Antero Tanskanen

Parametri-työkalujen kehittäminen PlanMill-toiminnanohjausjärjestelmässä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

4.5.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Antero Tanskanen Parametri-työkalujen kehittäminen PlanMill-toiminnanohjausjärjestelmässä 59 sivua 4.5.2014
Tutkinto	Tietotekniikan insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	lehtori Simo Silander senior consultant Marjukka Niinioja lehtori Jussi Alhorinne

Insinöörityössä oli tavoitteena kehittää PlanMill-toiminnanohjausjärjestelmän konfiguroinnissa ja räätälöinnissä käytettävien parametrien käsittelyprosessia. Erityisesti tavoitteena oli nopeuttaa, yksinkertaistaa ja tehdä turvallisemmaksi sen hetkistä monivaiheista ja asiantuntijuutta vaativaa parametrien konfigurointiprosessia. Projektin tilaaja oli PlanMill Oy.

Projektissa käytiin ensin läpi yrityksen sisällä ehdotettuja kehitysideoita ja kerättiin dataa tyypillisimmistä asiakkaiden tilaamista räätälöintitöistä alkuvuonna 2013. Aineistoa analysoitiin ja sen pohjalta pääteltiin parametrien konfigurointiprosessin ongelmakohdat, jonka jälkeen suunniteltiin prototyyppejä, joiden pohjalta suunniteltiin tuotteen kehitysjono (*Product backlog*).

Projektin toteutusosuudessa lähdettiin toteuttamaan Scrum-projektinhallinnan periaatteita noudattaen tuotteen kehitysjonoon listattuja ominaisuuksia. Projektin aikana tuotteen kehitysjono muuttui useita kertoja, kun osa ominaisuuksista todettiin liian suuritöisiksi toteutettavaksi annetun ajan puitteissa ja uusia parempia ideoita syntyi.

Lopputuloksena syntyi useita uusia ominaisuuksia ja toimintoja, joiden koettiin huomattavasti helpottavan parametrien käsittelyä. Tärkeimpiä toteutettuja ominaisuuksia olivat parametrien hakutyökalut sekä vakioparametrien tarkastelu- ja kopiointimahdollisuus suoraan asiakkaiden PlanMill-instansseissa, jolloin näitä varten ei enää tarvitse kirjautua erilliseen, niin sanottuun jaettujen parametrien PlanMill-instanssiin.

Avainsanat	parametrit, konfigurointi, saas, räätälöinti, työkalut, planmill, psa, toiminnanohjaus, erp, crm

Author(s) Title Number of Pages Date	Antero Tanskanen Improving Parameterization Tools In PlanMill PSA 59 pages 4 May 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Principal Lecturer Marjukka Niinioja, Senior consultant Jussi Alhorinne, Lecturer

Subject of this thesis was to improve parameterization processes in PlanMill PSA where parameters are used to configure and customize the system to fit the customers' needs. The aim was especially to make the originally multi-stepped and expertise involving parameterization processes faster, simpler and safer. This project was ordered by PlanMill Ltd.

The project began by going through all the existing improvement ideas in the company and researching what were the most commonly ordered customization tasks during the first half of the year 2013. Using the gathered data, were then figured out the most relevant parts of the process that needed enhancement. After that, different prototypes were designed, out of which one were chosen to be implemented in the project.

The implementation phase was carried out by using Scrum project management. During the implementation phase, the product backlog changed multiple times after some features were discovered too large to be implemented in the given time frame and new better ideas came up.

As an end product, multiple new tools and features were implemented, which employees felt had a great impact on workflow of working with parameters. The most important features that were implemented were search tools for parameters and the ability to view and copy default parameters in customer's instances, leaving out the need to ever again log in to a so called 'shared' instance to inspect the default parameters.

Keywords	parameters, configuration, saas, customization, tools, planmill, psa, erp, crm

Sisällys

1 Johdanto 1

2 PlanMill-järjestelmä 2

2.1 Käyttöliittymä 3

2.1.1 Listanäkymä 3

2.1.2 Yhteenvetosivu 4

2.1.3 Lomakenäkymä 5

2.1.4 Raporttinäkymä 5

2.2 Järjestelmän arkkitehtuuri 6

2.2.1 Parametrit PlanMill-järjestelmässä 10

2.2.2 Kieliasetusten ja enumeraatioiden määrittäminen 11

2.2.3 Käyttöoikeuksien hallinta 12

3 Parametrien konfigurointiin liittyvien haasteiden kartoitus 13

3.1 Räätälöinnit 13

3.1.1 Yleisimmät parametrien muokkaamiseen liittyvät toimenpiteet 13

3.1.2 Tyypillisen räätälöintitoimenpiteen vaiheet työn alkutilanteessa 15

3.2 Käyttäjien kokemat ongelmat 17

3.3 Aikaisemmin esitetyt parannusehdotukset 18

3.3.1 Jaettujen parametrien näkyminen asiakasinstansseissa 18

3.3.2 Hakutyökalu parametreille 19

3.3.3 Staattisen analyysin työkalut 19

3.3.4 Käyttöliittymäkenttien konfigurointi muokkauslomakkeella 19

3.4 Kehityskohteet 20

4 Parannusten toteuttaminen 21

4.1 Projektissa käytetyt menetelmät 21

4.1.1 BDD (*Behaviour-driven Development*) 22

4.1.2 Scrum ja projektitiimi 23

4.1.3 Kehitysympäristö 24

4.2 Prototyypit 25

4.2.1 Prototyyppi 1 – Lomakekenttäkohtainen muokkausnäkymä 25

4.2.2 Prototyyppi 2 – Lomakekohtainen muokkausnäkymä 27

4.2.3 Prototyyppi 3 – Listanäkymäpohjaan perustuva lomakekohtainen muokkausnäkymä 30

4.2.4 Prototyyppi 4 – Yksinkertaisempi toteutusvaihtoehto 31

4.2.5 Päätös projektissa toteutettavista toiminnoista 33
4.3 Arkkitehtuuriset päätökset uusiin toimintoihin liittyen 36
4.3.1 Jaettujen parametrien lukeminen 36
4.3.2 Tekstihaun toteuttaminen 37
4.4 Toteutetut ominaisuudet 38
4.4.1 Jaettujen parametrien näkyminen Parametrit-välilehdellä 39
4.4.2 Portaalihaun käyttäminen myös parametrien hakemiseen 44
4.4.3 Personoitavat suodattimet ja tekstihaku Parametrit-välilehdelle 45
4.4.4 Muut parannukset 48
4.5 Tuotantoon siirtyminen 50
4.5.1 Koodinkatselmointi ja manuaalinen testaus 50
4.5.2 Tuotantoon siirtymisen jälkeen havaitut ongelmat 51
4.6 Projektin jälkeen 53
4.6.1 Tyypillisen räätälöintitoimenpiteen vaiheet työn lopputilanteessa 53
4.6.2 Tulevaisuudessa toteutettavia asioita 56
5 Yhteenveto 57
Lähteet 59

Käsitteistö

Entiteetti	*Entiteetti (Entity)* on relaatiotietokantoihin liittyvä käsite, joka tarkoittaa asiaa, josta tallennetaan tietoa. Entiteetti edustaa aina jotakin tyyppiä [1, s. 69]. PlanMill-järjestelmässä tällaisia ovat esimerkiksi projekti, myyntitilaus, tuntikirjaus ja palvelupyyntö.
Instanssi	*Instanssilla* (*instance)* tarkoitetaan objektin ilmentymää tai toteutusta. Tämän työn yhteydessä sillä viitataan usein ilmentymään PlanMill-sovelluksesta, johon liittyy omassa osoitteessa toimiva verkkopohjainen käyttöliittymä sekä tietokanta, joka sisältää kaiken kyseisen instanssin sovellusdatan sekä instanssikohtaiset parametrit.
Merkintäkieli	*Merkintäkieli (kuvauskieli) (engl. markup language)* on formaali kieli, jolla kuvataan tekstin tai datan rakennetta tai esitystapaa metainformaatiolla [4].
Moduuli	Itsenäinen ohjelmisto-osa, joka voidaan aktivoida PlanMill-järjestelmään tuoden lisätoiminnallisuutta. Moduuleita (tai tuotteita) ovat esimerkiksi tuoterekisteri, kehityskeskustelut, poissaolojen hallinta.
Multitenanttinen	Multitenanttinen (*multi-tenant*) sovellusarkkitehtuuri on SaaS-mallille tyypillinen tapa palvella useita asiakkaita käyttäen yhtä sovelluspalvelimella pyöritettävää sovellusinstanssia.
Parametri	Parametri tarkoittaa tämän työn yhteydessä Parametri-entiteettiä, joita käytetään PlanMill-sovelluksen konfiguroimiseen sekä räätälöimiseen.
SaaS	*Software as a Service (SaaS)* on malli sovellusten myyntiin ja levitykseen, jossa ohjelmisto toimitetaan käytön mukaan laskutettavana palveluna. Tyypillistä SaaS-palvelulle on sen käyttö verkon yli verkkoselainta käyttäen. [2, s. 10.]

1 Johdanto

Opinnäytetyön toimeksiantaja PlanMill Oy on kotimainen pk-yritys, joka kehittää PlanMill-nimistä verkkopohjaista asiantuntija- ja projektiyritysten toiminnanohjausjärjestelmää. PlanMill-järjestelmä toimii sovellusvuokrauksena tarjottavana SaaS-palveluna *(Software as a Service),* ja sen modulaarisuuden ansiosta siihen voidaan aktivoida juuri asiakkaan tarpeita vastaavat toiminnallisuudet. PlanMill-järjestelmän valtti kilpailijoihin nähden ovatkin erityisesti sen edistyneet räätälöintimahdollisuudet, joilla ohjelmisto voidaan sovittaa asiakkaan erityisvaatimuksiin.

PlanMill-järjestelmän kehityksessä keskeisenä teemana on tänä vuonna yksinkertaistaminen. Sovelluksen toiminnallisuutta laajennetaan jatkuvasti, ja se on olemassaolonsa aikana käynyt läpi useampia laajoja muutoksia; siten osa koodista on vanhaa, osa uutta. On ajauduttu tilanteeseen, jossa sovelluksen hallinnoiminen on joiltain osin aikaa vievää ja vaatii asiantuntemusta. Asiakasmäärän tasainen kasvu tarkoittaa sitä, että ylläpidollisia tehtäviä on jatkuvasti enemmän ja niistä tulisi suoriutua entistä nopeammin.

Asiakkaiden tilaamat räätälöintityöt muodostavat suuren osan PlanMill Oy:n henkilökunnan arjesta. Nykytilassa räätälöintien tekeminen asiakkaan PlanMill-järjestelmään edellyttää usein teknistä asiantuntemusta.

Tämän opinnäytetyön tarkoituksena oli kehittää ratkaisuja, jotka helpottavat ja tehostavat PlanMill-järjestelmässä asiakaskohtaisten konfigurointien ja räätälöintien tekemiseen käytettävien parametrien käsittelyä.

Opinnäytetyön vaiheita oli ensin selvittää tyypillisimpiä räätälöintitehtäviä ja räätälöintiprosessin ongelmallisuutta. Kerätyn datan valossa suunniteltiin erilaisia ratkaisuja, joilla parametrien kanssa työskentelemistä pystyttäisiin mahdollisimman hyvin tehostamaan opinnäytetyön tekemiseen varatun ajan puitteissa. Tämän jälkeen lähdettiin toteuttamaan parhaaksi katsottua ratkaisua. Toteutusosuuden jälkeen arvioitiin, miten hyvin tavoitteisiin päästiin.

Työn aikana tehtiin yhteistyötä toisen Metropolian opiskelijan kanssa, joka teki samaan aikaan opinnäytetyötä kieliasetusten hallintatyökalujen kehittämisestä PlanMill-järjestelmässä. Näissä kahdessa työssä on yhtäläisyyksiä, sillä parametrien ja kieliasetusten

hallinta on samankaltaista PlanMill-järjestelmässä ja osa lopullisista ratkaisuista kosketti molempia alueita.

2 PlanMill-järjestelmä

PlanMill-järjestelmä pohjautuu alun perin Nokian vuonna 1988 kehittämään Nokia Planner -nimiseen projektinhallintatyökaluun. Vuoteen 1992 mennessä työkalun ympärille oli kehittynyt oma tuoteperhe, joka lanseerattiin nimellä Visual Planner ja tätä kehittämään perustettiin uusi yritys, ViSolutions Oy. Vuosien saatossa tuoteperhettä kehitettiin uusimpia IT-alan innovaatioita, kuten WAP-tekniikkaa (*Wireless Application Protocol*), hyödyntäen, ja samalla yritys koki useampia omistussuhdemuutoksia. Vuonna 2001 perustettiin PlanMill Oy, jolloin lanseerattiin myös yrityksen fokuksessa oleva uusi asiantuntijayritysten toiminnanohjausjärjestelmä. Vuonna 2006 yritys siirtyi Westend ICT Plc:n omistuksesta nykyisen johdon omistukseen. [11.]

Vuosien varrella järjestelmästä on kehittynyt monipuolinen kokonaisuus, johon on mahdollista aktivoida lukuisia tuotemoduuleita eri käyttötarkoituksiin, kuten asiakkuuksien-, projektien-, talouden-, henkilöstön- ja tuotteiden hallintaan. Lisäksi PlanMill-järjestelmään voidaan integroida muun muassa eri taloushallinnon- ja sähköisen laskutuksen järjestelmiä.

Asiakkaat tilaavat PlanMill-sovelluksen suurimmaksi osaksi niin sanottuna SaaS-palveluna (*Software as a Service*), eli sovellus toimitetaan pilvipalveluna, jolloin sitä käytetään verkkoselaimella ja sen käytöstä maksetaan käytön laajuuden mukaan. PlanMill on mahdollista tarpeen tullen asentaa myös asiakkaan omaan tuotantoympäristöön niin sanottuna On-premise-asennuksena.

Järjestelmää käyttäviä yrityksiä on yli sata ja sovelluksen käyttäjiä yli 20 000, jotka sijaitsevat 25 eri maassa. Asiakasyritykset ovat lähinnä erilaisia projekti- ja asiantuntijapalveluita tarjoavia yrityksiä. Tähän joukkoon kuuluu esimerkiksi taloushallinnon, sovelluskehityksen, lakitieteen ja mediatuotannon sektoreilla toimivia yrityksiä [7].

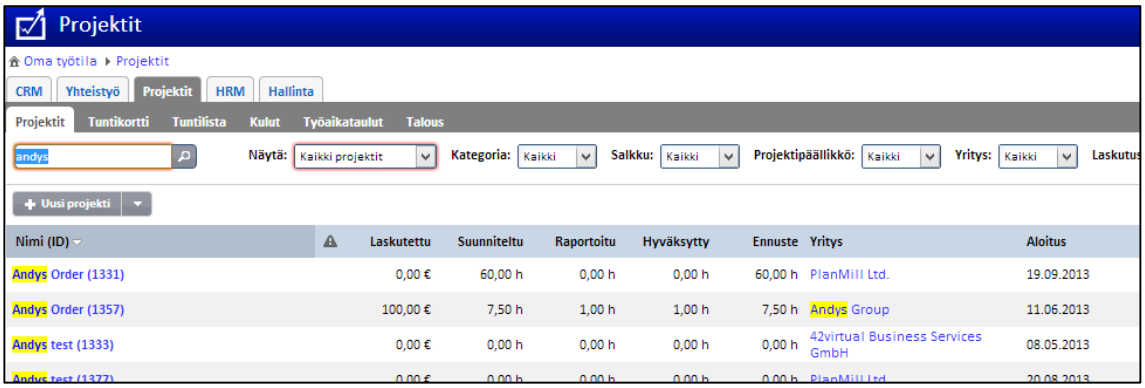
2.1 Käyttöliittymä

PlanMill on verkkosovellus, joka toimii useimmissa tietokoneiden ja mobiililaitteiden verkkoselaimissa. Mobiiliselaimille on oma suppeampi käyttöliittymä. Jokaiselle asiakasyritykselle on oma instanssinsa, johon yrityksen työntekijät kirjautuvat omilla henkilökohtaisilla tunnuksillaan. Instanssi on asiakaskohtainen ilmentymä PlanMill-sovelluksesta, johon liittyy omassa osoitteessa toimiva verkkopohjainen käyttöliittymä sekä tietokanta, joka pitää sisällään kaiken kyseiseen instanssiin liittyvän datan sekä instanssikohtaiset parametrit. Käyttöliittymä rakennetaan pitkälti parametrien perusteella.

Käyttöliittymä koostuu pääasiallisesti muutamasta erilaisesta sivupohjasta, joita ovat lista-, yhteenveto- ja lomakenäkymä sekä raportit. Täten kaikkien moduulien käyttäminen ja räätälöiminen PlanMill-sovelluksessa on yhtenäistä. Moduulilla tarkoitetaan tämän työn yhteydessä tuotemoduulia, joka voidaan aktivoida asiakkaan PlanMill-sovellukseen tuoden lisätoiminnallisuutta. Moduulissa käsitellään entiteettejä. Esimerkiksi Tuotteet-moduulissa tuote on entiteetti, joka sisältää yhteen tuotteeseen liittyvät kaikki tiedot. Tietokannassa entiteetti vastaa yleensä yhtä tietuetta yhdessä tai useammassa yhteen liittyvässä taulussa. Tässä luvussa on esitelty lyhyesti eri sivupohjat ja niille ominaiset piirteet.

2.1.1 Listanäkymä

Listanäkymää käytetään lähinnä silloin, kun halutaan etsiä tietty entiteetti tai tarkastella useamman entiteetin tietoja rinnakkain.



Kuva 1. Listanäkymä PlanMill-sovelluksessa.

Listanäkymä tarjoaa tekstihakukentän sekä suodattimia, joita voidaan käyttää halutun tyyppisten entiteettien listaamiseen (kuva 1). Suodattimet sekä taulukon sarakkeet ovat personoitavissa. Listatun datan vieminen eri formaatteihin on myös mahdollista. Lisäksi listanäkymien yhteydessä ovat usein myös painikkeet entiteettien luomiselle tai poistamiselle.

2.1.2 Yhteenvetosivu

Yhteenvetosivulla näytetään yhden entiteetin kaikki tiedot.

Kuva 2. Yhteenvetosivu PlanMill-sovelluksessa.

Yhteenvetosivulla entiteettiin liittyvät tiedot ovat luokiteltu kategorioittain (esim. ”Laskutustiedot”, kuva 2).

2.1.3 Lomakenäkymä

Lomakenäkymä on entiteetin muokkaustila, jossa muokataan olemassa olevan entiteetin tietoja tai luodaan uusi entiteetti.

Kuva 3. Lomakenäkymä PlanMill-sovelluksessa.

Lomakenäkymä koostuu tekstikentistä, pudotusvalikoista sekä muista käyttöliittymäelementeistä, joilla käyttäjä voi syöttää tietoja entiteetille (kuva 3).

2.1.4 Raporttinäkymä

PlanMill-järjestelmässä on useita raportteja eri moduuleihin liittyvän datan seurantaan. Raportti (Kuva 4) koostuu pitkälti samoista elementeistä kuin listanäkymä (kuva 1). Erona listanäkymään on se, että raportit ovat pitkälti käyttäjäroolikohtaisia ja helpottavat yritystoiminnan hallintaa ja ohjaamista eri alueilla, kuten henkilöstön-, projektien- ja taloudenhallinta.

Kuva 4. Omien projektien tilaraportti PlanMill-sovelluksessa.

