämä käyttäjä. Tämän avulla nähdään järjestelmästä, että työmääräimet ovat järjestelmän itsensä avaamia. Haetaan myös status-objekti, jonka kuvauksena on odottaa hyväksyntään. Automaattisesti luodut työt on työnjohtajan merkittävä hyväksytyiksi ennen kuin työntekijät voivat alkaa niitä tekemään. Oletuksena työmääräimen viitetiedot, laskutustyyppi ja hintaryhmä otetaan laitteen asiakas-objektilta. Ennakkohuollolta löytyy samat kentät, joilla ylitetään asiakkaalla määritetyt oletusarvot. Seuraavaksi luodaan työmääräinobjekti, jolle syötetään arvoina haetut tiedot ja ennakkohuollolle määritellyt tiedot. Jos työmääräintä luotaessa ei ilmene virheitä tallennetaan työmääräin. Luodulle työmääräimelle luodaan erillinen status-objekti. Tämän tarkoituksena on pitää kirjaa työn tilojen vaihdoksista. Seuraavaksi päivitetään ennakkohuollon deadline lisäämällä työmääräimelle määritettyyn deadlineen ennakkohuollolla määritelty intervalli.

Mikäli ennakkohuollolle on määriteltynä ennakkotyösuunnitelma haetaan siihen kuuluvat vaiheet. Nämä vaiheet lisätään luodulle työmääräimelle käyttäen luodun työmääräimen, ennakkohuollon ja työvaiheen tietoja. Jos työvaiheita lisätessä tapahtuu virheitä, jotka estävät työvaiheiden tallentamisen työmääräimelle, poistetaan jo lisätyt vaiheet sekä työmääräin. Poistamisen lisäksi luodaan lokimerkintä, jossa kerrotaan miksi kyseistä työmääräintä ei saatu lisättyä. Ennakkotyösuunnitelmaan liitetyt osat lisätään työmääräimelle materiaaleiksi, jos vaiheiden lisääminen onnistui. Osien lisääminen tapahtuu samalla tavalla kun vaiheidenkin lisääminen. Myös virheiden käsittely tapahtuu samalla tavalla.

5 JOHTOPÄÄTÖKSET JA TULEVAISUUS

Käyttöliittymän toteutuksessa käytetyt tekniikat olivat tuttuja entuudestaan. Erityisesti HTML- ja CSS-tekniikat olivat erittäin hyvin hallussa. Niiden osalta ei myöskään moduulia luotaessa tullut opittua mitään uutta johtuen yksinkertaisista sivujen rakenteista. JavaScript-kieli ja JavaScript-kirjasto jQuery olivat tekniikoita joiden kanssa on tullut aiemmin oltua tekemisissä. Datatables ja Gantt Chart jQuery-lisäosat olivat ihan uusia. Niiden käytöstä tuli opittua melko paljon. Datatables-lisäosa on sellainen, jota tulee varmasti käytettyä myös jatkossa paljon www-sivuilla taulukoissa.

Palvelinpäässä käytetystä Python-ohjelmointikielestä ei ollut aikaisemmin kokemusta yhtään. Nimi oli kyllä tuttu mutta syntaksi oli tuntematon. Vaikka moduulissa käytetyt rakenteet olivat pääosin yksinkertaisia, niin koen Pythonin olevan nyt melko hyvin hallussa. Django-ohjelmistokehyksestä en ollut ennen kuullutkaan. Sen toimintaperiaatteen ymmärtäminen oli melko helppoa. Ennen PHP-ohjelmointikieltä paljon käyttäneenä ja nyt Djangoon tutustuneena on Django ehdottomasti valintani kun seuraavaa www-sovellusta lähden kehittämään.

Moduulin valmistuttua järjestelmässä on yksi ominaisuus enemmän, josta on ainakin hyötyä ainakin asiakasyritykselle. Tuotteistuksen näkökulmasta moduulista on hyötyä tapauksissa, joissa suoritetaan töitä ennalta määriteltyinä aikoina. Liitettäessä moduulia järjestelmässä oleviin moduuleihin tuli myös pieniä muutoksia. Näistä muutoksista ehkä huomattavin on ennakkotyösuunnitelmat työmääräimellä. Työsuunnitelmia pystyy käyttämään myös normaalisti töitä avatessa. Tämä mahdollistaa samanlaisten töiden työvaiheiden ja materiaalien lisäämisen työlle nopeammin.