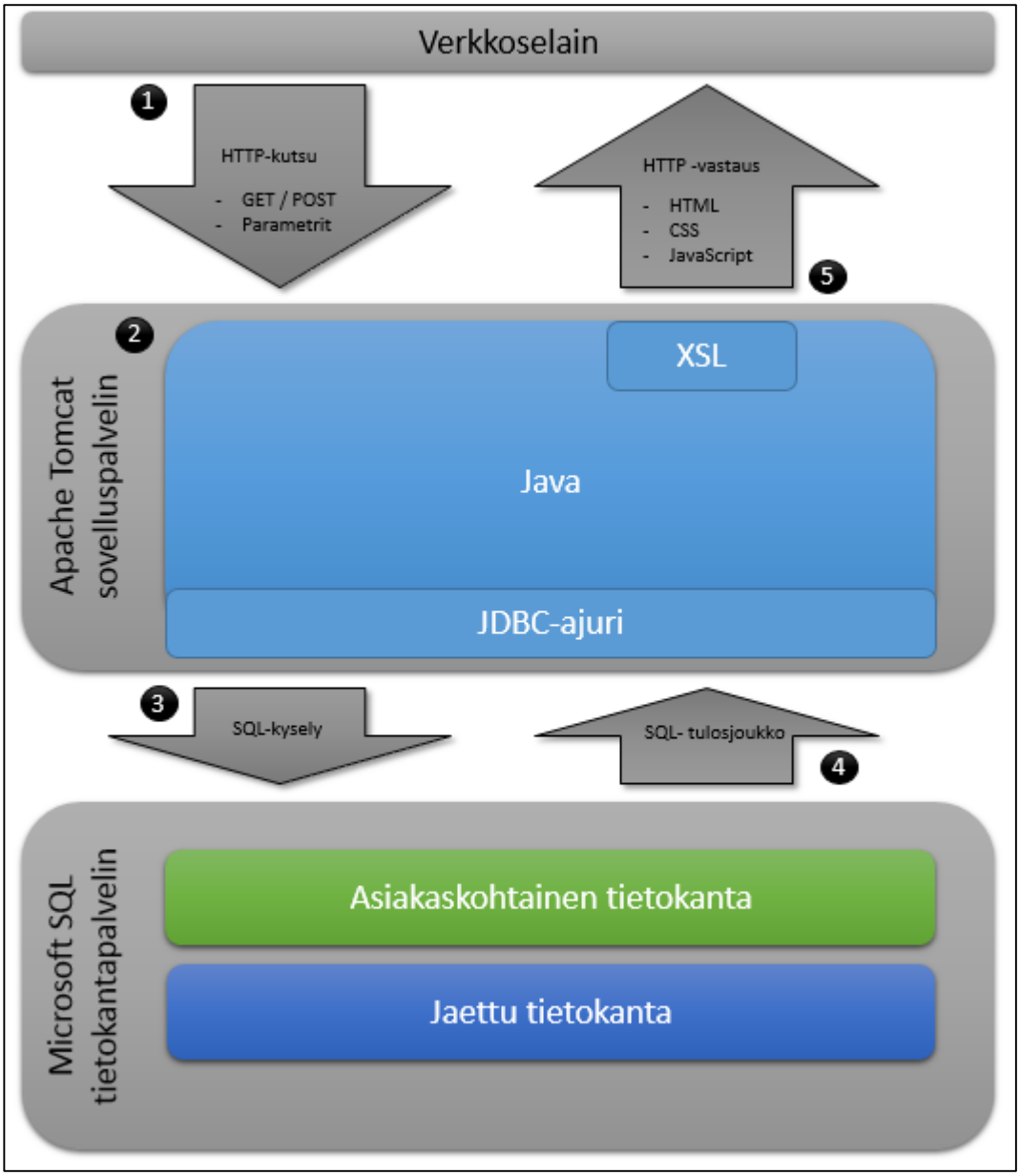
Uusien räätälöityjen raporttien luominen asiakkaan PlanMill-järjestelmään on hyvin yleistä. Raportteja hyödynnetään usein datan viemiseen PlanMill-järjestelmästä ulkoisiin järjestelmiin.

2.2 Järjestelmän arkkitehtuuri

Yksinkertaistettuna PlanMill-järjestelmän arkkitehtuuri koostuu sovellus- ja tietokantapalvelimesta. Sovelluspalvelimena toimii Apache Tomcat ja tietokantapalvelimena Microsoft SQL Server. Tuotannossa on kaksi kunkin tyyppistä palvelinta. Sovelluspalvelimet on synkronoitu keskenään, joten liikenne voidaan ohjata näille kuormituksen mukaan. Asiakkaiden tietokannat on jaettu kahden tietokantapalvelimen kesken, joista toisella palvelimella sijaitsee myös vakioparametrit sisältävä niin sanottu jaettu tietokanta. Lisäksi PlanMill-järjestelmän arkkitehtuuriin kuuluu paljon muutakin liittyen esimerkiksi palvelun saatavuuden varmistamiseen, mutta ne seikat eivät ole tämän työn kannalta olennaisia.

Itse sovellus on toteutettu SaaS-mallille ominaiseen tapaan noudattaen niin sanottua multitenanttista sovellusarkkitehtuuria, jossa sovelluspalvelimella on yksi sovellusinstanssi, joka palvelee useita asiakkaita samanaikaisesti [2, s. 10]. Korkean tason konfiguraation mahdollistavissa multitenanttisissa järjestelmissä on lisäksi tyypillistä, että käyttöliittymä- ja liiketoiminnallista koodia käsitellään kuin sovellusdataa. Käytännössä tämä tarkoittaa sitä, että a) edellä mainitut koodit tallennetaan ja luetaan tietokannasta kuten sovellusdata sekä b) käyttäjälle palautettava dokumentti rakennetaan dynaamisesti

edellä mainittujen asiakaskohtaisten koodien perusteella. Tämä arkkitehtuuri lisää sovelluksen skaalautuvuutta sekä erityisesti saatavuutta, sillä se mahdollistaa ajonaikaisten räätälöintien tekemisen. [3, s. 970-971.]



Kuva 5. PlanMill-järjestelmän arkkitehtuuri yksinkertaistettuna.

Tärkeimpiä käsitteitä PlanMill-järjestelmän sovellusarkkitehtuurista puhuttaessa ovat Java, XHTML, SQL, JavaScript, AJAX sekä parametrit (Luku 2.2.1, s. 10). Sovellusarkkitehtuurisia ratkaisuja voidaan hyvin selittää purkamalla auki kutsun prosessoimiseen liittyvä tapahtumaketju (vertaa Kuva 5):

1) Käyttäjä suorittaa toiminnon (*action*) PlanMill-järjestelmän käyttöliittymässä selaimellaan. Tilanteesta riippuen joko get- tai post-tyyppinen http-kutsu lähetetään sovelluspalvelimelle. Kutsun parametreissa kerrotaan sivun lataamiseen tarvittava informaatio, kuten ladattava moduuli (category), toiminnon tai tarkastelun kohteena olevien entiteettien id:t sekä esimerkiksi suodattimien arvot.

2) Http-kutsu prosessoidaan sovelluspalvelimella. Java-koodissa Category-parametrin arvon perusteella ladataan oikean moduulin koodi suoritukseen. Koodissa moduulit toteuttavat yhteistä rajapintaa, jossa on määritetty metodi toiminnon prosessoimista varten. Kyseiselle metodille annetaan attribuutteina tieto suoritettavasta toiminnosta, sessiotiedot sekä kyseinen http-kutsu. Moduulin koodissa, apuluokkia hyödyntäen,
   - tarkistetaan, onko käyttäjällä tarvittavat oikeudet toiminnon suorittamiseksi ja palautetaan virhesivu tarvittaessa
   - valmistellaan tietokantakyselyt annettujen http-parametrien perusteella
   - suoritetaan tietokantakyselyt
   - rakennetaan palautettava HTML-sivu XSL-pohjia käyttäen.

   PlanMill-parametreja luetaan paljon Java-koodissa, sillä niissä on märitetty muun muassa tietokantakyselyt, käyttöoikeudet ja palvelinten osoitteet. Parametrit luetaan tietokannoista ja tallennetaan sovelluspalvelimen muistiin, jolloin niiden käyttäminen sovellusinstanssin koodissa on tehokasta.

   Javassa luetaan toiminnon suorittamiseen tarvittavien tietokantakyselyjen pohjat parametreista. Pohjat noudattavat yleensä PlanMill-spesifiä merkintäkieltä, joka sisältää paikkamerkkejä (*placeholders*), jotka korvataan Javassa oikeilla arvoilla. Tällaisia paikkamerkkejä on erilaisia;
   - Tietyt hakasulkeilla ympäröidyt termit, kuten ”[userid]”, korvataan sessiotiedoista saatavilla arvoilla.
   - Kysymysmerkit korvataan http-kutsussa lähetettyjen parametrien arvoilla.
   - Tietyt aaltosulkeilla ympäröidyt termit, kuten {columns}, joita käytetään tietokantakyselyjä määrittävien parametrien yhteydessä. Ne korvataan asianomaisista käyttöliittymäparametreista luetuilla arvoilla.

- Erityisempi paikkamerkki mahdollistaa parametrin osan suorittamisen vain, jos käyttäjällä on tarkistettava käyttöoikeus. Syntaksiltaan paikkamerkki on ”[access:’<tarkistettava käyttöoikeus>’:’<ehdonalainen osa>’]” ja Java-koodissa ehdonalainen osa liitetään suoritettavaksi vain jos tarkistettava käyttöoikeus löytyy käyttäjältä.

3) Java-koodista suoritetaan JDBC-ajuria (*Java Database Connectivity Driver*) hyödyntäen tietokantakyselyjä joko tiedon hakemiseksi, lisäämiseksi tai sen päivittämiseksi riippuen toiminnosta. Tietokantapalvelimena käytetty Microsoft SQL Server tukee dynaamista SQL:ää, joka mahdollistaa kyselyjen rakentamisen ajon aikana; tämä on erittäin tärkeä ominaisuus nykyisen PlanMill-sovellusarkkitehtuurin kannalta. Suoritetut kyselyt tallennetaan muistiin joksikin aikaa, jolloin niitä ei tarvitse jokaista pyyntöä varten luoda uudelleen.

4) Tietokantapalvelin palauttaa sovelluspalvelimelle tulosjoukon (*resultset*) eli tehdyn tietokantakyselyn perusteella löytyneet rivit; tai mikäli kysely tuotti virheen, palautuu virhe-ilmoitus. Java-koodissa saatu data muunnetaan XML-muotoon, josta rakennetaan käyttäjälle palautettava HTML-sivu hyödyntäen XSL-tekniikkaa (*eXtensible Sylesheet Language*). XSL-tekniikka mahdollistaa sen, että samasta datasta voidaan tuottaa erilaisia ulosanteja ja tätä hyödyntäen on esimerkiksi toteutettu PlanMill-järjestelmän mobiiliversio.

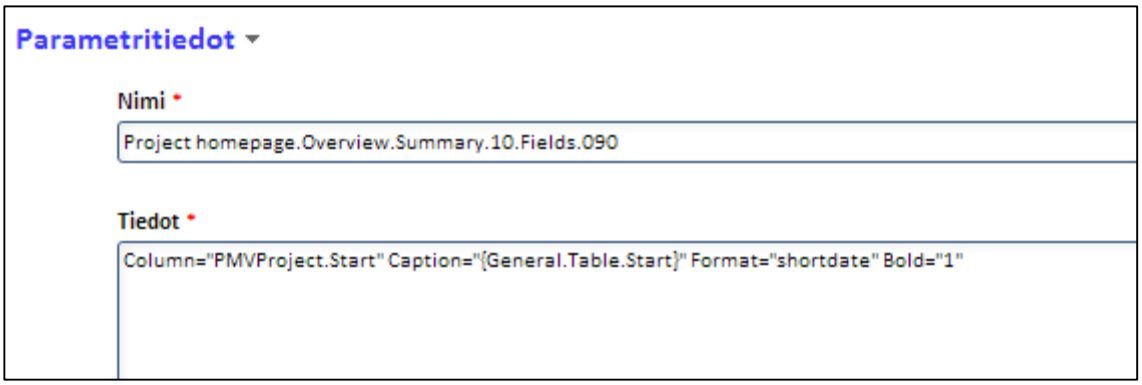
5) Valmisteltu HTML-sivu lähetetään sovelluspalvelimelta asiakkaan selaimeen. PlanMill-järjestelmässä käytetään paljon AJAX-tekniikkaa (*Asynchronous JavaScript And XML*), mikä tarkoittaa käytännössä sitä, että verkkosivulta JavaScript-koodia käyttäen tehdään http-pyyntö palvelimelle asynkronisesti, jolloin verkkosivulle voidaan ladata uutta sisältöä lataamatta koko sivua uudelleen. Esimerkiksi, kun käyttäjä muuttaa suodattimen arvoa listanäkymässä tai raportilla, niin koko sivua ei ladata uusiksi, vaan pelkästään listan sisältö. Sovellusarkkitehtuuriselta kannalta AJAX:illa tehdyn http-pyynnön prosessoimiseen liittyvä tapahtumaketju on pitkälti samanlainen kuin tavallisen http-pyynnönkin. Tyypillisimmät AJAX-kutsut on määritelty PlanMill-järjestelmän JavaScript-kirjastoissa ja uusia AJAX-kutsuja on mahdollista määrittää parametreissa, jolloin niitä voidaan kutsua tarkoitusta varten tehtyä funktiota käyttäen.

2.2.1 Parametrit PlanMill-järjestelmässä

Parametreista puhuttaessa PlanMill-järjestelmän yhteydessä tarkoitetaan Parametri-entiteettejä, joihin voidaan liittää muun muassa

- nimi
- datasisältö, joka noudattaa PlanMill-spesifiä merkintäkieltä
- rooli- ja/tai käyttäjäkohtaisuus
- kuvaus ja erillinen kommentti
- palvelupyynnön tunnistenumero räätälöityjä parametreja varten.

PlanMill-järjestelmässä parametreja käytetään tuettujen ominaisuuksien konfiguroimiseen, kuten sähköpostiasetusten tekemiseen, sekä käyttöliittymä- ja liiketoiminnallisten koodien räätälöimiseen. Oletusarvoisia parametreja PlanMill-järjestelmässä on yli 30 000 kappaletta. Parametreja voidaan käyttää muuttamaan järjestelmän toimintaa lennosta tekemättä muutoksia palvelimella sijaitsevaan koodiin. Räätälöinnit tehdään asiakasinstansseihin pääasiallisesti muuttamalla parametreja asiakasinstanssin käyttöliittymän kautta (Kuva 6).



Kuva 6. Parametrin muokkaustila, jossa muokattavana projektin yhteenvetosivulla näytettävä projektin alkamispäivämääräkenttä.

PlanMill-järjestelmän parametreja varten on kehitetty oma merkintäkieli, joka on yhdistelmä HTML-, SQL- sekä JavaScript-kieliä. Parametrin datasisältö riippuu paljon siitä, mitä parametrilla määritetään. Parametrin data on useissa tapauksissa pelkkää SQL- tai JavaScript-kieltä, mutta käyttöliittymäelementtejä määrittävät parametrit ovat erityisiä ja koostuvat attribuuteista, joiden arvona voi olla mitä tahansa edellä mainituista (Kuva 7).

034	19.10.2013 21:24	Column="description" Width="1%" Caption="_iconmap(ui-icon-comment) " Format="iconmap" Visible="1" ColumnSQL="CASE WHEN Request.HasDescription = 1 THEN N'ui-icon-comment' ELSE NULL END" ToolTip="{General.Table.Description}" OnMouseOver="PLANMILL.ui.overlay.viewListOverlay('[instance]',this,'Load description for request','Request.Id')" OnMouseOut="PLANMILL.ui.overlay.closeOverlay(false)"

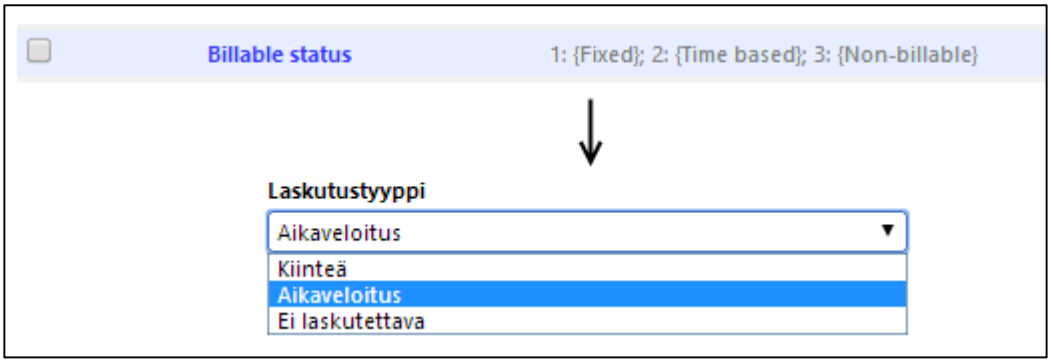
Kuva 7. Eräs käyttöliittymäelementin määrittävä parametri, jossa on paljon attribuutteja.

Attribuutit vastaavat syntaksiltaan pitkälti HTML-kielen attribuutteja, ja ne tarjoavat erittäin ilmaisuvoimaiset työkalut käyttöliittymäelementtien muokkaamiseen, mutta niiden hallitseminen on vastaavasti haastavaa ja vaatii paljon perehtymistä.

PlanMill-järjestelmän arkkitehtuuri rakentuu siten, että oletusarvoiset parametrit luetaan kaikille asiakkaille yhteisestä jaetusta tietokannasta. Jokaista asiakasta varten on lisäksi oma tietokanta, joka sisältää asiakaskohtaisen datan sekä räätälöidyt parametrit. Sovelluspalvelimella käytetään ensisijaisesti asiakkaan tietokannasta löytyviä parametreja ja toissijaisesti jaetussa tietokannassa sijaitsevia parametreja. Käyttöliittymän kautta pääsee tarkastelemaan ja muokkaamaan kyseessä olevan instanssin tietokannassa olevia parametreja. Jaettu tietokantakin on oma instanssinsa, joten kaikkia instansseja koskeviin oletusarvoisiin parametreihin pääsee käsiksi sen käyttöliittymän kautta.

2.2.2 Kieliasetusten ja enumeraatioiden määrittäminen

PlanMill-järjestelmässä kieliasetuksia ja enumeraatioita käsitellään pitkälti samalla tavalla kuin parametreja. Itseasiassa enumeraatiot ovat PlanMill-järjestelmässä Parametri-entiteettejä, joilla on erityistarkoitus. Enumeraatioita käytetään muuttamaan käyttöliittymässä näkyvät kenttien arvot tietokantaan tallennettavaan arvoon ja päinvastoin.



Kuva 8. Esimerkki enumeraatiosta, jossa arvot on määritetty viittauksilla Kieli-entiteetteihin.

Enumeraatio-parametrien data-arvot koostuvat listasta puolipisteellä eroteltuja avain-arvo pareja, joissa ensimmäinen on tietokannasta luettu kentän arvo ja jälkimmäinen käyttöliittymässä näkyvä termi, joka on useimmiten viittaus Kieli-entiteetin nimeen (Kuva 8).

Kieli-entiteettejä lisäämällä tai muokkaamalla voidaan luoda uusia kieliasetuksia järjestelmään tai muuttaa oletuksena olevia termejä vastaamaan paremmin asiakkaan tarpeita. Kieli-entiteetti sisältää nimen, datan sekä tiedon siitä, minkä kielinen termi on. PlanMill-merkintäkieli mahdollistaa viittaamisen Kieli-entiteetteihin käyttämällä aaltosulkeita (kuva 8), jolloin järjestelmä korvaa aaltosulkeet ja niiden väliin kirjoitetun Kieli-entiteetin nimen kieliasetusten mukaisella termillä.

2.2.3 Käyttöoikeuksien hallinta

PlanMill-järjestelmässä jokainen käyttäjä kuuluu johonkin rooliin. Rooleilla määritellään käyttäjäryhmäkohtaiset käyttöoikeudet. Yleisiä rooleja ovat esimerkiksi esimies, taloushenkilö ja tiiminvetäjä. Rooleille voidaan määritellä käyttöoikeudet asiakkaan toiveiden mukaisesti heille aktivoitujen tuotteiden puitteissa. Jokainen tuote muodostaa oman käyttöoikeusryhmän, joka sisältää joukon käyttöoikeuksia tiettyihin moduuleihin. Järjestelmän ylläpitäjärooli on luonnollisesti kaikista korkeimman tason rooli, jolla on pääsy erityistoimintoihin, kuten parametrien hallintaan.

Eri moduuleihin ja niiden toimintoihin vaadittavat käyttöoikeudet on määritelty parametritasolla. Järjestelmässä käyttöoikeuksia määritellään kohteesta riippuen joko 1) käyttöoikeusparametreilla, joita käytetään moduulien käyttöoikeustarkistuksissa, 2) parametrikohtaisilla käyttöoikeusasetuksilla, joita hyödynnetään usein räätälöintien yhteydessä tai 3) parametrin attribuuttitasoisilla käyttöoikeustarkistuksilla, joilla voidaan tehdä edistyneempiä, dynaamisiakin käyttöoikeustarkistuksia, kuten tässä työssä käytetyissä ratkaisuissa.

3 Parametrien konfigurointiin liittyvien haasteiden kartoitus

3.1 Räätälöinnit

Asiakasyritysten vaatimukset toiminnanohjaussovellukselle ovat lähes aina uniikit. Vaatimuksiin vaikuttavat muun muassa

- asiakasyrityksen toimiala
- asiakasyrityksen tuotteiden ja palveluiden tyyppi
- kulttuuriset erot
- erilaiset säännökset
- asiakasyrityksen toimintastrategia. [6, s. 18.]

Erilaisten vaatimusten takia räätälöintien tekeminen asiakkaiden instansseihin muodostaa suuren osan PlanMill Oy:n henkilökunnan arjesta. Räätälöinnit PlanMill-sovelluksessa ovat pitkälti tehtävissä muuttamalla oikeita parametreja käyttöliittymän kautta, mutta oikeiden parametrien löytäminen tai parametrien muokkaaminen vie toisinaan aikaa. Asiakasmäärän kasvaessa tällä on vaikutusta toimitusaikoihin ja palvelun tasoon. Se myös vaikeuttaa uusien työntekijöiden kouluttamista ja hidastaa vanhojen työntekijöiden työskentelyä.

3.1.1 Yleisimmät parametrien muokkaamiseen liittyvät toimenpiteet

Osana opinnäytetyötä tehtiin pieni selvitystyö asiakkaiden vuoden 2013 alkupuolella tilaamista toimenpiteistä. Selvitys toteutettiin yhdessä toisen Metropolian opiskelijan kanssa, joka teki opinnäytetyötä samaan aikaan PlanMill-sovelluksen kieliasetusten hallintatyökaluista. Tavoitteena oli selvittää yleisimmät parametrien ja kieliasetusten muokkaamiseen liittyvät tilatut toimenpiteet ja se, mihin osaan sovellusta nämä toimenpiteet kohdistuivat, jotta voitaisiin suunnitella parannuksia oikeisiin kohteisiin.

Selvityksen data muodostui vuoden 2013 alkupuoliskolla yrityksessä vastaanotetuista palvelupyynnöistä, jotka olivat tyypiltään tilattuja toimenpiteitä. Tilattuja toimenpiteitä kyseiselle aikajaksolle kertyi yhteensä 310. Tilatut toimenpiteet käytiin yksitellen läpi ja pyrittiin ryhmittelemään mahdollisimman hyvin. Tarkemmin luokiteltiin parametrien ja kieli-

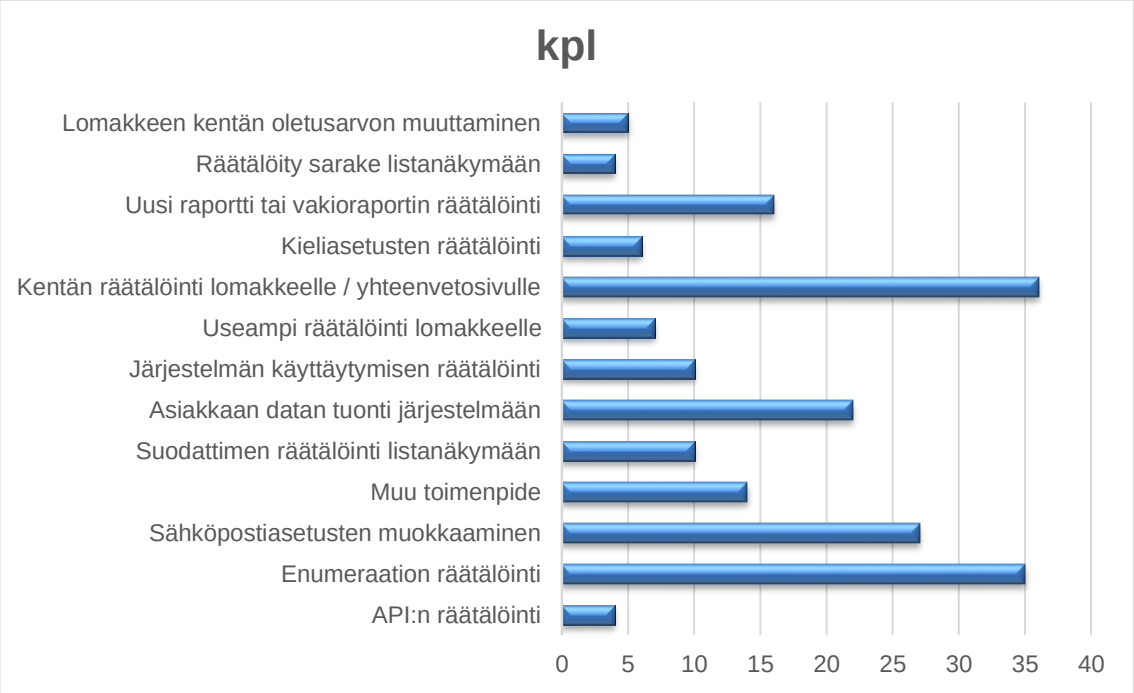
asetusten muokkaamiseen liittyvät toimenpiteet. Toimenpiteet jaettiin kategorioihin niiden luonteen perusteella. Lisäksi ne pyrittiin ryhmittelemään kategorian sisällä niiden tehtävätyypin mukaan.

Taulukko 1. Tilattujen toimenpiteiden lopullinen ryhmittely kategorioihin.

Kategoria	Selite	Lkm
Tapaamiset	Workshopit, asiakaspalaverit	11
Tietokantaoperaatiot	Tietokannan kautta tehtävät operaatiot, kuten massapäivitykset	18
Asennus	Sovelluksen konfiguroiminen käyttöliittymän kautta	35
Pystytys	Uuden instanssin asentaminen, integraation käyttöönotto, uuden tuotemoduulin käyttöönotto	19
Parametrisointi	Parametrien, kieliasetusten tai enumeraatioiden muokkaaminen, uudet raportit	196
Muut	Muut kuin ylläoleviin ryhmiin sopivat, esim. PlanMill wikiin liittyvät toimenpiteet	31
Yhteensä		**310**

Kuten taulukossa 1 on nähtävissä, muodostaa parametrisoinniksi luokiteltujen parametrien ja kieliasetusten muokkaamiseen liittyvät toimenpiteet selkeästi suurimman osan (196 kpl) tarkastelujakson aikana tehdyistä tilatuista toimenpiteistä.

Taulukko 2. Parametrien ja kieliasetusten muokkaamiseen liittyvät toimenpiteet.



Selkeästi yleisimpiä parametrien ja kieliasetusten muokkaamiseen liittyviä tilattuja toimenpiteitä 2013 alkupuoliskolla olivat lomakenäkymässä olevan syöttökentän räätälöinti sekä enumeraation räätälöinti, kuten taulukosta 2 näkyy. Edellä mainitut asiat liittyvät PlanMill-järjestelmän personointiin, eli asiakkaat joko tarvitsevat uusia kenttiä entiteetin kuvaamiseen tai haluavat muuttaa käyttöliittymässä käytettävää terminologiaa vastaamaan paremmin omaa toimialaa. Lisäksi tästä joukosta selkeästi enemmän tilattuja toimenpiteitä olivat sähköpostiasetusten muokkaaminen, asiakkaan datan tuonti järjestelmään sekä uuden raportin tekeminen tai olemassa olevan muokkaaminen. Nämä tulokset selittyvät sillä, että kyseisenä ajanjaksona PlanMill Oy sai uusia asiakkaita, jotka teettävät käyttöönottoon liittyen paljon juuri edellä mainitun tyyppisiä toimenpiteitä.

3.1.2 Tyypillisen räätälöintitoimenpiteen vaiheet työn alkutilanteessa

Tässä luvussa kuvataan erään tyypillisen räätälöintitoimenpiteen vaiheet työn alkuvaiheessa. Saman toimenpiteen vaiheet käydään läpi luvussa 4.6.1 projektissa tehtyjen muutosten jälkeen, jotta voidaan arvioida projektin tulosten vaikutuksia käytettävyyteen. Esimerkkinä käytetään suhteellisen yksinkertaista räätälöintitoimenpidettä, jossa lomakkeelle lisätään uusi tekstikenttä. Nämä vaiheet perustuvat erään uuden työntekijän tätä opinnäytetyötäkin silmällä pitäen tekemiin ohjeisiin, joiden yhteydessä hän myös kertoi omia ajatuksiaan prosessin nykytilan ongelmallisuudesta; näistä kerrotaan enemmän seuraavassa luvussa. Seuraavat välivaiheet ovat melko tiivistetyt ja saattavat sisältää useamman askeleen per välivaihe. Niiden tarkoitus on luoda lukijalle suuntaa-antava käsitys prosessin monimutkaisuudesta.

1) Kirjaudu asiakkaan instanssiin ja mene lomakkeelle, johon räätälöinti tulee tehdä. Laita muistiin selaimen osoiterivillä olevan ”category” GET-parametrin arvo, sillä se näyttää polun, jonka alta kyseisen lomakkeen parametrit löytyvät hallintapaneelissa (kuva 9). Kokeneempi työntekijä ohittaisi tämän vaiheen. Tärkeää on myös painaa mieleen lisättävää kenttää edeltävän ja seuraavan kentän nimet.

Kuva 9. GET-parametrin kopiointi.

2) Siirry hallintapaneeliin ja siellä Parametrit-välilehdelle. Tarkista, onko polku jo olemassa ja löytyykö sen alta entuudestaan räätälöityjä parametreja lisättävää kenttää edeltävälle tai seuraavalle kentälle.
   a. Mikäli parametreja ei asiakkaan instanssista löydy, kirjaudu jaettujen parametrien instanssiin ja mene siellä samaan paikkaan eli Parametrit-välilehdelle hallintapaneelin alla. Avaa polku parametripuusta (kuva 10).

Kuva 10. Parametrin polun auki avaaminen.

3) Katso, mikä on lisättävää kenttää edeltävän kentän parametrin nimi (numero) ja sitä seuraavan kentän parametrin nimi, sillä uusi kenttä halutaan lisätä näiden kahden väliin. Parametrien nimet koostuvat polusta, jossa ne sijaitsevat ja viimeinen pisteellä erotettu osa on parametrin muista samassa polussa olevista identifioiva osa, useimmiten numero, joiden suuruusjärjestys myös määrää kenttien esiintymisjärjestyksen lomakkeella. Parametreja saattaa olla lukuisia, joten kannattaa käyttää selaimen tekstihakutoimintoa (kuva 11).

Kuva 11. Oikean parametrin paikantaminen.

4) Avaa parametri muokkaustilaan klikkaamalla sen nimeä ja ota koko polku talteen nimikentästä (kuva 12).

Kuva 12. Parametrin nimen kopioiminen.

5) Siirry asiakkaan instanssiin ja Parametrit-välilehdellä klikkaa Uusi parametri –painiketta.
6) Liitä talteen ottamasi parametrin koko polku nimikenttään ja muuta viimeinen pisteellä erotettu osa siten, että uusi nimi sijoittuu numerojärjestyksessä edeltävän ja seuraavan kentän nimien väliin. Täytä sitten datakenttään uutta räätälöityä tekstikenttää varten tarvittavat tiedot; uudelle työntekijälle tämä tarkoittaa käytännössä toisen samankaltaisen räätälöidyn kentän parametrista mallia ottamista tai omien muistiinpanojen käyttämistä. Tallenna lopuksi parametri.
7) Siirry sitten lomakkeelle tarkistamaan, että lisätty kenttä näkyy ja toimii oikein. Yleensä tämän jälkeen sama operaatio tehdään kentän lisäämiseksi vähintään myös yhteenvetosivulle.

3.2 Käyttäjien kokemat ongelmat

Erityisen vaikeaksi parametrien kanssa työskentelevät uudet työntekijät kokivat oikeiden parametrien löytämisen. Kuten edellä mainittiin, osoiteriviltä löytyvän *category*-parametrin arvo antaa suunnan, mutta oikean kentän parametri saattaa olla sen alla vielä useamman alakategorian takana, joiden nimet eivät ole usein informatiivisia. Niminä on päädytty käyttämään monissa paikoissa pelkkiä numeroita, koska ne ovat helposti laitettavissa suuruusjärjestykseen; järjestelmä lisää elementit käyttöliittymään käyttäen alfanumeerista suuruusjärjestystä. Tämä nimeäminen kuitenkin aiheuttaa päänvaivaa sekä vanhoille, että uusille työntekijöille ja johtaa usein siihen, että käyttäjä itse päättelee kohdan suurin piirtein ja sitten klikkailee kategorioita auki, kunnes löytää oikean.

Toinen erityisen vaikeaksi koettu asia uusien työntekijöiden keskuudessa oli parametrien muokkaus. Käyttöliittymäparametrin data-arvo voi sisältää lukuisen määrän erilaisia attribuutteja, joista jotkut ovat ominaisia vain tietyn näkymän parametreille sekä osalla on eri merkitys eri näkymien yhteydessä. Esimerkiksi *Column*-attribuutilla voidaan listanäkymän yhteydessä viitata toisen sarakkeen arvoon listassa, kun taas lomakenäkymän yhteydessä kyseisen attribuutin arvo tarkoittaa tietokantataulun saraketta, johon kentän arvo yhdistetään. Ongelmia tuottavat lisäksi parametrit, jotka ovat herkkiä niiden arvojen syntaksin suhteen tai joiden arvoissa käytetään erityissyntaksia. Esimerkiksi, joidenkin parametrien syntaksissa rivinvaihto on sallittu, kun taas toisissa se aiheuttaa ongelmia. Uuden työntekijän täytyy tukeutua paljon PlanMill-wikiin, mutta sekään ei läheskään aina ole riittävän kattava, jolloin uusi työntekijä päätyy konsultoimaan kokeneempia työntekijöitä keskeyttäen heidän työnteon.

Lisäksi, edellisessä luvussa mainitut räätälöintiprosessin vaiheet eivät ole pelkästään turhan monimutkaiset, mutta ne myös mahdollistavat vakavien virheiden tekemisen. Usein instanssit menevät käyttäjällä sekaisin, kun niitä on auki selaimessa monella välilehdellä. Epähuomiossa työntekijä saattaa muokata parametria tai jopa poistaa sen vahingossa jaettujen parametrien instanssista, jolloin muutos vaikuttaa kaikkiin asiakasinstansseihin aiheuttaen pahimmillaan käyttökatkoksia käyttäjille. Peruutustoimintoa ei ole, jolloin alkuperäinen arvo täytyy tarpeen tullen palauttaa tietokannan kautta hakemalla vanha arvo historiataulusta.

3.3 Aikaisemmin esitetyt parannusehdotukset

Yksi syy tämän opinnäytetyön alkuun saattamiselle oli lista palvelupyyntöjä, jotka koskivat henkilökunnan ehdottamia parannuksia parametrien käsittelyyn liittyen. Opinnäytetyön toteutusosuutta suunniteltaessa pyrittiin nämä ehdotukset ottamaan mahdollisemman hyvin huomioon ja niistä keskusteltiin muutamissa henkilökunnan kesken käydyissä eri palavereissa. Tässä luvussa käydään läpi näihin ehdotuksiin liittyviä ajatuksia.

3.3.1 Jaettujen parametrien näkyminen asiakasinstansseissa

Ehdotus jaettujen parametrien näkymisestä asiakasinstansseissa sai paljon kannatusta sekä uusilta, että vanhoilta työntekijöiltä. Tässä ehdotuksessa jaetut parametrit näytettäisiin asiakasinstanssin Parametrit-välilehdellä harmaalla värillä, jotta käyttäjälle olisi

selvää, mitkä parametrit ovat paikallisia ja mitkä jaettuja. Lisäksi käyttäjän olisi mahdollista valita suodattimesta näytettäväksi pelkästään paikalliset tai jaetut parametrit ja uusi painike mahdollistaisi jaetun parametrin kopioimisen paikalliseksi. Jaettuihin parametreihin ei pystyisi tekemään muutoksia asiakasinstanssin kautta. Ilmeinen hyöty näistä muutoksista olisi se, ettei enää tarvitsisi mennä jaettujen parametrien instanssiin, mikä yksinkertaistaisi ja nopeuttaisi prosessia sekä estäisi vahinkojen tapahtumisen jaettuihin parametreihin.

3.3.2 Hakutyökalu parametreille

Ehdotuksessa parametrien hakutyökalusta ei tarkemmin määritelty, minkälainen hakutyökalun tulisi olla, mutta toiveena oli, että oikean parametrin löytäminen olisi helpompaa. Ehdotukseen sisältyi myös ajatus siitä, että koko tapa selata parametreja voitaisiin suunnitella uusiksi, mahdollisesti korvaten nykyinen Parametrit-välilehdellä oleva puurakenne ja kategoriasuodatin jollain tehokkaammalla ratkaisulla.

3.3.3 Staattisen analyysin työkalut

Staattisen analyysin työkaluja koskevassa ehdotuksessa listattiin lukuisia kohteita, joihin staattista analyysia voitaisiin käyttää. Analyysien pohjalta olisi mahdollista tunnistaa puutteita järjestelmässä ja siistiä vanhoja jäänteitä pois. Staattisen analyysin työkaluilla voitaisiin tarkistaa lähinnä asioita, kuten

- puuttuvia käyttöoikeuksia, esimerkiksi raporteista
- parametreista puuttuvia olennaisia attribuutteja tai niiden puutteellisia arvoja
- parametrissa viitatun enumeraation tai kielen puuttumista
- käyttäjän personoimien sarakkeiden ja suodattimien vastaavuutta olemassa oleviin.

3.3.4 Käyttöliittymäkenttien konfigurointi muokkauslomakkeella

Käyttöliittymäparametrien muokkaamisen haasteellisuus voitaisiin sivuuttaa kehittämällä muokkauslomake, joka tarjoaisi selkeämmän ja yksinkertaisemman rajapinnan parametrien muokkaamiselle. Tämä ehdotus liittyi pariin toiseen ehdotukseen, joista toisessa haluttiin ylläpitäjänä mahdollisuus päästä muokkaamaan kentän parametria helposti esimerkiksi käyttöliittymässä kenttää klikkaamalla ja toisessa haluttiin valmiita pohjia, joita

voitaisiin käyttää uusien räätälöityjen kenttien lisäämisessä. Yhdessä nämä kolme ehdotusta muodostivat potentiaalisen kokonaisuuden, joka sai työntekijöiden keskuudessa kannatusta

3.4 Kehityskohteet

Edellä esiteltyjä parannusehdotuksia sekä selvitystyön tuloksia käsiteltiin useissa eri palavereissa henkilökunnan sekä projektiin liitettyjen henkilöiden kesken; yksi näistä henkilöistä oli toinen samaan aikaan opinnäytetyötä tekevä opiskelija. Keskusteluissa tuli esille vielä jonkin verran uusia aspekteja, joihin toivottiin kiinnitettävän huomiota toteutusta suunniteltaessa. Eräs tällainen seikka oli asiakkaiden itse tekemien yksinkertaisten muokkausten mahdollistaminen, kuten enumeraatio-parametrien ja Kieli-entiteettien muokkaaminen tai uusien syöttökenttien lisääminen lomakkeille. Lisäksi ajatuksena oli, että toisen opinnäytetyön tuloksia voitaisiin hyödyntää tässä työssä siten, että parametreista viitattujen Kieli-entiteettien ja enumeraatio-parametrien löytäminen ja muokkaaminen olisi myös helpompaa.

Keskustelujen pohjalta päätettiin kohdistaa toteutettavat parannukset käyttöliittymäelementtejä määrittävien parametrien käsittelyyn ja ensisijaisesti lomakenäkymään. Tämä päätös oli luontevaa tehdä, kun otetaan huomioon, että 1) asiakkaiden eniten tilaamat toimenpiteet kohdistuivat käyttöliittymäelementtejä määrittäviin parametreihin ja erityisesti lomakenäkymään sekä 2) käyttöliittymäelementtejä määrittävät parametrit ovat haasteellisimpia käsitellä. Kerätyn aineiston pohjalta tehtiin yhteenveto parametrien muokkaamiseen liittyvistä eri aspekteista, tai prosessin vaiheista, joihin koettiin tarvittavan parannuksia.

Taulukko 3. Aineiston pohjalta kerätyt kehityskohteet.

Kehityskohde	Selite	Ehdotetut ratkaisut
Parametrien löytäminen	parametrien löytämistä edistävät työkalut	parametrihaku, Parametrit-välilehden uudelleen suunnittelu
Parametrien muokkaaminen	parametrien muokkaamista helpottavat työkalut	edistyksellisempi parametrin muokkauslomake
Parametrien lisääminen	uusien parametrien luomista helpottavat työkalut	valmiit pohjat käyttöliittymä-parametrien luomiseen
Turvallisuus / virheiden välttäminen	seikat, jotka vähentävät virheiden syntymistä ja edistävät turvallisuutta	jaettujen parametrien näkyminen paikallisesti
Staattinen analyysi	työkalut, jotka edistävät puutteiden löytämistä parametreissa	yksi työkalu, jolla voidaan etsiä ja listata eri tyyppisiä puutteita kaikista parametreista
Asiakkaiden itse tekemät muutokset	työkalut, jotka ovat niin helppoja ja turvallisia, että voidaan antaa asiakkaiden itse käyttää	lomakkeen muokkausnäkymä
Kieli-entiteettien ja enumeraatio-parametrien löytäminen / muokkaaminen	työkalut, jotka helpottavat parametreissa viitattujen Kieli-entiteettien ja enumeraatio-parametrien tarkastelemista / muokkaamista	toisessa opinnäytetyössä toteutettavien työkalujen hyödyntäminen

Taulukon 3 kehityskohteiden sekä ehdotettujen ratkaisujen valossa lähdettiin suunnittelemaan prototyyppejä.

4 Parannusten toteuttaminen

4.1 Projektissa käytetyt menetelmät

Projektissa käytettiin BDD (*Behaviour-driven Development*) -kehitysprosessin periaatteita sekä Scrum-projektinhallinnan menetelmiä. BDD:tä käytettiin projektissa osittain kokeilumielessä, sillä PlanMill Oy:ssä on suunnitelmia hyödyntää BDD:tä tulevaisuudessa testauksen automatisointiin. Lisäksi kyseisen kehitysprosessin hyötyjä haluttiin mitata käytännössä. Scrum-projektinhallintaa haluttiin käyttää, koska projekti oli luonteeltaan sellainen, että alussa ei voitu kunnolla tietää, ovatko asetetut tavoitteet realistisia ja Scrum mahdollisti nopean muutoksiin reagoimisen. Tässä luvussa kerrotaan hieman näistä menetelmistä ja miten niitä projektissa sovellettiin.

4.1.1 BDD (*Behaviour-driven Development*)

BDD lainaa paljon testivetoisen kehityksen (Test-driven Development) periaatteita. Testivetoinen kehitys on sovelluskehityksen metodiikka, joka sanoo, että jokaista sovelluksen osaa varten kehittäjän täytyy määrittää ensin testisarja, sitten tehdä toteutus ja lopuksi todentaa, että toteutus läpäisee testit. BDD tuo lisätarkennusta tähän määrittelyyn sanoen, että testit tulee tehdä sillä tasolla, millä työn tilaaja on määrittelyn antanut. BDD myös sanelee raamit, joita testien tulee noudattaa. Testit kerrotaan käyttäen käyttäjätarinoita. Käyttäjätarina koostuu

- yksiselitteisestä otsikosta
- lyhyestä kuvauksesta, joka kertoo kuka tekee, mitä tekee ja miksi
- hyväksymiskriteereistä, jotka koostuvat erilaisista skenaarioista, joihin liittyy alkuehdot, tehtävä toimenpide sekä oletettu lopputulos.