Tulevaisuudessa moduuliin voidaan kehittää ominaisuus, joka loisi uusia ennakkohuoltoja itse. Nämä ennakkohuollot tarvitsisivat käyttäjän hyväksynnän ennen kuin niiden perusteella alettaisiin luomaan uusia työmääräimiä. Tällaista ominaisuutta varten järjestelmään tarvitaan enemmän tietoa laitteisiin tulleista vioista. Tietojen perusteella pystyttäisiin päättelemään vian esiintymisen tiheys. Tiheyden perusteella avattaisiin uusi huolto ennen kuin vika ehtisi ilmestyä laitteelle ja tuotanto mahdollisesti katkeaisi. Tehtyjen töiden tietojen perusteella järjestelmä voisi myös ehdottaa mah-

dollisia päivitys- tai parannusehdotuksia. Tarkastelemalla samanlaisille laitteille tehtyjä parannuksia voisi järjestelmä näyttää työnjohtajalle, että tällaisille laitteille voisi tehdä samat toimenpiteet. Työnjohtaja voisi tämän jälkeen olla yhteydessä asiakkaaseen ja ehdottaa parannuksia.

Moduuliin voitaisiin myös lisätä muita ennakkohuoltojen työmääräimeksi nousemisehtoja. Esimerkiksi laitteelta tulisi järjestelmään käyttötunnit ja määritellyn tuntimäärän jälkeen avautuisi automaattisesti ennakkohuolto. Tällaista voitaisiin yksinkertaisimmillaan soveltaa esimerkiksi yrityksen autojen huoltamiseen oikein väliajoin. Työntekijä kirjaisi järjestelmään autolla ajetut kilometrit aina ajon jälkeen. Tällöin järjestelmä pysyisi kokoajan perillä kilometreistä.

LÄHTEET

KP-ServiceParner Oy:n www-sivut. Viitattu 22.09.2014. http://kp-servicepartner.com/

Python Software Foundation. What is Python? Executive Summary. Viiattu 23.09.2014. https://www.python.org/doc/essays/blurb/

Python Software Foundation. General Python FAQ. Viitattu 19.09.2014. https://docs.python.org/3/faq/general.html

Python Software Foundation. History and License. Viitattu 19.09.2014. https://docs.python.org/3.2/license.html

Python Software Foundation. What's New in Python. Viitattu 23.09.2014. https://docs.python.org/3.2/whatsnew/index.html

Python Software Foundation. Comparing Python to Other Languages. Viitattu 23.09.2014. https://www.python.org/doc/essays/comparisons

Digi Wiki. Python Garbage Collection. Viitattu 23.09.2014. http://www.digi.com/wiki/developer/index.php/Python_Garbage_Collection

Python Software Foundation. Design and History FAQ. Viitattu 19.09.2014. https://docs.python.org/3/faq/design.html

Django Web Framework:n www-sivut 2014. Viitattu 19.09.2014. https://www.djangoproject.com/

Python wiki. Web Frameworks. Viitattu 23.09.2014. https://wiki.python.org/moin/WebFrameworks

The Django book. Chapter 1: Introduction to Django. Viitattu 19.09.2014. http://www.djangobook.com/en/2.0/chapter01.html

w3schools. HTML Introduction. Viitattu 19.09.2014. http://www.w3schools.com/html/html_intro.asp

W3C. HTML & CSS. Viitattu 19.09.2014. http://www.w3.org/standards/webdesign/htmlcss

w3schools. CSS Introduction. Viitattu 19.09.2014. http://www.w3schools.com/css/css_intro.asp

w3schools. JavaScript Introduction. Viitattu 19.09.2014. http://www.w3schools.com/js/js_intro.asp

Java. How is JavaScript different from Java? Viitattu 19.09.2014. http://www.java.com/en/download/faq/java_javascript.xml

Douglas Crockford. 2001. JavaScript: The World's Most Misunderstood Programming Language. Viitattu 19.09.2014.
http://www.crockford.com/javascript/javascript.html

jQuery:n www-sivut. Viitattu 19.09.2014. http://jquery.com/

Laine, H. 2011. Ohjelmistoarkkitehtuurit, syksy 2012.
www.cs.helsinki.fi/u/aptuovin/arkkit/s12/Ohjelmistoarkkitehtuurit_3.pdf

Johdatus tietojärjestelmiin. Viitattu 23.09.2014.
http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien_kaytto_ja_kehittaminen/johdatus_tietojarjestelmiin/kehittamistyon_vaiheet_ja_elikaarimallit/kehittamistyon_vaiheet_ja_elinkaarimallit_asia.htm

Paakki J. 2011. Ohjelmistojen vaatimusmäärittely.
www.cs.helsinki.fi/u/paakki/Vaatimus-11-Luentokalvot-1.pdf