BDD ei sanele tarkasti miten testit kirjoitetaan, mutta olettaa, että kaikissa testeissä käytetään yhtenäistä formaattia.

```
1. Searching parameters

USER STORY: I want to search parameters by using a text search.

RULE1: User must be on the Parameters tab


GIVEN
        <Search string> is typed into the text search field
WHEN
        User clicks the search button
THEN
        Show list of parameters

                WHERE
                        Parameter.Name CONTAINS <Search string>
                        OR Parameter.Data CONTAINS <Search string>
```

Kuva 13. Projektissa käytetty BDD-formaatti.

Projektissa käytetty formaatti (kuva 13) on hieman karsittu versio, jossa on jätetty pois käyttäjän roolin esittely, koska kaikissa tarinoissa käyttäjänä on ylläpitohenkilö. Syntaksissa on käytetty SQL-kielestä tuttuja termejä, kuten ”WHERE” ja ”CONTAINS”, koska ne ovat henkilökunnalle tuttuja ja helposti miellettävissä olevia termejä. Lisäksi muuttujia kuvaamaan päätettiin käyttää aaltosulkeita.

Scenario	Description	Search string	Expected minimum results	Tes-ter	Test result
1	Search a string that is contained in parameters name	address-toserver	System.Mail.Address-ToServer	JH	OK
2	Search a string that is contained in parameters data	generic/in-clude	System.Folders.CSS	JH	OK

Kuva 14. Esimerkki siitä, miten testiskenaariot kuvattiin tauluihin.

Käyttäjätarinakohtaiset testiskenaariot listattiin taulukoihin, joissa oli rivi jokaista testiskenaariota kohden (Kuva 14). Sarakkeiden määrä myös vaihteli käyttäjätarinasta riippuen. Toteutuksen lopussa hyväksymistestit tehtiin näiden taulukoiden pohjalta ja tulokset raportoitiin suoraan niille varattuihin sarakkeisiin.

4.1.2 Scrum ja projektitiimi

Scrum on ketterän ohjelmistotuotannon viitekehys, jolle on ominaista iteratiivisuus projektin hallinnassa sekä kasautuva sovelluskehitys. Scrumin keskiössä on tiivis itsestään ohjautuva tiimi, jossa panostetaan kommunikaatioon tiimin jäsenten kesken pyrkien siten saamaan yhteisymmärrys projektin suunnasta ja tavoitteista. Pääperiaate Scrumissa on tiedostaa, että projektin aikana muutoksia matkaan todennäköisesti tulee joko tiimin tai tuoteomistajan eli asiakkaan osalta. Tähän varaudutaan projektinhallinnallisilla menetelmillä, jotka perustuvat nopeaan muutoksiin reagoimiseen. Näitä ovat koko projektin pilkkominen tehtävätasolle ja niiden jakaminen lyhyisiin työjaksoihin eli sprintteihin sekä päivittäinen tiimin jäsenten seuranta. Lisäksi tärkeänä pidetään tiiviin kommunikaation ylläpitämistä tuoteomistajan kanssa [8].

Tässä projektissa hyödynnettiin Scrumin periaatteita, sillä ne sopivat projektin luonteeseen hyvin. Projektin alussa opinnäytetyön tekijä ei tuntenut kehitettävää järjestelmää sen syvällisemmin ja siten oli vaikeaa arvioida, kuinka realistisia projektille asetetut tavoitteet olivat. Scrum mahdollisti tavoitteiden muuttamisen lennosta sitä mukaan, kun projekti eteni ja asiat selkiintyivät. Projektia varten valittiin henkilökunnasta yksi PlanMill-järjestelmän hyvin tunteva henkilö toimimaan projektissa niin sanotussa scrum masterin roolissa, jonka tehtävä on vastata siitä, että Scrumin periaatteita noudatetaan projektissa. Sama henkilö toimi projektissa myös tuoteomistajana, jonka tehtävä on edustaa

asiakasta projektissa ja jolla on ensisijainen päätösvalta projektin tavoitteiden määrittelyssä. Päiväpalaverien pitäminen scrum masterin kanssa koettiinkin projektin kannalta erittäin tärkeäksi. Näissä palavereissa käytiin Scrumin periaatteiden mukaisesti läpi edellisen päivän tekemiset sekä suunnitelmat sille päivälle ja kerrottiin mahdollisista ongelmista, jotka estivät työn etenemistä. Projektin kehitystiimiin kuului opinnäytetyön tekijän lisäksi yksi yrityksen seniorikehittäjistä toimimaan teknisen asiantuntijan roolissa sekä toisen opinnäytetyön tekijä, jonka voidaan ajateltavan osaksi samaa projektia, sillä töiden aiheet olivat niin lähellä toisiaan, että päivittäinen kommunikaatio oli usein välttämätöntä töiden tekijöiden kesken.

Projekti suunniteltiin alun perin jaettavaksi määrätyn kestoisiin sprintteihin, mutta koska toteutusosuudelle varattu aika oli kokonaisuudessaan vain parista kolmeen viikkoa, päädyttiin sprint-ajattelua soveltamaan hieman. Käytännössä projekti eteni siten, että tuotteen kehitysjonossa (*Sprint backlog*) olevia asioita toteutettiin yksi kerrallaan ja jokaisen valmistuttua arvioitiin jäljellä olevan ajan puitteissa, mitä seuraavaksi tulisi kehittää, jotta saataisiin projektin päättymisajankohtaan mennessä eheä kokonaisuus aikaiseksi. Tuotteen kehitysjonoon projektin päättyessä jääneet asiat jäivät odottamaan toteuttamista tulevaisuudessa tehtävissä ”sprinteissä”.

4.1.3 Kehitysympäristö

Pääasiallisina kehitystyökaluina projektissa käytettiin Eclipse-ohjelmointiympäristöä ja Microsoft SQL Management Studio -tietokantojen hallintasovellusta. Kaikki Java-ohjelmointi tapahtui Eclipse-ohjelmointiympäristössä ja siinä projektin hallintaan käytettiin Maven-työkalua, joka voidaan integroida Eclipse-kehittimeen. Maven on työkalu, jolla voidaan automatisoida projektiin liittyvien koodikirjastojen hallinta ja projektin rakentaminen. Microsoft SQL Management Studiota käytettiin paljon tietokantakyselyjen ja -proseduurien kehittämisessä sekä testauksessa.

Versionhallinnassa käytettiin Subversion-versionhallintajärjestelmää ja TortoiseSVN-asiakassovellusta. Versionhallinta on olennainen asia ohjelmistotuotannossa; sitä käytetään ohjelmistoprojektin tuotosten hallintaan, ohjelmiston kehityksen seurantaan sekä hallittuun kehitykseen [10]. Koska samaan aikaan tehtiin kahta opinnäytetyötä, joissa tehdyt muutokset todennäköisesti vaikuttaisivat toisiinsa, versionhallinta toteutettiin kah-

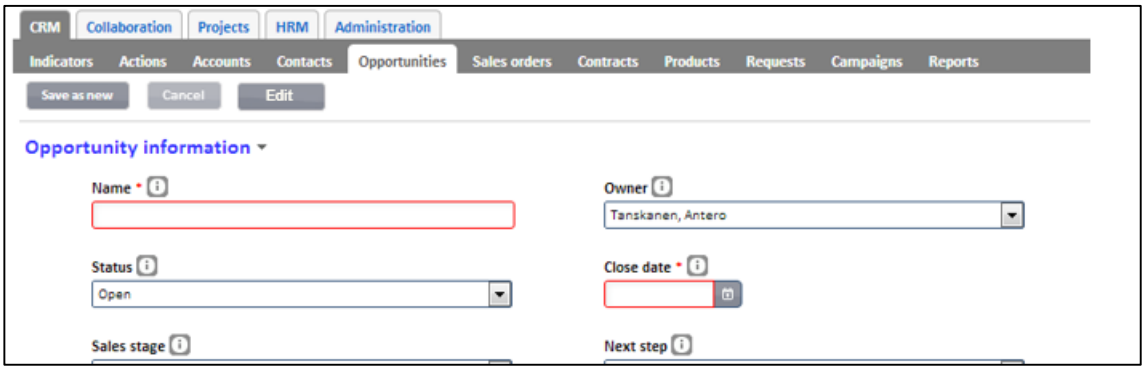
den projektin kesken siten, että projektikohtaiset kehityshaarat periytyivät samasta ylähaarasta, jolloin tehtyjen muutosten seuranta ja sulauttaminen näiden kahden välillä olisi mahdollisimman helppoa.

4.2 Prototyypit

Luvussa 3 esitetyn aineiston pohjalta suunniteltiin ja valmisteltiin Microsoft PowerPoint -sovellusta hyödyntäen muutama vaihtoehtoinen toteutushahmotelma eli prototyyppi, joissa kuvattiin, miten parametreihin liittyviä prosesseja voitaisiin parantaa PlanMill-järjestelmässä aiemmin mainittujen kehityskohteiden osalta (Luku 3.4, s. 20). Prototyyppejä suunniteltaessa pyrittiin ottamaan mahdollisimman hyvin huomioon olemassa olevien sivupohjien ja käyttöliittymäelementtien hyödyntäminen (Luku 2.1, s. 3). Prototyypit valmistettiin iteratiivisesti ja jotkin niistä perustuvat osittain toisiin aiemmin tehtyihin prototyyppeihin. On tärkeää huomioida, että tässä vaiheessa opinnäytetyön tekijällä oli vain omiin arvioihin perustuva käsitys eri prototyyppien toteuttamisen työmäärästä ja teknisestä vaativuudesta. Prototyyppien valmistuttua ne esiteltiin projektitiimille, jonka kesken arvioitiin eri vaihtoehtojen hyötyjä suhteessa työn teknisen toteutuksen vaativuuteen nähden.

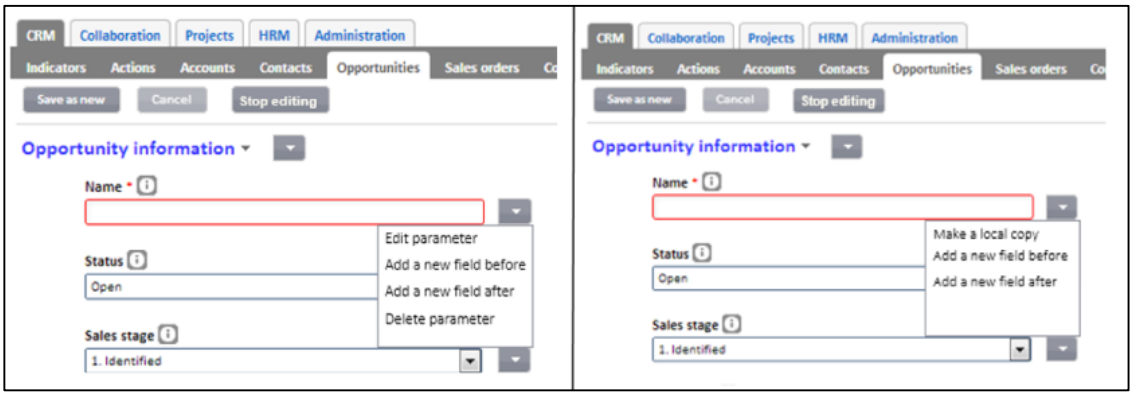
4.2.1 Prototyyppi 1 – Lomakekenttäkohtainen muokkausnäkymä

Ensimmäisessä prototyypissä ajatuksena oli, että lomakenäkymästä saa nopeasti ja helposti muokatuksi mitä tahansa lomakkeen kenttää tai lisätyksi uusia kenttiä.



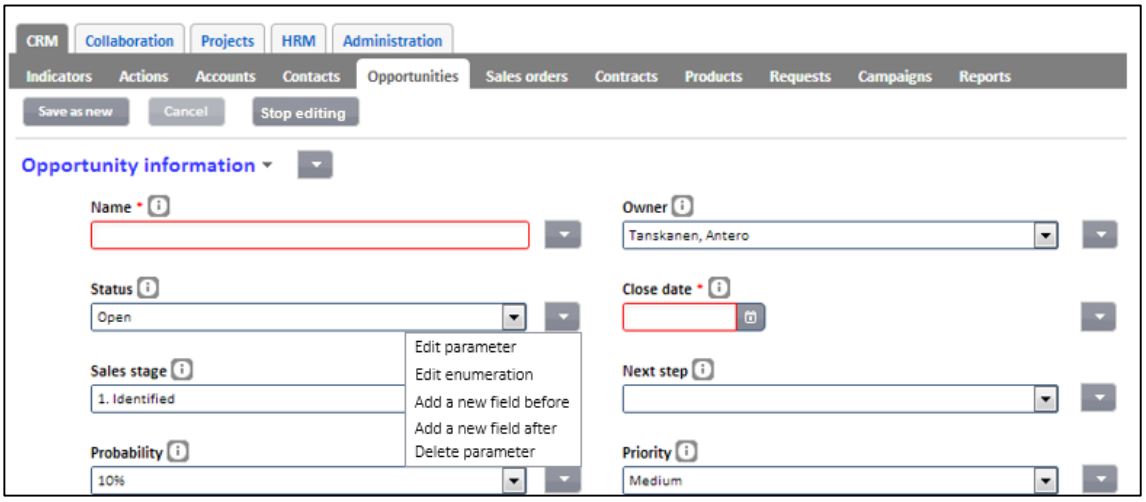
Kuva 15. Näkymä lomakkeesta ylläpitäjäroolilla tavallisessa tilassa.

Lomakenäkymään lisätään uusi ”Edit”-painike (Kuva 15), jota klikkaamalla siirrytään lomakkeen muokkaustilaan. Muokkaustilassa jokaisen lomakekentän yhteyteen ilmestyy muokkausvalikkopainike, josta aukeava valintalista tarjoaa työkalut kyseisen kentän muokkaamiseen ja uusien kenttien lisäämiseen joko ennen tai jälkeen kentän (Kuva 16). Mikäli halutaan muokata kenttää, jonka parametri tulee jaetusta tietokannasta, täytyy siitä ensin tehdä paikallinen kopio valitsemalla muokkausvalikosta ”Make a local copy” -toiminto.



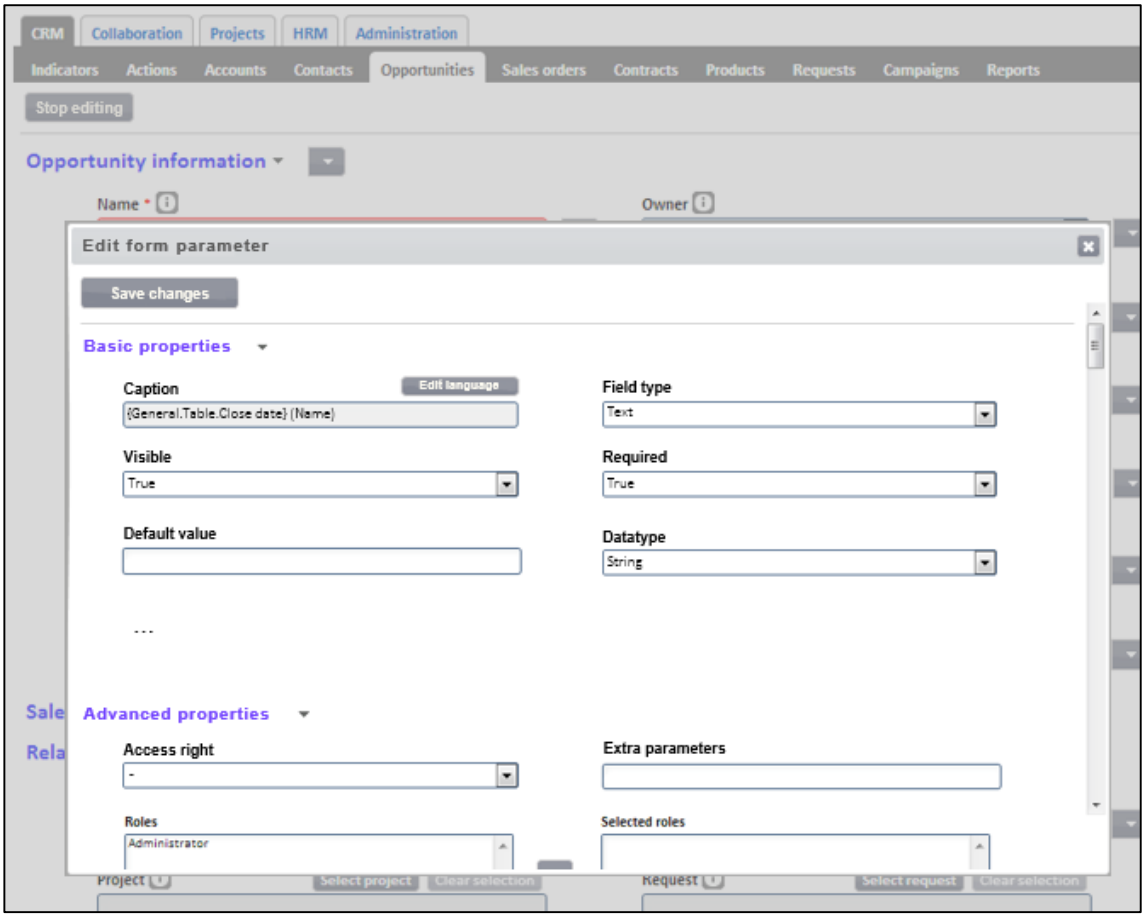
Kuva 16. Muokkausvalikon tarjoamat toiminnot kentälle, jonka parametri on paikallinen (vasemmalla) tai jaettu (oikealla).

Lisäksi pudotusvalikkotyyppisten kenttien muokkausvalikko tarjoaa mahdollisuuden siirtyä helposti muokkaamaan kenttään liittyvää enumeraatio-parametria valitsemalla ”Edit enumeration” -toiminto (Kuva 17).



Kuva 17. Muokkausvalikon tarjoamat toiminnot pudotusvalikkotyyppiselle kentälle.

Valitsemalla muokkausvalikosta ”Edit parameter” -toiminto (kuva 17), aukeaa uusi ikkuna, jossa lomakenäkymäsivupohjaa hyödyntäen tarjotaan yksinkertainen käyttöliittymä kyseisen kentän ominaisuuksien muokkaamiseen (Kuva 18).

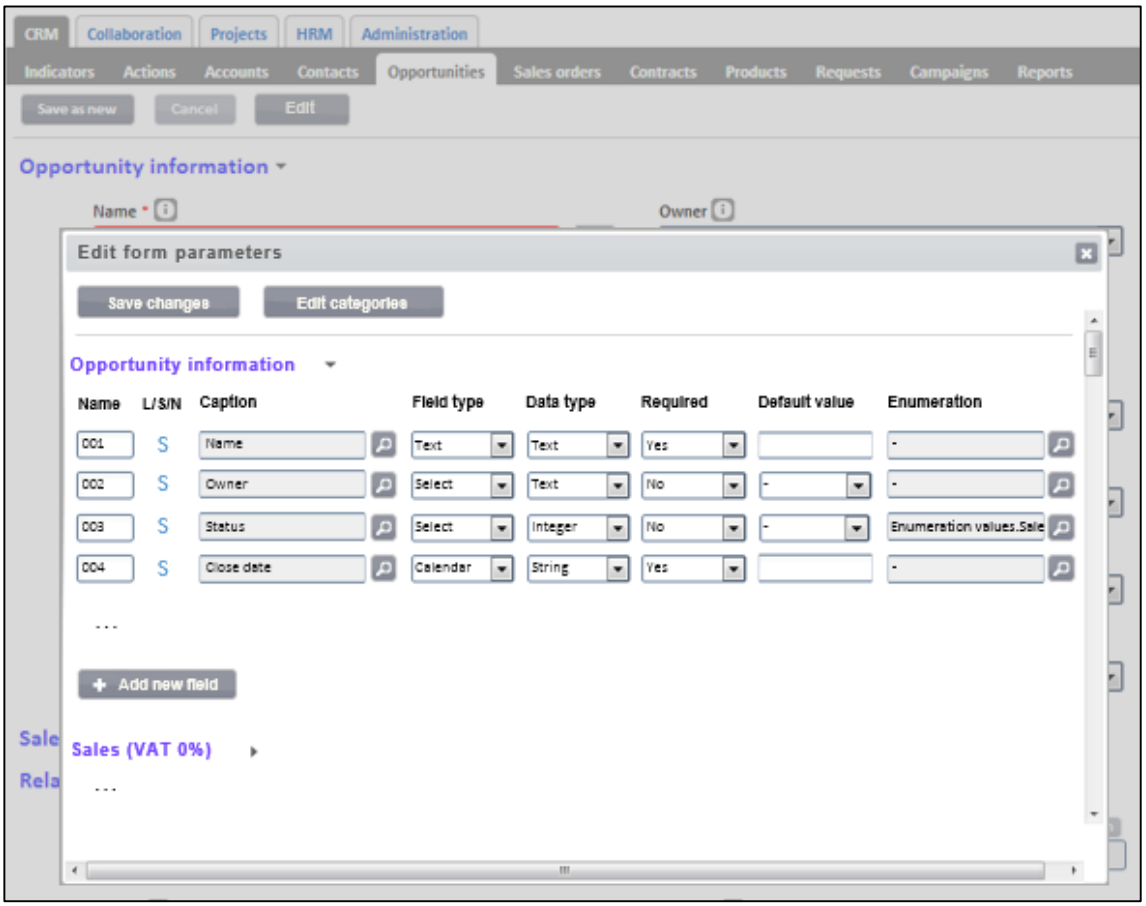


Kuva 18. Lomakekentän muokkauslomake.

Lomakekentän muokkauslomakkeella (kuva 18) ominaisuuksien muuttaminen tapahtuu tavallisia lomakekenttiä hyödyntäen, jolloin käyttäjän ei välttämättä tarvitse ymmärtää parametreissa käytettyä merkintäkieltä. Muokkauslomakkeen kentät on jaettu kahteen kategoriaan: 1) perusasetukset sekä 2) edistyneemmät asetukset. Edistyneempien asetusten alla on mahdollista muokata esimerkiksi kenttään liittyviä skriptejä sekä muokata käsin parametrin koko datasisältöä.

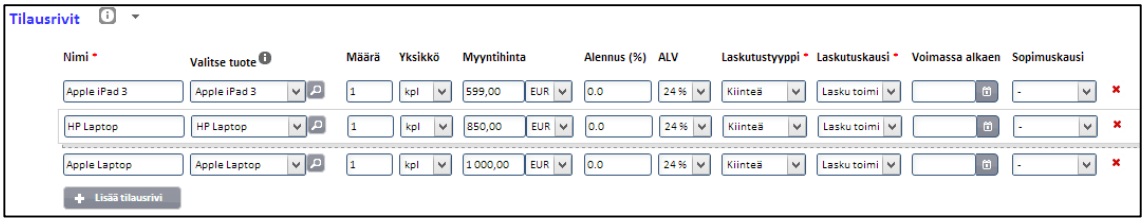
4.2.2 Prototyyppi 2 – Lomakekohtainen muokkausnäkymä

Toisessa prototyypissä lomakkeen kaikkia kenttiä voidaan muokata helposti yhdellä muokkauslomakkeella.



Kuva 19. Lomakkeen muokkausnäkymä, jossa hyödynnetty monirivivalitsin-käyttöliittymäelmmenttiä.

Lomakkeilla näkyy ”Edit”-painike, kuten ensimmäisessä prototyypissäkin (Kuva 15), jota klikkaamalla aukeaa lomakkeen muokkausnäkymä. Muokkausnäkymä tarjoaa yksinkertaisen käyttöliittymän muokata kaikkia lomakkeen kenttiä samassa näkymässä (Kuva 19). Näkymä pohjautuu olemassa olevan, niin sanotun, ”monirivivalitsin”-toiminnallisuuden hyödyntämiseen, joka mahdollistaa tietyntyyppisten entiteettien lisäämisen, poistamisen, muokkaamisen ja niiden keskinäisen järjestyksen muuttamisen (Kuva 20). Monirivivalitsimia käytetään muun muassa lomake- ja yhteenvetonäkymissä.

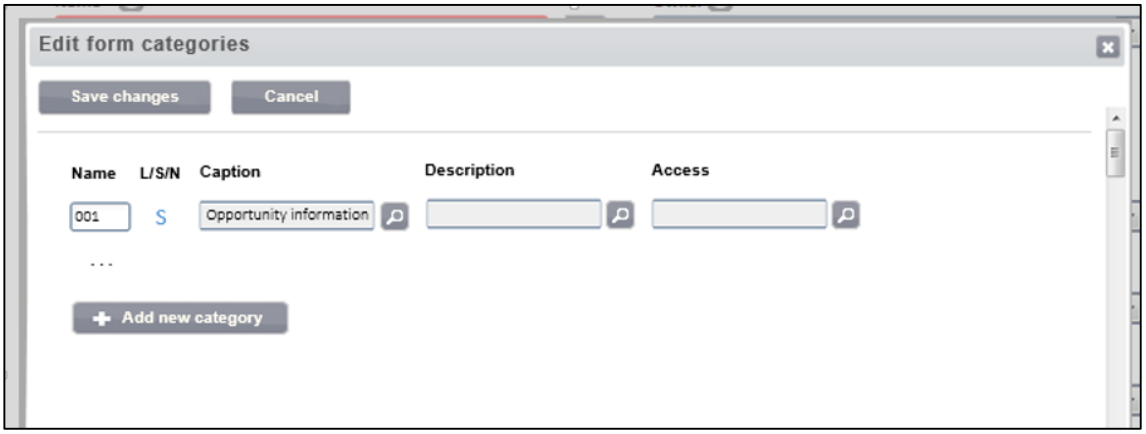


Kuva 20. Esimerkki monirivivalitsimesta myyntitilauslomakkeella.

Tämän prototyypin muokkausnäkymässä monirivivalitsimen avulla on mahdollista muuttaa lomakkeen kenttiin liittyvien parametrien perusominaisuuksia, kuten kentän

- otsikkoa
- tyyppiä
- tallennettavan tiedon tyyppiä
- pakollisuutta
- oletusarvoa.

Lisäksi rivillä näytetään parametrin nimi ja tieto siitä, onko parametri paikallinen (L = ”Local”), jaettu (S = ”Shared”) vai uusi (N = ”New”). Kenttä on uusi silloin, kun käyttäjä klikkaa ”Add new field” -painiketta, jolloin uuden kentän parametria voidaan käsitellä näkymässä, mutta sitä ei ole vielä tallennettu tietokantaan. Samaan kategoriaan kuuluvien kenttien keskinäistä järjestystä voidaan muuttaa klikkaamalla riviä ja siirtämällä sitä järjestyksessä ylös- tai alaspäin. Tällöin myös parametrin nimi muuttuu automaattisesti, sillä se määrää kenttien esiintymisjärjestyksen (vertaa Luku 3.1.2, kohta 3, s. 15). Paikallisia ja uusia parametreja voidaan poistaa samalla tapaa kuin muissa monirivivalitsimissa eli klikkaamalla punaista ruksia rivin lopussa. Muutokset tallentuvat tietokantaan klikatessa ”Save changes” -painiketta.

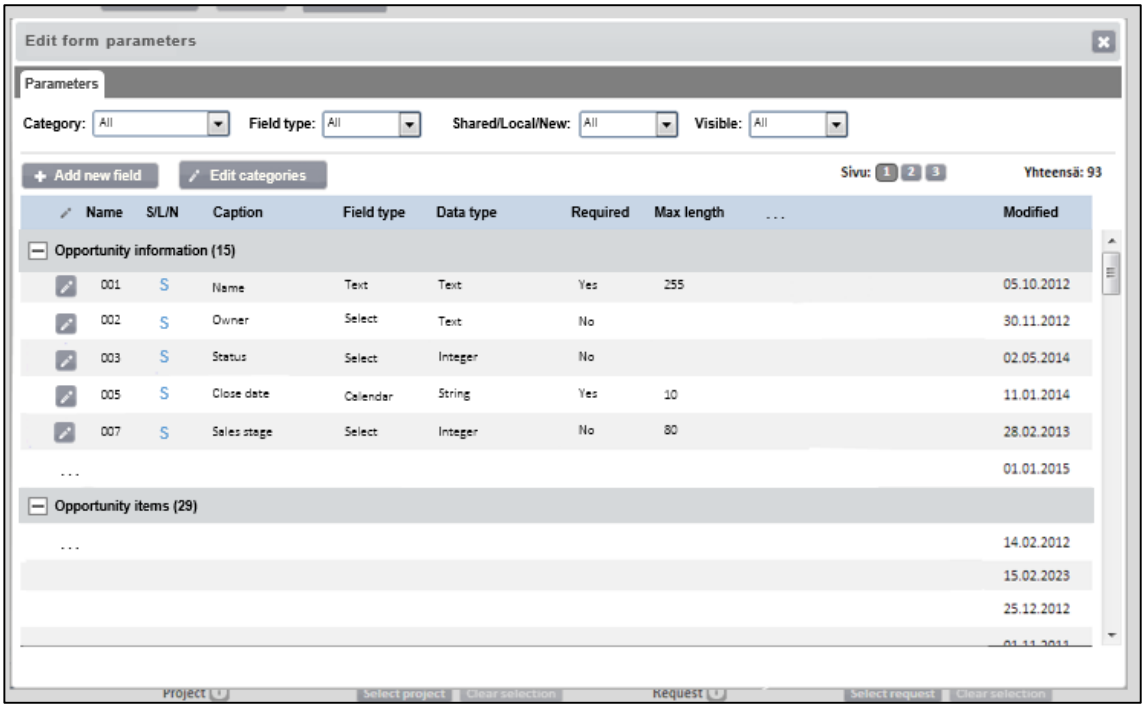


Kuva 21. Kategorioiden muokkausnäkymä.

Muokkausnäkymässä on lisäksi ”Edit categories” -painike, jonka kautta voidaan lisätä lomakkeelle uusia kategorioita (kuva 19). Tämä näkymä on toteutettu samalla tavalla monirivivalitsinta hyödyntäen (Kuva 21).

4.2.3 Prototyyppi 3 – Listanäkymäpohjaan perustuva lomakekohtainen muokkausnäkymä

Kolmannen prototyypin muokkausnäkymässä on hyödynnetty listanäkymäsivupohjaa ja siinä yhdistetään ideoita kahdesta edeltävästä prototyypistä (Kuva 22).



Kuva 22. Lomakkeen muokkausnäkymä käyttäen listanäkymäpohjaa.

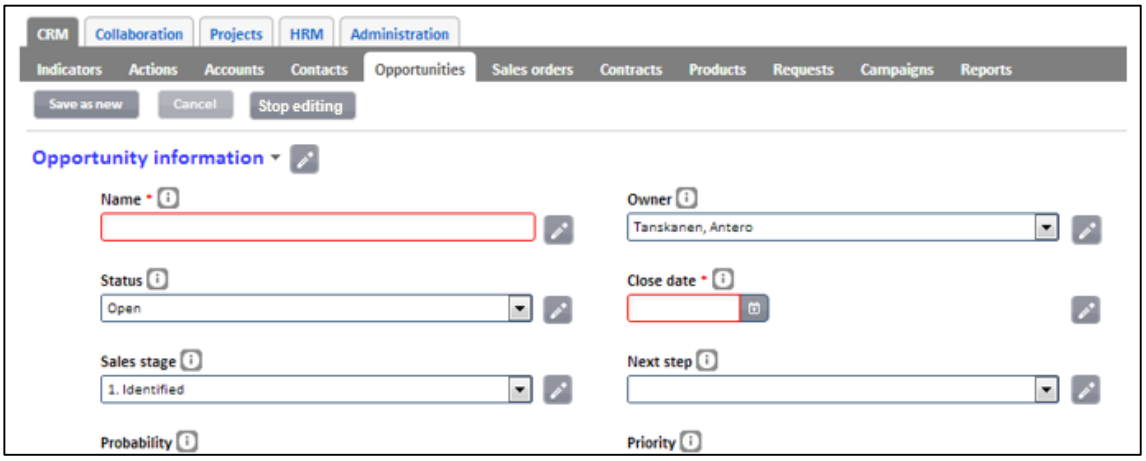
Toisen prototyypin mukaisesti voidaan muokkausnäkymässä tarkastella kaikkien parametrien perustietoja yhdellä kertaa. Parametreja pääsee muokkaamaan klikkaamalla rivin alussa olevaa kynäkuvaketta, joka avaa ensimmäisen prototyypin mukaisen yksinkertaistetun parametrin muokkauslomakkeen (Kuva 18). Lomakekategorioita muokataan toisen prototyypin mukaista lomaketta käyttäen (Kuva 21). Parametrin poistamista ei ole käyttöliittymäluonnoksessa huomioitu, mutta se voidaan toteuttaa kuten muissa listanäkymissä eli valitsemalla rivin ja klikkaamalla ”Delete selected” -painiketta. Kenttien keskinäistä järjestystä muutetaan tässä prototyypissä käsin parametrien nimeä muuttamalla.

Etuna listanäkymäsivupohjan hyödyntämisessä on se, että se mahdollistaa personoitavien suodattimien ja sarakkeiden hyödyntämisen. Suodattimien avulla voidaan entisestään helpottaa halutun kentän löytämistä. Personoitavia sarakkeita taas voidaan käyttää,

kun halutaan tarkastella jotain erityistä kenttään liittyvää tietoa, jota ei normaalisti haluta näkymässä näytettävän.

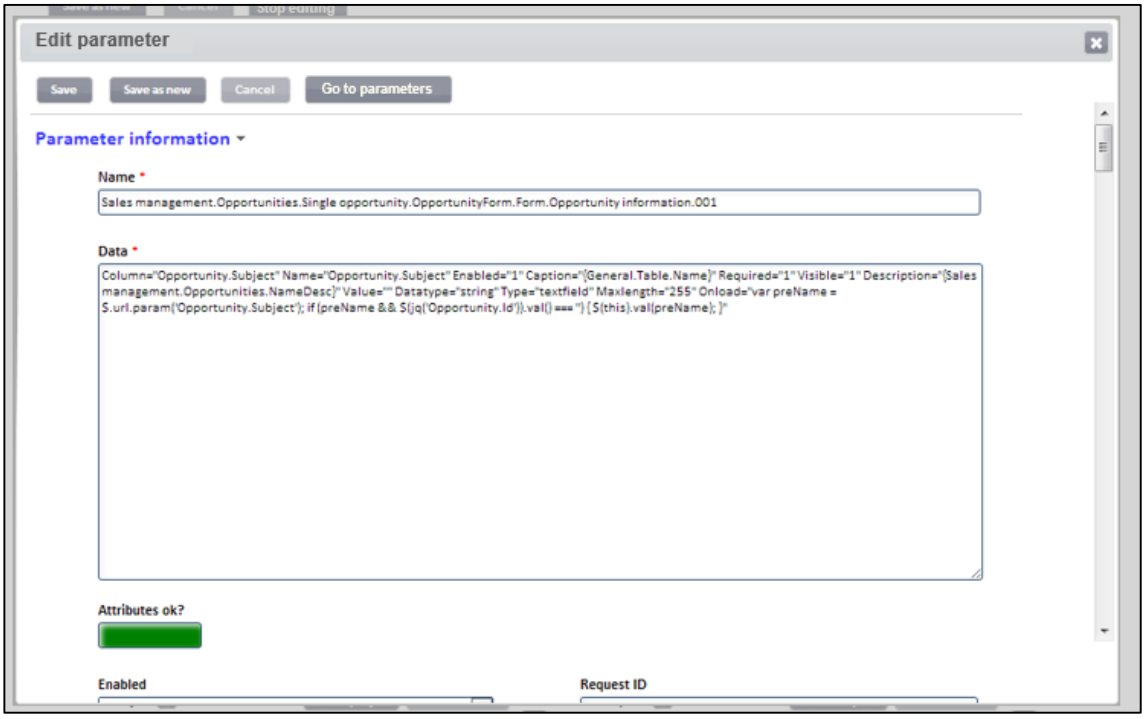
4.2.4 Prototyyppi 4 – Yksinkertaisempi toteutusvaihtoehto

Neljäs prototyyppi on huomattavasti yksinkertaisempi kuin aiemmat, ja se tehtiin tuoteomistajan esittämien ehdotusten pohjalta. Siinä ensimmäisen prototyypin mukaisesti lomakkeilla näytetään ylläpitäjäroolille ”Edit”-painike (kuva 15), jota klikkaamalla avautuu lomakkeen muokkaustila, jossa jokaisen lomakekentän yhteyteen ilmestyy muokkauspainike (Kuva 23).



Kuva 23. Lomake muokkaustilassa, jolloin lomakekenttien yhteyteen ilmestyy muokkauspainike.

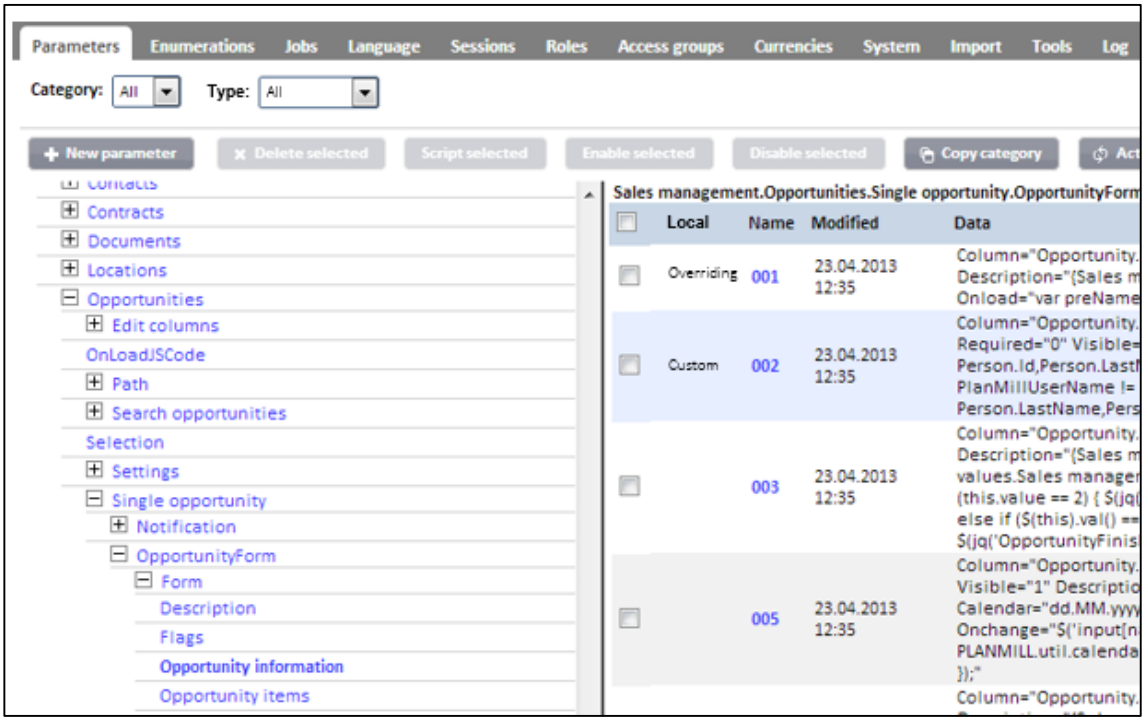
Muokkauspainiketta klikatessa avautuu vanha parametrin muokkauslomake peiteikkunassa, jossa on lisäksi ”Go to parameters” -painike, jota klikkaamalla Parametrit-välilehti aukeaa uuteen selaimen välilehteen (Kuva 24).



Kuva 24. Parametrin muokkauslomake avattuna peiteikkunaan.

Parametrit-välilehti näyttää sekä paikalliset että jaetut parametrit, ja ne voidaan suodattaa käyttäen suodatinta (Kuva 25). Lisäksi parametrilistassa on uusi ”Local”-sarake, jota tarkastelemalla nähdään, onko parametri

- jaettu (”Shared”)
- pelkästään paikallinen (”Custom”)
- paikallinen, joka ylikirjoittaa samannimisen jaetun parametrin (”Overriding”).



Kuva 25. Parametrit-välilehti, jossa näytetään myös jaetut parametrit.

Avaamalla jaettu parametri muokkauslomakkeelle, tallennusvaihtoehtona on ainoastaan ”Save as new”, jota klikkaamalla parametrista tallentuu paikallinen kopio. Kun jaetusta parametrista tehdään paikallinen kopio, niin jaettu parametri peittyy sen alle eikä näy enää käyttöliittymässä.

4.2.5 Päätös projektissa toteutettavista toiminnoista

Projektiryhmän kesken valittiin prototyyppi, josta lähdettiin tekemään toteutusta. Tuote-omistaja arvioi eri prototyyppien hyödyllisyyttä ja käytettävyyttä, kun taas projektiin allokoitu seniorikehittäjä arvioi teknisen toteutuksen vaativuutta.

Taulukko 1. Kriteeristö prototyyppien arvioimiseen.

Parannus	**Kriteerit**
Edistää parametrien löytämistä	• jaettujen parametrien tarkasteleminen asiakkaan instanssissa • vähentää parametrin muokkauslomakkeelle pääsyyn vaadittavien klikkausten määrää (instanssin sisällä) • helpottaa oikean parametrin löytämistä
Edistää parametrien muokkaamista	• data-arvon jäsennelty esittäminen • käyttöliittymäparametrin attribuuttien muokkaaminen käyttäen valmiita valinta-arvoja
Edistää parametrien luomista / järjestämistä	• käyttöliittymäparametrin automaattinen nimeäminen • vähentää käyttöliittymäparametrien järjestyksen muuttamiseen vaadittavien vaiheiden määrää • helpottaa paikallisen kopion tekemistä jaetusta parametrista
Edistää turvallisuutta / vähentää virheitä	• vähentää mahdollisuutta antaa virheellinen arvo parametrille • vähentää tarvetta käydä jaettujen parametrien instanssissa • poistaa kokonaan tarpeen käydä jaettujen parametrien instanssissa
Edistää staattisen analyysin tekemistä	• mahdollistaa parametrien listaamisen jonkin ehdon mukaan • äly, joka tunnistaa parametreissa olevia puutteita
Tukee asiakkaiden itse muokkaamista	• parametrin muokkaus käyttäen asiakkaille tuttuja käyttöliittymäelementtejä • rajoitetut muokkausvälineet
Edistää Kieli-entiteettien ja enumeraatio-parametrien käsittelyä	• mahdollistaa parametrissa viitatun Kieli-entiteetin tai enumeraatio-parametrin tarkastelun parametrin muokkauksen yhteydessä • vähentää parametrissa viitatun Kieli-entiteetin tai enumeraatio-parametrin muokkaamiseen tarvittavien klikkausten määrää

Taulukossa 1 on esitetty opinnäytetyön tekijän oman näkemyksen pohjalta tehdyt kriteerit, joiden perusteella projektissa valmisteltujen prototyyppien paremmuutta voidaan arvioida eri osa-alueilla. Taulukko 2 pohjautuu näihin kriteereihin.

Taulukko 2. Prototyyppien arviointi.

	1	2	3	4
Edistää parametrien löytämistä	***	***	***	***
Edistää parametrien muokkaamista	***	***	***	-
Edistää parametrien luomista/järjestämistä	***	***	*	*
Edistää turvallisuutta / vähentää virheitä	**	**	**	**
Edistää staattisen analyysin tekemistä	-	-	**	**
Tukee asiakkaiden itsemuokkaamista	**	***	**	-
Edistää Kieli-entiteettien ja enumeraatio-parametrien käsittelyä	***	***	***	-
Toteutuksen tekninen vaativuustaso	XXX	XXX	XXXX	XX

- = ei täytä yhtään kriteeriä, * = täyttää alle puolet kriteereistä, ** = täyttää yli puolet kriteereistä, * = täyttää kaikki kriteerit**

Taulukossa 2 on esitetty, miten eri prototyypit vastaavat kriteerejä sekä oma arvio teknisen toteuttamisen vaativuudesta, josta kertoo X-merkkien lukumäärä.

Kolme ensimmäistä prototyyppiä olivat opinnäytetyön tekijän käsialaa ja neljäs prototyyppi tehtiin vasta sen jälkeen, kun oli todettu, että kolme aiempaa prototyyppiä ovat liian työläitä tehtäväksi projektin toteutukselle varatussa ajassa. Lopulta päätettiin, että toteutettavan ratkaisun on hyvä koostua pienemmistä toisistaan riippumattomista osasista, jolloin ajan loppuessa kesken voidaan kuitenkin osa toteutetuista toiminnallisuuksista siirtää tuotantoon.

Lisäksi päädyttiin etsimään ratkaisuja, jotka ovat yleispäteviä ja hyödyttävät myös järjestelmän kehittämisessä sekä vian etsinnässä. Neljäs prototyyppi valmisteltiin näiden seikkojen pohjalta ja siksi se edusti eniten haluttua tavoitetilaa; vaikkakin se kohdistuu vähiten alkuperäisiin kehityskohteisiin (luku 3.4, s. 20). Siinäkin todettiin heti ilmeisiä puutteita ja selvää oli alusta alkaen, että toteutuksen sisältö tulee muuttumaan projektin aikana sitä mukaan, kun uusia ideoita syntyy ja tehdyissä määritelmissä huomataan puutteita.

Sprintin alkaessa tuotteen kehitysjonoon oli päädytty laittamaan seuraavat asiat:

- Tekstihaku, tai paremminkin suodatin, Parametrit-välilehdelle, jonka avulla voidaan rajata parametripuun kokoa ja listattavien parametrien lukumäärää.

- Jaetut parametrit näytetään asiakkaiden instansseissa Parametrit-välilehdellä, jolloin jaettujen parametrien instanssiin ei tarvitse enää kirjautua kuin silloin, kun halutaan tehdä sinne muutoksia.
- Uusi ”Local”-sarake Parametrit-välilehdelle, joka indikoi, onko parametri paikallinen vai jaettu (neljännessä prototyypissä esitellystä kolmijaottelusta luovuttiin).
- Suodatin Parametrit-välilehdelle, jolla voidaan valita näytettäväksi kaikkien parametrien sijaan vain jaetut tai paikalliset parametrit.
- Jaettu parametri voidaan avata muokkauslomakkeelle kuten paikallinenkin, mutta tallennusvaihtoehtona on vain ”Save as new”, jolloin parametrista tallennetaan paikallinen kopio.
- Lomakkeen muokkaustila, jossa lomakkeen kenttien yhteyteen ilmestyy painikkeet, joiden kautta päästään suoraan tarkastelemaan kyseistä parametria Parametrit-välilehdellä.

4.3 Arkkitehtuuriset päätökset uusiin toimintoihin liittyen

Projektin aikana jouduttiin tekemään muutamia perustavanlaatuisia arkkitehtuurisia päätöksiä uusiin toimintoihin liittyen. Tärkeimmät kysymykset liittyivät jaettujen parametrien lukemistapaan sekä tekstihaun toteuttamiseen. Nämä seikat herättivät jonkin verran keskustelua yrityksessä työskentelevien teknisten asiantuntijoiden välillä, ja lopullisista ratkaisuista ei oltu täysin yksimielisiä.

4.3.1 Jaettujen parametrien lukeminen

Jaettujen parametrien näyttämiseen asiakasinstansseissa oli pari vaihtoehtoista arkkitehtuurista toteutustapaa. Ensimmäinen tapa oli tehdä rakenteellisia muutoksia sovelluspalvelimen Java-koodiin ja lukea jaetut parametrit sovelluspalvelimen muistista. Toinen tapa oli toteuttaa muutos muokkaamalla parametrien listaamiseen käytettyjä tietokantakyselyitä siten, että projektin alkutilanteen pelkkien asiakkaan tietokannassa sijaitsevien paikallisten parametrien lisäksi myös vakioparametrit luetaan jaetusta tietokannasta.

Ensimmäisen tavan etuja on, että muistista lukeminen on huomattavasti nopeampaa ja kuormitusta ei synny tietokantapalvelimelle. Haittapuolina taas on se, että sovelluspalvelimelta vaaditaan enemmän muistiresursseja, sillä nykytilanteessa muistissa on vain aktiiviset parametrit, mutta muutoksen jälkeen sinne tallennettaisiin kaikki, sekä aktiiviset

että passiiviset, asiakaskohtaiset ja jaetut parametrit. Lisäksi toteuttaminen vaatii järjestelmän syvällistä tuntemista, jotta sovelluspalvelimen muistista luetut jaetut parametrit saadaan liitettyä tietokannasta luettaviin paikallisiin parametreihin. Myös konfigurointimahdollisuudet käyttöönoton jälkeen ovat melko rajalliset, mikä koettiin hieman ongelmallisena ajatuksena, sillä haluttiin mahdollisuus kytkeä toiminnallisuus helposti pois päältä asiakaskohtaisesti, mikäli käyttöönotto epäonnistuisi.

Toisen tavan, missä tietokantakyselyitä muokkaamalla myös jaetut parametrit luetaan mukaan parametrien listaukseen, etuina on, että se on huomattavasti helpommin ja nopeammin toteutettavissa, ja se mahdollistaa helpon konfiguroinnin parametrien avulla. Suuria muutoksia palvelimen Java-koodiin ei tarvitse tehdä, ja siten tämä tapa on käyttöönoton kannalta turvallisempi ratkaisu. Haittapuolina taas on, että tietokantakyselyjen tekeminen on todennäköisesti huomattavasti hitaampaa kuin muistista lukeminen Java-koodissa, ja se kuormittaa sitä tietokantapalvelinta, jolla jaettu tietokanta sijaitsee. Tästä seuraa myös se, että vaikka kehitysympäristössä pystyttäisiin todentamaan hyvä suorituskyky ja lyhyet viiveajat, niin ei ole varmuutta siitä, miten toteutus käyttäytyy tuotantoympäristössä, jossa palvelinkokoonpano eroaa testiympäristöstä, ja jossa on paljon samanaikaisia käyttäjiä. Erityisesti askarrutti suorituskyky niissä instansseissa, joiden asiakaskohtainen tietokanta on eri palvelimella kuin jaettu tietokanta.

Päätös toteutustavasta tehtiin projektitiimiin kuuluvan seniorikehittäjän toimesta. Päätettiin toteuttaa toiminnallisuus parametreja hyödyntämällä, kun otettiin huomioon opinnäytetyön tekijän kokemattomuus ja projektille varattu lyhyt aika. Ajateltiin, että on parempi saada jokin toimiva ratkaisu aikaiseksi annetussa ajassa sen sijaan, että syntyisi vain keskeneräinen ratkaisu. Luotettiin myös siihen, että tietokantakyselyitä optimoimalla saadaan kohtuullisen hyvä suorituskyky aikaiseksi.

4.3.2 Tekstihaun toteuttaminen

Tekstihaku parametreille oli toinen mietinnän kohde pitkälti samoista syistä kuin jaettujen parametrien näyttäminenkin, eli toiminnon raskaus ja siitä aiheutuvat viiveajat. Tekstihaku voitaisiin toteuttaa käyttämällä kahta eri SQL Serverin tukemaa hakutekniikkaa; 1) täystekstihakua (”f*ull text search”*) tai 2) niin sanottua villikorttihakua (”*wild card search”*) [9].

Täystekstihaku on pitkälti optimoitu hakutekniikka, joka perustuu indeksoituihin tietokantatauluihin. Tämän hyödyntäminen siis edellyttää, että tietokantataulun sarakkeet, joihin haku kohdistuu, ovat indeksoitu. Lähtötilanteessa parametrien tietokantatauluja ei oltu indeksoitu ja täystekstihaun hyödyntäminen edellyttäisi kyseisen taulun indeksoimista kaikissa tietokannoissa. Täystekstihakujen ehdottomana etuna on nopeus, mutta tekniikan ongelma on, että se ei tue sanan osan löytämistä muualta kuin sanan alusta. Esimerkiksi, jos haettaisiin sanalla ”Mill”, niin vastaavuutta ei löytyisi sanasta ”PlanMill”, mutta ”Millionaire” löytyisi. Täystekstihakuja käytetään pääasiallisesti PlanMill-järjestelmän tekstihauissa, mutta se ei sovellu parametrihakuun kovin hyvin, sillä parametreissa on usein pitkiä yhtäjaksoisia ”sanoja”, joiden keskeltä halutaan hakea. Koodiesimerkissä 1 on esitetty vastaava haku toteutettuna täystekstihakuna ja villikorttihakuna.

```
-- Full Text Search
SELECT
    *
FROM
    Parameter
WHERE
    CONTAINS(Name, '"Sales*" AND "management*"')
    OR CONTAINS(Data, '"Sales*" AND "management*"')

-- Wildcard search
SELECT
    *
FROM
    Parameter
WHERE
    (Name LIKE '%Sales%' AND Name LIKE '%management%')
    OR (Data LIKE '%Sales%' AND Data LIKE '%management%')
```

Koodiesimerkki 1. Vastaava haku tehtynä täystekstihakuna ja villikorttihakuna.

Villikorttihaku on raskas, mutta perusteellinen hakutekniikka. Sillä voidaan hakea sanan osia myös keskeltä sanaa, ja tästä syystä se soveltui paremmin käytettäväksi parametrien tekstihaussa tekniikan raskaudenkin uhalla. Lisäksi sen käyttöönotto ei vaatinut muutoksia tietokantatauluihin.

4.4 Toteutetut ominaisuudet

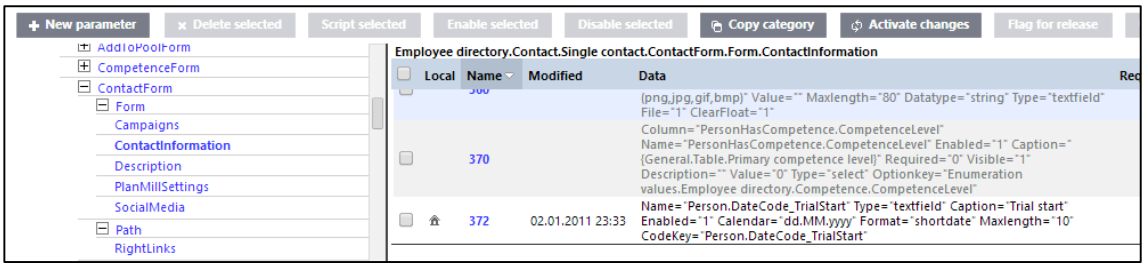
Projektin alussa opinnäytetyön tekijällä oli järjestelmästä melko pinnallinen käsitys, johon kuului paljon niin sanottuja harmaita alueita. Siksi toteutusta tehtäessä aikaa kului paljon selvittelyyn, kokeiluihin ja järjestelmän arkkitehtuuriin paneutumiseen koodia tutkimalla.

Toteutusta tehtäessä asetettiin Scrumin mukaisesti pieniä tavoitteita, joissa onnistuminen ohjasi työn kulkua: alussa oli, olosuhteet huomioon ottaen, erittäin vaikea arvioida, mihin tavoitteisiin lopulta päästäisiin. Alusta alkaen kuitenkin pyrittiin löytämään ratkaisuja, jotka olisivat helppoja ja nopeita toteuttaa, mutta joilla olisi suuri vaikutus käytettävyyteen. Tuotteen kehitysjono kasvoikin projektin myötä kun uusia ongelmakohtia havaittiin ja uusia ideoita syntyi. Projektissa toteutettavat ominaisuudet myös muuttuivat alkuperäisistä sitä mukaan kun kehitysjonossa olevia ominaisuuksia priorisoitiin uudelleen. Tässä luvussa on esitelty lopulta toteutetut ominaisuudet ja kerrotaan niiden toteuttamiseen liittyneistä seikoista.

4.4.1 Jaettujen parametrien näkyminen Parametrit-välilehdellä

Jaettujen parametrien näkyminen Parametrit-välilehdellä oli yksi alkuperäisessä tuotejonossa olevista asioista ja yksi varmimmin toteutettavista, sillä sen hyödyt olivat kaikista ilmeisimmät. Tätä ominaisuutta varten kehitetyt toiminnallisuudet myös avasivat tien muille uusille ominaisuuksille, kuten jaettujen parametrien hakemiselle käyttäen portaalihakua, josta puhutaan myöhempänä.

Lopullinen toteutus Parametrit-välilehden osalta sisälsi jaettujen parametrien näyttämisen lisäksi uuden ”Local”-sarakkeen, jossa pieni taloa esittävä ikoni indikoi, että parametri on paikallinen (Kuva 26). Lisäksi jaettujen parametrien kohdalla tekstin väriä himmennettiin, jotta erottaminen olisi mahdollisimman helppoa.



Kuva 26. Jaettujen parametrien näyttäminen Parametrit-välilehdellä.

Lopullisessa toteutuksessa jaetut parametrit oli mahdollista avata muokkauslomakkeelle, kuten paikallisetkin, mutta ainoa tarjottu tallennusvaihtoehto oli uutena tallentaminen, jolloin jaetusta parametrista tallentui kopio asiakkaan tietokantaan. Lisäksi joihinkin Parametrit-välilehden työkaluihin, kuten Parametrin poisto, lisättiin tarkistus, joka näyttää käyttäjälle virheilmoituksen, mikäli käyttäjä yrittää poistaa jaetun parametrin.

Jaettujen parametrien näyttäminen asiakasinstanssissa parametreissa määritettyjä tietokantakyselyitä muuttamalla on sinänsä erittäin yksinkertainen toimenpide, mutta tämän ominaisuuden ympärille tuli kehittää vielä liuta muuta toiminnallisuutta, jotta ominaisuus olisi valmis käyttöönotettavaksi tuotannossa. Suurimpia mietinnän aiheita tätä ominaisuutta toteutettaessa olivat

- tietokantakyselyjen optimoiminen
- jaetun tietokannan osoitteen lukeminen parametrista ja sen liittäminen tietokantakyselyyn
- jaettujen parametrien tunnistaminen ja käsittely asiakas- ja palvelinpuolella
- konfiguroiminen, eli toiminnallisuuden kytkeminen päälle tai pois päältä roolikohtaisesti.

Tietokantakyselyjen optimoimista tapahtui monessa vaiheessa projektin aikana ja viimeiseksi vielä tuotantoon siirtymisen jälkeen. Koodiesimerkissä 2 on nähtävissä eräs kehitysversio parametrilistauksessa käytetyn tietokantakyselyn määrittävästä parametrista hieman karsittuna. Siinä on keltaisella värillä korostettu kohdat, joiden käsittely jouduttiin lisäämään uutena palvelinpuolen Java-koodiin. Kyseinen parametri määrittää SQL-skriptin, jossa tehdään kaksi kyselyä: ensimmäinen asiakkaan tietokantaan ja toinen uutena jaettuun tietokantaan.

```
SELECT
    {columns}
FROM (
    SELECT
        {columns}
    FROM
        (SELECT
            Id
            ,Name
            ,Data
            ,Description
        FROM
            Parameter
        ) AS Parameter
    WHERE {where}

[accessshared:'Administration.Parameters view shared':'
    UNION

    SELECT
        {columns}
    FROM
        (SELECT
            -Id
            ,Name
            ,Data
            ,Description
        FROM
            {share}.dbo.Parameter
        ) AS Parameter
    WHERE {where}
']
    ) AS Parameter ORDER BY {orderby};
```

Koodiesimerkki 2. Parametrilistauksen määrittävä SQL-parametri hieman karsittuna.

Toiminnallisuutta varten PlanMill-merkintäkieleen toteutettiin pari uutta paikkamerkkiä ”[accessshared]” ja ”{share}”, jotta pystyttäisiin kontrolloimaan jaettujen parametrien näyttämistä roolikohtaisesti sekä lukemaan jaetun tietokannan osoite dynaamisesti parametrista. Mahdollisuus kontrolloida jaettujen parametrien näkymistä roolikohtaisesti oli tärkeä, sillä haluttiin helppo tapa kytkeä toiminnallisuus pois päältä, mikäli se osoittautuisi joissain asiakasinstansseissa käyttökelvottomaksi. Lisäksi parissa erityistapauksessa asiakkaan käyttäjäroolilla on pääsy Parametrit-välilehdelle ja jaettujen parametrien arkaluontoisuuden takia haluttiin estää niiden näkyminen.

```
...
// if it is an accesshared, do this special treatment to avoid the situation where user has
// 'View shared' -access right but instance doesn't have shared database
boolean isAccessShared = ParamConst.PLACEHOLDER_ACCESS_SHARED.contains(regexMatcher.group(1));
boolean isShareConflict = false;
if (isAccessShared && ParamControl.getSystemParameter(ParamControl.SHARED_SOURCE) == null) {
    isShareConflict = true;
}
// Check the access' and if ok, add the sql-code, otherwise add
// the false sql-code or, if not given, an empty string
if (ModuleHelper.checkAccess(scont, regexMatcher.group(2), isAccessAll) && !isShareConflict) {
    regexMatcher.appendReplacement(buffer, regexMatcher.group(3));
} else {
    ...
```

Koodiesimerkki 3. AccessShared-paikkamerkin käsittelyyn liittyvää koodia.

Jaettujen parametrien näkymisen kontrolloimista varten määritettiin oma ”View shared” -käyttöoikeus ja kehitettiin uusi paikkamerkki ”[accessshared]”, joka toimii muuten samoin kuin aiemmin työssä esitelty ”[access]”-paikkamerkki, mutta jossa on lisäksi tarkistus siitä, onko kyseessä oleva instanssi jaetun tietokannan instanssi (Koodiesimerkki 3). Kyseisessä instanssissa ei luonnollisesti ole määritetty jaetun tietokannan parametria ja ilman lisätarkistusta syntyisi virhe.

Kuva 27. Share-paikkamerkin korvaaminen JDBC-osoitteesta saaduilla tiedoilla

Koska jaetun tietokannan osoite on määritetty JDBC-osoitteena parametrissa, tuli sen lukemista, muuntamista ja tietokantakyselyyn lisäämistä varten kehittää jonkin verran lisälogiikkaa. Tätä varten parametreihin kehitettiin uusi paikkamerkki ”{share}”, joka korvattiin Java-koodissa niin sanotulla ”*four part naming*” -skeemalla, jolla SQL Server -ympäristössä voidaan viitata etäpalvelimella olevaan tietokantatauluun (kuva 27).

```sql
DECLARE @source nvarchar(255);

-- Share system parameters
SET @source = dbo.PMSfn_getParameter(N'System.Shared database.source', @PersonId);
SET @source = REPLACE(@source, N'=', N':');
SET @source = REPLACE(@source, N'//', N'');
SET @source = REPLACE(@source, N'jdbc:', N'');

IF @source IS NOT NULL
BEGIN
    IF CHARINDEX(N'jtds:', @source) = 1 -- JTDS-driver
    BEGIN
        SET @source = REPLACE(@source, N'jtds', N'');
        SET @source = REPLACE(@source, N'/', N'].[');
        SET @source = N'[' + dbo.PMSfn_ExtractElementFromData(@source, N'sqlserver') + ']';
    END;
    ELSE
    BEGIN -- MS-driver
        SET @source =
            N'[' + dbo.PMSfn_ExtractElementFromData(@source, N'sqlserver') +
            N'].[' + dbo.PMSfn_ExtractElementFromData(@source, N'databaseName') +
            N']';
    END;

END;
```

Koodiesimerkki 4. SQL-skripti, jolla parsitaan JDBC-osoitteesta palvelimen ja tietokannan nimi.

Logiikka tähän oli jo olemassa osana erästä tietokantaproseduuria, mutta toiminnallisuutta varten siitä kehitettiin oma funktio (Koodiesimerkki 4).

Jaettujen parametrien erottaminen asiakaskohtaisista parametreista oli myös yksi olennainen seikka, joka piti ratkaista, sillä asiakaskohtaisessa ja jaetussa tietokannassa voi olla parametreja samalla id-numerolla. Parametrien erottavaa tietoa tarvittiin myös parametrilistauksessa, jotta pystyttiin indikoimaan, mitkä parametrit ovat paikallisia ja mitkä jaettuja. Lisäksi tieto tarvittiin parametrin muokkauslomakkeen lataamisessa. Ensimmäinen ratkaisu ongelmaan oli, että parametrilistauksessa käytetty SQL-kysely palautti alkuperäisten sarakkeiden lisäksi ylimääräisen sarakkeen, jonka arvo kertoi, onko parametri paikallinen (1) vai jaettu (0). Sitten esimerkiksi parametrin muokkauslomakkeelle vieviin URL-osoitteisiin lisättiin ”Local” GET-parametri (Kuva 28) ja sitä varten toteutettiin käsittely Java-koodiin.

1. versio
index.jsp?category=Administration.Parameters.SingleParameter&Id=494692&Local=0

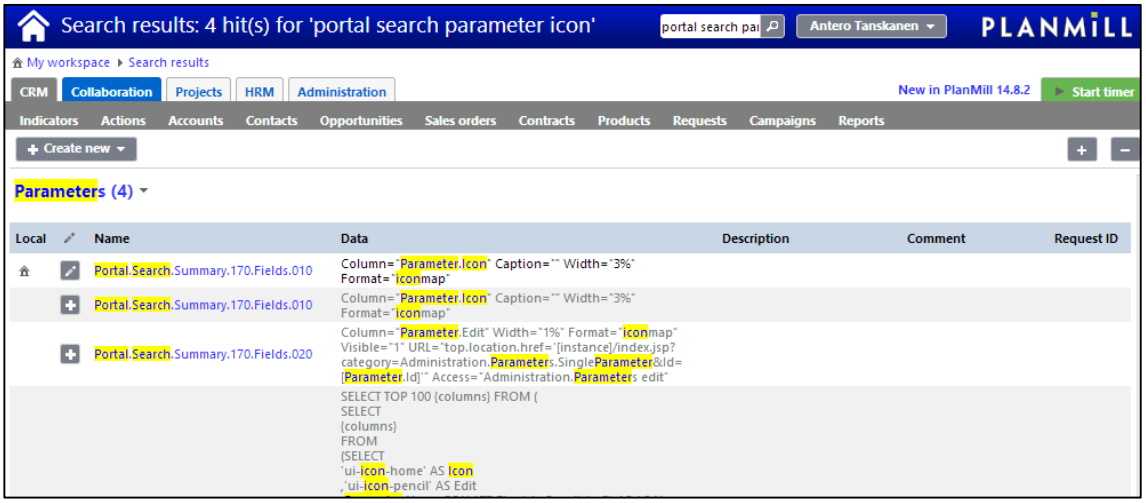
2. versio
index.jsp?category=Administration.Parameters.SingleParameter&Id=-494692

Kuva 28. Alkuperäinen (1.) ja lopullinen (2.) versio jaetun parametrin erottamisesta paikallisesta muokkauslomakkeen URL-osoitteessa.

Lopulta kuitenkin todettiin, että jaettujen parametrien tunnistaminen käyttämällä niiden id-numerossa negatiivista kokonaislukua positiivisen sijaan (Kuva 28), helpottaa erottelun käsittelyä Java-koodissa sekä parametreissa, ja se tekee kyselystäkin siistimmän näköisen (Koodiesimerkki 2).

4.4.2 Portaalihaun käyttäminen myös parametrien hakemiseen

Portaalihaku on nimensä mukaisesti koko portaalin laajuinen haku, jolla voidaan hakea hakutermeille vastaavuutta missä tahansa moduulissa esiintyvistä termeistä. Portaalihaku on käyttäjälle aina näkyvissä käyttöliittymän oikeassa yläkulmassa (kuva 29). Portaalihaun käyttämiseen parametrien etsimisessä päädyttiin sen jälkeen, kun tuoteomistajan kanssa oli todettu, että 1) Parametrit-välilehdelle olisi vaikea toteuttaa täysverinen hakutoiminnallisuus nykyisten rakenteiden puitteissa ja 2) haku saataisiin toteutettua erittäin helposti parametritasolla hyödyntämällä valmista portaalihakutoiminnallisuutta. Lisäksi tarvittavat toiminnallisuudet jaettujen parametrien näyttämiseksi asiakasinstanssissa oli jo toteutettu osana Parametrit-välilehden muutoksia, joten ominaisuus saatiin helposti tukemaan myös jaettujen parametrien etsimistä. Lisäksi tämä ominaisuus korvaa hieman alun perin suunniteltua, mutta lopulta pois jätettyä ”lomakkeen muokkaus tila” -toiminnallisuutta, jossa lomakkeilta olisi suora pääsy lomakkeen parametreihin.



Kuva 29. Parametrien haku käyttäen portaalihakua. Keltaisella korostetut alueet näkyvät myös käyttöliittymässä.

Käyttöönottovaiheessa portaalihaku parametrien osalta näytti Kuva 29 nähtävät sarakkeet, joihin myös haku kohdistui. Uusien kenttien lisääminen myöhemmin on helppoa. Hakutuloksista pääsee nopeasti Parametrit-välilehdelle tarkastelemaan kyseisen parametrin kategoriaa klikkaamalla parametrin nimeä. Myös muokkauslomakkeelle pääsee suoraan klikkaamalla parametrin nimeä edeltävää kynä- tai plus-kuvaketta riippuen siitä, onko kyseessä paikallinen vai jaettu parametri. Tuoteomistaja koki erottelevat kuvakkeet tärkeiksi, koska jaetun parametrin tapauksessa ei muokata jaettua parametria, vaan siitä tehtävää paikallista kopiota.

4.4.3 Personoitavat suodattimet ja tekstihaku Parametrit-välilehdelle

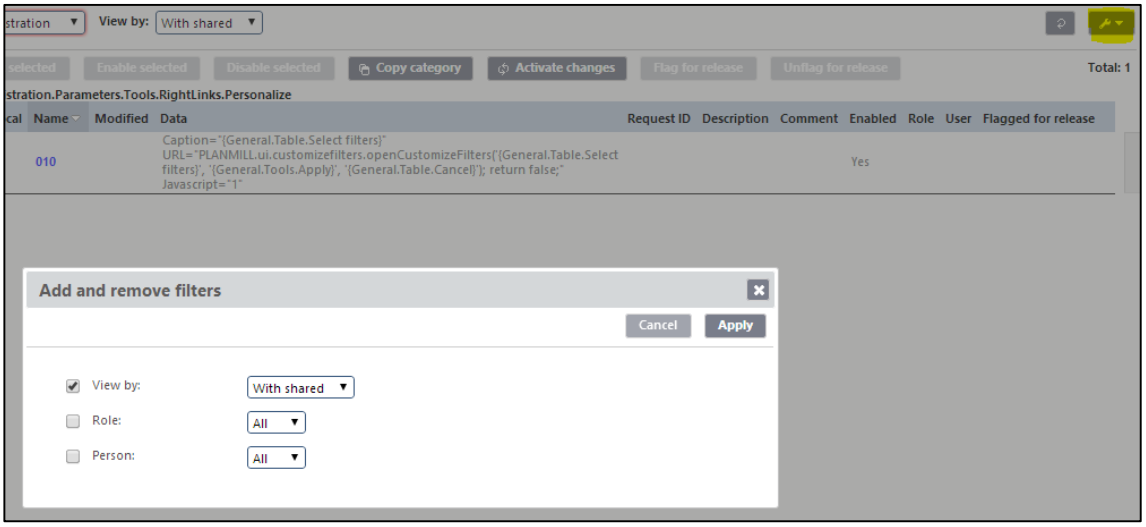
Parametrit-välilehdellä oli entuudestaan olemassa kategoria-suodatin, jonka perusteella luultiin, että kyseinen listanäkymä tukee suodattimia siinä missä muutkin listanäkymät. Osoittautui kuitenkin, että palvelinpuolen Java-koodissa Parametrit-moduuli oli toteutettu tukemaan pelkästään kategoria-suodatinta. Tästä syystä projektissa toteutettavaksi jäi myös suodatintuen toteuttaminen Parametrit-välilehdelle. Olennaisimmat muutokset toteutettiin moduulin koodiin, mutta muutoksia piti tehdä myös pariin asianomaiseen XSL-pohjaan. Moduulin koodiin tehtävät muutokset olivat periaatteessa helppoja tehdä, sillä sinne tuli vain lisätä käsittely ”{where}”-paikkamerkille, joka oli toteutettavissa olemassa olevia aliohjelmia hyödyntäen; yksi tavallisten suodattimien käsittelyä varten ja toinen tekstihakutoiminnallisuutta varten. Tämän lisäksi kyseinen paikkamerkki tuli lisätä parametrilistaukseen liittyviin tietokantakyselyihin (Koodiesimerkki 2, s. 41). Nyt tilanteen teki kuitenkin erilaiseksi se, että tietokantakyselyssä oli kaksi kertaa kyseinen paikkamerkki,

eivätkä sen hetkiset paikkamerkin korvaamiseen käytetyt aliohjelmat suoraan tukeneet kahden paikkamerkin korvaamista. Tämä erityiskäsittely jouduttiin lisäämään Javassa Parametrit- ja Portaalihaku-moduulin koodiin (Koodiesimerkki 5).

```
private String addSearch(Map<String, String> data, String sql, List<Object> list, SessionContainer scont) throws
    if (sql.indexOf(DBConst.KEY_WHERE) != -1) {
        String sSearchElements = data.remove(ParamConst.SEARCHFIELDS);
        String [] searchElements = sSearchElements.split(SystemConst.STRING_DELIMITER);
        Map<String, String> searchData = new HashMap<String, String>();
        searchData.put(PMConst.GUI_SEARCHKEY, CastUtil.getString(params, PMConst.GUI_SEARCHKEY, ""));

        List<Object> lParams = new ArrayList<Object>();

        // this replaces {share}-placeholders in Search fields -parameter
        sSearchElements = ModuleHelper.getSharedDatabase(scont, sSearchElements);
        sql = SQLHelper.addFullTextSearch(sSearchElements, sql, lParams, searchData, scont);

        list.addAll(0, lParams);
        lParams.clear();

        String where = SQLHelper.addSQLSearchAndCharacter(scont, searchData, null, searchElements, lParams, "");

        // Add the correct amount of parameters according to the count of WHERE-placeholders
        int count = StringUtil.count(sql, DBConst.KEY_WHERE);
        for (int j = 0; j < count; j++) {
            list.addAll(lParams);
    }

        sql = StringUtil.replace(sql, DBConst.KEY_WHERE, where);

    }
    return sql;
}
```

Koodiesimerkki 5. Portaalihaku-moduulin addSearch()-metodiin tehdyt muutokset (korostettu keltaisella).

Projektin aikana tuotteen kehitysjonoon lisättiin vielä henkilökunnan toiveesta henkilö- ja rooli-suodattimien toteutus. Tuoteomistajan kanssa päätettiin, että Parametrit-välilehdelle olisi hyvä saman tien lisätä tuki personoitaville suodattimille, kuten muissakin listanäkymissä (Kuva 30). Sen jälkeen kun palvelinpuolen koodin muutokset oli tehty, niin nämä muutokset oli helposti toteutettavissa pelkkiä parametreja muuttamalla.

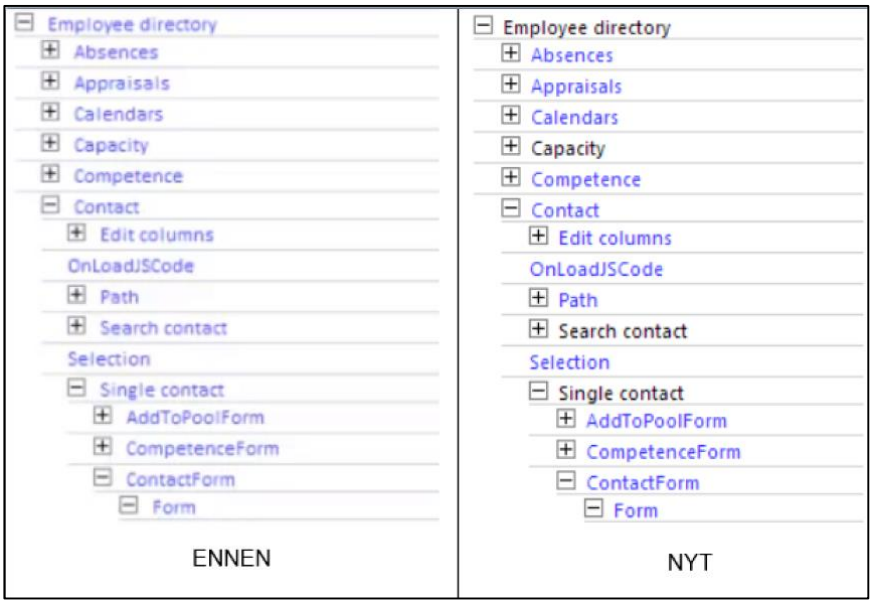


Kuva 30. Personoitavat suodattimet Parametrit-välilehdellä (Personointi-painike korostettu keltaisella).

Alun perin suunniteltu ”Local”-suodatin, jolla voitaisiin suodattaa pelkästään paikalliset tai jaetut parametrit näkyviin, päätettiin lopulta otsikoida havainnollisemmin nimellä ”View by”. Lisäksi päädyttiin siihen, että kyseisessä suodattimessa on vaihtoehtoina vain näyttää joko jaetut ja paikalliset parametrit (”With shared”) tai pelkästään paikalliset parametrit (”Local only”), koska todettiin, ettei ole mielekästä asiakasinstanssin kontekstissa tarkastella pelkästään jaettuja parametreja.

4.4.4 Muut parannukset

Projektin aikana tuotteen kehitysjonoon kerääntyi uusia parannusideoita, kun käytettävyyttä pohdittiin projektitiimin keskinäisissä palavereissa ja muutamat näistä ideoista päätyivät vielä projektissa toteutettavaksi, koska niiden todettiin olevan nopeasti tehtäviä, mutta käytettävyyteen merkittävästi vaikuttavia muutoksia. Merkittävin yksinkertainen muutos oli mahdollisesti ylimääräisten linkkien poistaminen Parametrit-välilehden kategoriapuusta (Kuva 31).



Kuva 31. Ylimääräisten linkkien poistaminen Parametrit-välilehden kategoriapuusta.

Aikaisemmin kategoriapuu näytti jokaisen puussa esiintyvän kategorian linkkinä, vaikkei kyseisen kategorian alla ollut yhtään listattavaa parametria. Oikeaa parametria etsiessään käyttäjä ei voinut välttyä klikkailemasta eri kategorioiden linkkejä kokeillakseen, onko etsittävä parametri kyseisellä tasolla. Ylimääräiset linkit saatiin korjattua melko pienellä vaivalla tekemällä pieniä koodimuutoksia kahteen puun rakentamisen vaiheeseen; TreeData.java -luokkaan ja htmltree.xsl -pohjaan (Koodiesimerkki 6).

TreeData.java:

```
...
TreeNode newval = (TreeNode)tree.get(path);
if (newval == null) {
    newval = new TreeNode(XML_ELEMENT_CATEGORY);
    newval.setAttribute(XML_ATTR_CAPTION, columnname);

    if (j == columnnames.size() - 1) {
        newval.setAttribute(XML_ATTR_LINK, path);
    }
    newval.setAttribute(XML_ATTR_INTERNALID, counter + "");
    if (current.getAttribute(XML_ATTR_INTERNALID) != null) {
        newval.setAttribute(XML_ATTR_INTERNALPARENTID,
        current.getAttribute(XML_ATTR_INTERNALID));
    }
    addNodeData(newval, params);
    current.addChild(newval);
    tree.put(path, newval);
}
...
```

htmltree.xsl:

```
...
<xsl:choose>
    <xsl:when test="@Link">
        <xsl:element name="a">
            <xsl:attribute name="id">
                node<xsl:value-of select="translate(@Link, ' ', '&#x203B;')"/>
            </xsl:attribute>
            <xsl:attribute name="href">#</xsl:attribute>
            <xsl:attribute name="onclick">
                return PLANMILL.ui.tree.cat(this);
            </xsl:attribute>
            <xsl:value-of select="@Caption"/>
        </xsl:element>
    </xsl:when>
    <xsl:otherwise>
            <xsl:value-of select="@Caption"/>
    </xsl:otherwise>
```

Koodiesimerkki 6. Ylimääräisten linkkien parametripuusta poistamista varten tehdyt muutokset.

Toinen parannus, mikä päätettiin toteuttaa projektissa, oli uusi ”Auto scroll” -toiminnallisuus, jolla voidaan sivun latautumisen yhteydessä GET-parametrina annetun id:n perusteella vierittää listanäkymää näyttämään tiettyä entiteettiä. Tämä haluttiin toteuttaa, koska koettiin, ettei esimerkiksi portaalihakutuloksista siirryttäessä Parametrit-välilehdelle ole riittävää pelkästään näyttää oikeaa kategoriaa käyttäjälle, sillä joidenkin kategorioiden alla voi olla satoja parametreja, joiden sekaan etsitty parametri hukkuu. Toteutetussa toiminnallisuudessa lisättiin tuki uudelle GET-parametrille ”focusid”, jonka arvoksi annettiin fokusoitavan entiteetin id.

Kuva 32. Auto scroll -toiminnon seurauksena näytetty ja valittu rivi.

Sivun latautumisen yhteydessä sivu automaattisesti vierittyy animaation kera kyseisen entiteetin kohdalle ja valitsee sen, jolloin rivi tulee myös korostetuksi (Kuva 32). Käyttöönottovaiheessa tätä toiminnallisuutta hyödynnettiin portaalihakutuloksista Parametrit-välilehdelle vievissä linkeissä, sekä parametrin muokkauslomakkeen tallennuspainikkeen yhteydessä, mutta voidaan jatkossa soveltaa missä tahansa listanäkymässä.

bindLoadEvent.xsl:

```
...
PLANMILL.ui.list.scrollToFieldAndSetChecked($.url.param('focusid'));
...
```

list.js:

```
...
scrollToFieldAndSetChecked: function (id) {
    var $field = $('.table tbody tr[data-id="' + id + '"]');
    if($field !== undefined){
        var $body = $('#body');
        $body.animate({scrollTop: $field.offset().top - $body.offset().top + $body.scrollTop()}, 550);
        $field.click();
    };
...
```

Koodiesimerkki 7. Koodia liittyen ”Auto scroll” -toiminnallisuuteen.

”Auto scroll” -toiminnallisuutta varten toteutettiin uusi JavaScript-funktio scrollToFieldAndSetChecked(), jota kutsuttiin sivun latautumisen yhteydessä (Koodiesimerkki 7). Käyttöönottovaiheessa tämä koodi osoittautui kuitenkin vialliseksi, josta kerrotaan enemmän luvussa 4.5.2, s. 51.

4.5 Tuotantoon siirtyminen

4.5.1 Koodinkatselmointi ja manuaalinen testaus

Yrityksen kehitysprosessiin kuuluu itse koodin tuottamisen jälkeen vielä seniorikehittäjän suorittama koodinkatselmointi sekä toimintojen manuaalinen testaus jonkun muun kuin kehittäjän toimesta. Koodinkatselmoinnin tarkoituksena on estää huonon koodin pääty-

mistä tuotantoon ja myös ohjata juniorikehittäjiä noudattamaan yhteisiä ohjelmoimiskäytäntöjä. Projektissa tuotettu koodi palautui koodinkatselmoinnista korjattavaksi muutaman kerran, sillä koodinkatselmoinnin teki eri kerroilla eri henkilö, jolloin asiaan saatiin uusia perspektiivejä. Tässä vaiheessa tehtiin vielä joitain merkittäviäkin muutoksia. Päätettiin esimerkiksi, että alun perin Java-koodin tasolla tehty JDBC-osoitteen muuntaminen ”four part naming” -skeemaa noudattavaksi (Kuva 27), tulisikin tehdä tietokantafunktiossa (Koodiesimerkki 4).

Koodinkatselmoinnin läpäistyä koodi meni manuaaliseen testaukseen. Manuaalinen testaus tehtiin BDD-testiskenaarioiden pohjalta (Luku 4.1.1, s. 22). Manuaalinen testaus on osoittautunut koodikatselmoinnin lisäksi erittäin olennaiseksi osaksi kehitysprosessia, sillä vaikka kehittäjä itse ohjelmoinnin yhteydessä suorittaakin testaamista, niin usein manuaalisen testauksen aikana ilmenee jotain, joka on esimerkiksi päässyt ohjelmoijalta unohtumaan. Projektin muutoksia testattaessa mitään merkittävää ei löytynyt enää manuaalisen testauksen vaiheessa, vaikka yhden kerran testaaja palauttikin koodin korjattavaksi, koska eräs testiskenaario oli vanhentunut, eikä vastannut uusinta määrittelyä.

Onnistuneiden koodikatselmoinnin ja manuaalisen testauksen jälkeen on vielä version julkaisemisprosessiin liittyvät vaiheet, joita ovat kehityshaaran koodin yhdistäminen päähaaraan, versiotestaus, sekä tuotantoon siirtäminen. Kehityshaaran koodin yhdistämisen yhteydessä tehdään vielä niin sanottu yhdistämiskatselmointi (”merge review”), jossa katsotaan läpi ja korjataan mahdolliset konfliktit kahden haaran koodien välillä. Versiotestauksen tarkoituksena on tunnistaa mahdolliset toiminnalliset häiriöt, joita saattaa aiheutua versiossa julkaistavien eri ominaisuuksien yhteisvaikutuksesta. Versiotestauksessa suoritetaan vakiotestisarja kriittisten kohtien testaamiseksi ja kehityshaarakohtaisia testejä. Testauksen jälkeen uusi versio siirretään tuotantopalvelimille version julkaisemisesta vastuullisen henkilön toimesta, joka on usein joku seniorikehittäjistä.

4.5.2 Tuotantoon siirtymisen jälkeen havaitut ongelmat

Kuten työssä aiemmin kävi ilmi, ei projektin aikana ollut varmuutta siitä, miten jaettujen parametrien näyttäminen asiakasinstansseissa käyttäytyisi tuotantoympäristössä, jossa samanaikaisia käyttäjiä on useita ja osassa instansseista asiakaskohtainen tietokanta sijaitsee eri palvelimella kuin jaettu tietokanta. Tuotantoon siirryttäessä jaettujen parametrien näyttämiseen vaadittava käyttöoikeus oli oletuksena pois kaikilta rooleilta, jolloin

ominaisuuden toimivuutta pystyttiin testaamaan käytännössä aluksi parissa instanssissa. Todettiinkin, että juuri instanssit, joiden asiakaskohtainen tietokanta sijaitsee eri palvelimella kuin jaettu tietokanta, osoittautuivat erityisen ongelmallisiksi; jaettujen parametrien listaaminen Parametrit-välilehdellä sekä portaalihaussa aiheutti ajoittain useiden kymmenien sekuntien latausaikoja. Tästä syystä esimerkiksi portaalihaun käyttäminen oli epäkäytännöllisen raskasta. Asiakasinstanssit, joissa asiakaskohtainen sekä jaettu tietokanta sijaitsivat samalla tietokantapalvelimella, suoriutuivat operaatioista huomattavasti nopeammin. Tämä on seurausta lähinnä siitä, että SQL Server pystyy optimoimaan saman tietokantapalvelimen sisällä tehtyjä kyselyjä, mutta optimoimismahdollisuudet menetetään tehtäessä kyselyjä etäpalvelimelle. Vaara oli tiedossa jo etukäteen ja luotettiin siihen, että tietokantakyselyjä optimoimalla asiaa voitaisiin parantaa jälkikäteen. Projektitiimiin kuuluvan seniorikehittäjän asiantuntemuksen avulla jaettuja parametreja lukevia tietokantaoperaatioita saatiin optimoitua siten, että sivun latausajat jäivät pahimmissakin tapauksissa alle kymmeneen sekuntiin ja käytettävyys koettiin riittävän hyväksi kaikissa instansseissa. Hakutermien käyttäminen luonnollisesti tekee kyselyistä raskaampia ja niiden lukumäärä on vähintäänkin suoraan verrannollinen latausaikaan.

Tietokantakyselyjen optimointi oli nopeasti tehty ja muilta pahemmilta ongelmilta tuotantoon siirtymisessä vältyttiin. Yksi vika kuitenkin päätyi tuotantoon rikkoen toisen toiminnallisuuden, joka oli samaan aikaan testikäytössä PlanMill Oy:n omassa operatiivisessa instanssissa. Kyseinen toiminnallisuus oli reaaliaikainen kommentointityökalu, joka katosi käyttöliittymästä, koska tässä projektissa lisätty JavaScript-koodi ”Auto scroll” -ominaisuuteen liittyen (Luku 4.4.4, s. 48) aiheutti suoritusvirheen selaimen JavaScript-moottorissa. Todennäköisesti tällä oli myös muita vaikutuksia käyttöliittymän toimivuuteen, mutta ongelma huomattiin juuri edellä mainitun seikan perusteella. Ongelma saatiin nopeasti korjattua tekemällä pieni korjaus ongelman aiheuttaneeseen funktioon, mutta ominaisuuteen liittyen tehtiin jälkikäteen vielä pari korjausta, kuten se, että kyseinen funktio suoritetaan vain listanäkymäsivuilla.

```
scrollToFieldAndSetChecked: function (id) {
        var $field = $('.table tbody tr[data-id="' + id + '"]');
        if /* ($field !== undefined) */ ($field.length) {
            var $body = $('#body');
            $body.animate({scrollTop: $field.offset().top - $body.c
            $field.click();
        }
    }
```

Koodiesimerkki 8. Vian aiheuttanut kohta (kommentoitu).

Koodissa oleva virhe (Koodiesimerkki 8) oli ehdossa, joka kontrolloi sitä, suoritetaanko sivun vieritys: viallisessa koodissa tämä ehto toteutui aina, josta seurasi, että kutsuttiin olemattoman objektin funktiota. Tämä virhe keskeytti lopun JavaScript-koodin suorittamisen sivulla. Virhe oli luonteeltaan niin näkymätön, ettei sitä huomattu missään kehitys- ja julkaisuprosessin vaiheessa.

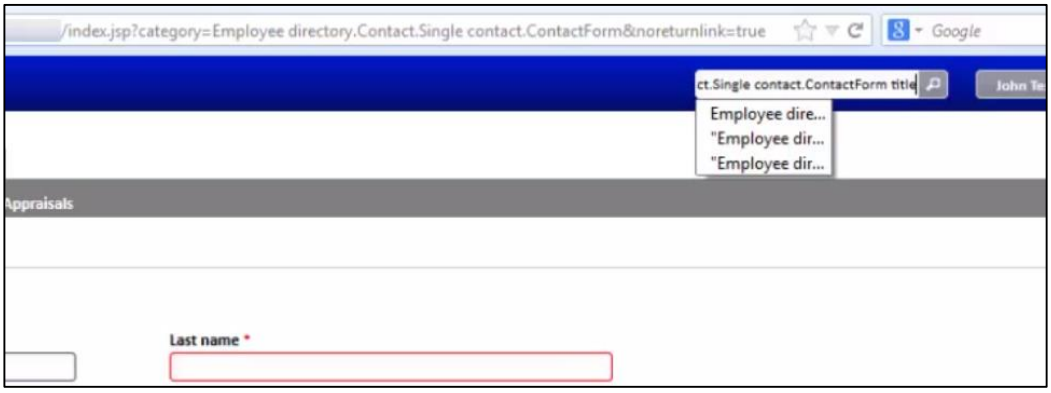
4.6 Projektin jälkeen

PlanMill-henkilökunta on hyvin omaksunut projektissa tehdyt parannukset osaksi jokapäiväisiä prosesseja. Muutaman kuukauden käytössä olon aikana on henkilökunnalta tullut paljon hyvää palautetta ja moni kokee, että parannukset helpottavat huomattavasti työntekoa. Olennaisin parannus on se, että jaetut parametrit ovat tarkasteltavissa ja kopioitavissa paikallisiksi suoraan asiakkaan instanssista, jolloin monta välivaihetta jää pois aiempaan tilanteeseen verrattuna. Myös ylimääräisten linkkien poistaminen parametripuusta (Luku 4.4.4, s. 48) on osoittautunut erittäin hyödylliseksi parannukseksi. Ongelmia ei ole ilmennyt uusien ominaisuuksien myötä.

4.6.1 Tyypillisen räätälöintitoimenpiteen vaiheet työn lopputilanteessa

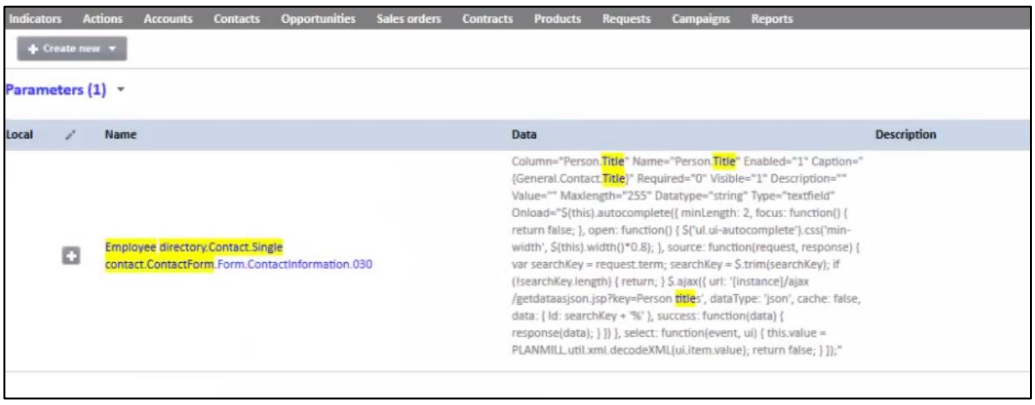
Tässä luvussa on kuvattu, kuinka sama räätälöintitoimenpide kuin luvussa 3.1.2 tapahtuu projektissa toteutettujen parannusten jälkeen.

1. Kirjaudu asiakkaan instanssiin ja mene lomakkeelle, johon uusi kenttä tulee lisätä.
2. Kopioi selaimen osoiterivillä olevan ”category” GET-parametrin arvo ja liitä se portaalihakukenttään. Kirjoita hakukenttään lisäksi muita tarkentavia hakutermejä; esimerkiksi lisättävää kenttää edeltävän kentän otsikko (kuva 33). Suorita sitten haku.



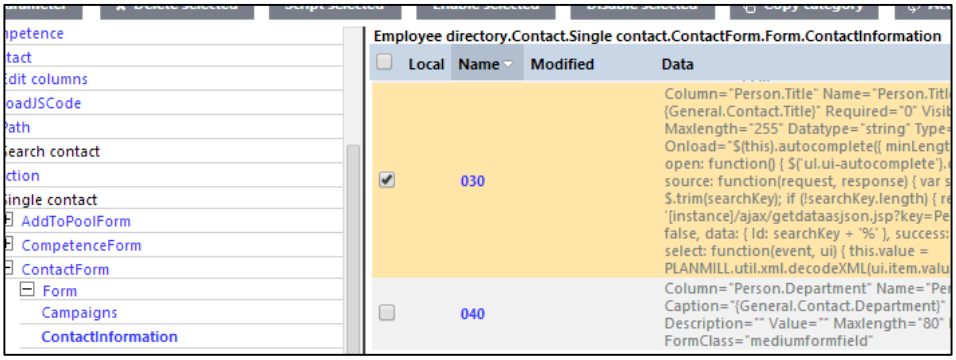
Kuva 33. Portaalihaun käyttäminen parametrin etsimiseen.

3. Selaa hakutuloksia ja etsi sieltä lisättävää kenttää edeltävän kentän parametri (kuva 34). Klikkaa parametrin nimeä päästäksesi Parametrit-välilehdelle.



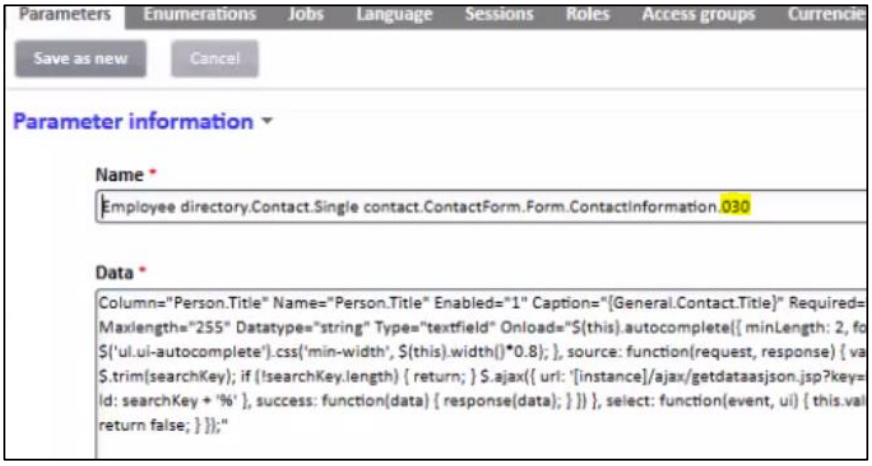
Kuva 34. Parametrihaun tulokset portaalihaussa.

4. Parametrit-välilehti avautuu automaattisesti näyttämään oikean kohdan (kuva 35). Katso mikä on lisättävää kenttää edeltävän kentän parametrin nimi (numero) ja sitä seuraavan kentän parametrin nimi, sillä uusi kenttä halutaan lisätä näiden kahden väliin. Avaa sitten esimerkiksi edeltävän kentän parametri muokkauslomakkeelle.



Kuva 35. Parametrit-välilehti ja Auto scroll -toiminnon seurauksena valittu parametri.

5. Muuta nimen viimeinen pisteellä erotettu osa siten, että uusi nimi sijoittuu numerojärjestyksessä edeltävän ja seuraavan kentän nimien väliin (kuva 36). Täytä sitten datakenttään uutta räätälöityä tekstikenttää varten tarvittavat tiedot. Lopuksi tallenna parametri uutena (”Save as new”).



Kuva 36. Parametrin muokkauslomake, jossa parametrin nimiosa korostettuna keltaisella.

6. Siirry lomakkeelle tarkistamaan, että lisätty kenttä näkyy ja toimii oikein. Yleensä tämän jälkeen sama operaatio tehdään kentän lisäämiseksi vähintään myös yhteenvetosivulle.

Läpikäyty skenaario kuvaa vain yhden tilanteen, jossa projektissa toteutetut ominaisuudet helpottavat. Jos jokainen edellä mainittu välivaihe purettaisiin askeleisiin, niin projektissa tehtyjen parannusten vaikutus korostuisi entisestään. Toiseksi, käytetty esimerkki ei anna osviittaa siitä, kuinka suuri ero toimenpiteen suorittamisessa on ajallisesti alku- ja lopputilanteen välillä. Kolmanneksi, todellisuudessa uudella työntekijällä kului alkutilanteessa aikaa myös muun muassa parametripuun availuun ja oikean parametrin löytämiseen, sillä aina parametrikategorioiden nimet eivät ole selkokieliset. Uudet hakutyökalut auttavat tähänkin ongelmaan.

4.6.2 Tulevaisuudessa toteutettavia asioita

Uusia kehitysideoita on tullut paljon vielä projektin jälkeenkin ja niitä on listattu tuotteen kehitysjonoon. Kehitysjonossa on projektin päätyttyä useita sekä pienempiä korjaustoimenpiteitä että suurempia kehitysideoita:

- Mahdollisuus parametri- ja roolikohtaisesti määrittää parametrien näkyvyys Parametrit-välilehdellä asiakasinstansseissa. Tämä mahdollistaisi sen, että tulevaisuudessa voitaisiin antaa asiakkaan itse muokata parametreja yksinkertaisten räätälöintien tekemiseksi riskeeraamatta arkaluontoisia parametreja.
- Parametrin muokkauslomakkeen avaaminen peiteikkunassa (*overlay window*) AJAX-tekniikkaa hyödyntäen koko sivun lataamisen sijaan portaalihakutulossivulla sekä Parametrit-välilehdellä ja myöhemmin suoraan lomakkeella. Tällä parannettaisiin käytettävyyttä sekä kevennettäisiin palvelimiin kohdistuvaa taakkaa.
- Palauta-toiminto, jolla voidaan nopeasti palauttaa parametrista aikaisempi versio historiataulua hyödyntäen. Yksinkertaisimmillaan se voisi olla painike, jolla voidaan palauttaa edellinen versio parametrista, mutta mahdollisuus tarkastella ja palauttaa mikä tahansa aikaisempi versio, olisi vielä parempi.
- Passiivisten parametrien näyttäminen omalla erottuvalla värillä Parametrit-välilehdellä, jotta paremmin huomaa, mikä parametri on aktiivinen ja mikä ei.
- Enemmän näytettäviä kenttiä portaalihaun parametrituloksiin; mieluusti kaikki samat kuin Parametrit-välilehdellä.
- Enemmän suodattimia Parametrit-välilehdelle ja parannetut valintalistat niihin, jottei niissä näytettäisi valinta-arvoja, joilla ei löydy yhtään listattavaa parametria.
- Tietokantakyselyiden ja palvelinpuolen koodin optimointia.

Edellä mainittujen lisäksi kehitysjonossa on luonnollisesti myös kaikki projektissa kesken jääneet ominaisuudet, kuten muokkauspainikkeiden lisääminen lomakkeille sekä muut luvussa 3.3 esitetyt parannusehdotukset.

5 Yhteenveto

Projektin lähtötilanteessa PlanMill-järjestelmän konfigurointiin ja räätälöintiin käytettävien parametrien käsiteltävyydessä koettiin puutteita. Projektin tarkoituksena oli selvittää parametrien käsittelyprosessin ongelmakohtia ja kehittää uusia menetelmiä prosessien parantamiseksi. Yrityksessä oli jo entuudestaan ollut esillä kehitysideoita, jotka otettiin huomioon projektin toteutuksessa. Lisäksi selvitettiin, minkä tyyppisiä konfigurointi- ja räätälöintitoimenpiteitä asiakkaat eniten tilaavat, jotta pystyttiin kohdistamaan toteutettavat parannukset oikeisiin kohteisiin.

Projektin toteutus aloitettiin suunnittelemalla, kerätyn datan pohjalta, vaihtoehtoisia käyttöliittymähahmotelmia eli prototyyppejä. Projektitiimin kanssa päädyttiin toteutukseen, joka koostuu useammista pienistä ja toisistaan riippumattomista parannuksista. Tähän päädyttiin, koska järjestelmä oli opinnäytetyön tekijälle varsin uusi, ja mikäli aika loppuisi projektissa kesken, pystyttäisiin ainakin osa parannuksista julkaisemaan.

Toteutusvaiheen aikana tuotteen kehitysjono päivittyi lähinnä, koska osa alun perin suunnitelluista parannuksista huomattiin puutteellisiksi ja toisaalta projektin aikana syntyi uusia parempia kehitysideoita. Projektin aikana saatiin toteutettua useita parannuksia parametrien käsittelyprosesseihin. Tärkeimpänä parannuksena toteutettiin mahdollisuus niin sanottujen jaettujen parametrien tarkastelemiseen ja paikalliseksi kopioimiseen asiakkaiden PlanMill-instansseissa. Muita toteutettuja parannuksia olivat esimerkiksi uudet parametrien hakumahdollisuudet, personoitavat suodattimet Parametrit-välilehdelle sekä ylimääräisten linkkien poistaminen Parametrit-välilehden parametripuussa.

Ominaisuudet julkaistiin tuotantoon ja suuremmilta ongelmilta vältyttiin, vaikka jonkin verran koodin optimointia jouduttiin tekemään velä jälkikäteen; se oli kuitenkin tiedossa ja siihen oli varauduttu. Toteutetut parannukset saivat hyvän vastaanoton henkilökunnalta ja uudet ominaisuudet on omaksuttu kiinteäksi osaksi parametrien käsittelyprosesseja.

Projektin aikana syntyi lisäksi monia hyviä uusia parannusideoita toteutettujen parannusten lisäksi, jotka lisättiin tuotteen kehitysjonoon ja joita voidaan toteuttaa tulevissa projekteissa.

Lähteet

1. Sikha Bagui, Richard Earp 2012. Database Design Using Entity-Relationship Diagrams, Second Edition. Taylor & Francis Group.

2. Antero Järvi, Jussi Karttunen, Tuomas Mäkilä & Jouni Ipatti 2011. Saas-käsikirja 2011. Turun yliopisto.

3. Wonjae Lee & Min Choi. A Multi-tenant Web Application Framework for SaaS 2012. IEEE Xplore.

4. Merkintäkieli. Verkkosivu. Wikipedia. <fi.wikipedia.org/wiki/Merkintäkieli>. 15.3.2013. Luettu 21.9.2013.

5. Parameters (computer programming). Verkkosivu. Wikipedia. <en.wikipedia.org/wiki/Parameter_(computer_programming)>. 16.9.2013. Luettu 28.9.2013.

6. Wei Sun, Xin Zhang, Chang Jie Guo, Pei Sun & Hui Su. Software as a Service: Configuration and Customization Perspectives 2008. IEEE Xplore.

7. PlanMill-järjestelmän asiakkaat. Verkkosivu. PlanMill Oy. <planmill.fi/Customers>. Luettu 22.9.2013.

8. Scrum. Verkkosivu. Wikipedia. <fi.wikipedia.org/wiki/Scrum>. 16.4.2013. Luettu 29.9.2013.

9. Full-Text Search (SQL Server). Verkkosivu. http://technet.microsoft.com/en-us/library/ms142571.aspx#like. Luettu 17.12.2013.

10. Ohjelmiston versionhallinta. Verkkosivu. Wikipedia. <http://fi.wikipedia.org/wiki/Ohjelmiston_versiohallinta>. 11.3.2013. Luettu 10.11.2013.

11. PlanMill Oy:n historia. Verkkosivu. PlanMill Oy. <planmill.fi/About Us/Company>. Luettu 15.3.2014.