Max Mikkola

Kommentointimoduulin toteutus PlanMill-toiminnanohjausjärjestelmään

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

5.5.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Max Mikkola Kommentointimoduulin toteutus PlanMill-toiminnanohjausjärjestelmään 31 sivua 5.5.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander Senor consultant/Manager Marjukka Niinioja

Työn tarkoituksena oli luoda PlanMill-toiminnanohjausjärjestelmään kommentointitietoja hoitava moduuli. Aikaisemmin järjestelmästä ei löytynyt keskitettyä tapaa kommentoida järjestelmän tietueita. Uuden moduulin voi kiinnittää kaikkiin järjestelmän muihin moduuleihin, ja sen avulla kaikkien järjestelmän tietueiden kommentointi on helppoa.

Työssä pilotoitiin Backbone.js:ää, yhtä muutamasta JavaScript-kirjastosta, jotka mahdollistavat MVC-tyylisten ohjelmien rakentamisen selaimeen. Myös AngularJS:ää harkittiin pilotoitavaksi projektissa ja sitä testattiin alustavasti.

Uudet toiminnallisuudet pyrkivät parantamaan PlanMillin sekä asiakkaiden sisäistä kommunikointia PlanMillin avulla.

Projekti alkoi laaja-alaisena, ja se sisälsi myös ulkoisen kommunikaation. Kaksi viikkoa projektin alun jälkeen kokonaisuutta pienennettiin sisältämään vain järjestelmän sisäiset kommunikaatiot.

Kommentointimoduuli toteutettiin alun myöhästymisestä huolimatta onnistuneesti ajallaan.

Avainsanat	PlanMill, PSA, Backbone

Author(s) Title Number of Pages Date	Max Mikkola Implementation of a New Commenting Module into PlanMill PSA-system 31 pages 5.5.2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Marjukka Niinioja, Senior consultant/Manager

The aim of this study was to implement a new communication module into the PlanMill PSA-System. Before, there was a lack of a centralized way of commenting on many objects within the system. The new module can be attached to any other module in the system, and it allows easy commenting for all objects in the system.

The study also piloted Backbone.js, one of a few JavaScript libraries that allow for construction of MVC-style applications in the front end. AngularJS was also considered for use in the project, and it was preliminarily tested.

These new functionalities aim to help internal communications within PlanMill and customers alike with the PlanMill PSA-System.

The project itself was at first open and ambitious, including outbound communications within the scope of the study. The scope was narrowed down to only include internal communications two weeks after the project started.

The new module was implemented successfully within the allotted timeframe.

Keywords	Planmill,PSA,Backbone

Sisällys

Lyhenteet

AJAX	Asynchronous JavaScript and XML. Kokoelma teknologioita, joiden avulla voidaan kommunikoida palvelimen kanssa tausta-ajona ilman, että verkkosivua tarvitsee ladata kokonaan uudestaan.
API	Application Programming Interface. Ohjelmointirajapinta.
CRM	Customer Relationship Management. Asiakkuudenhallinta.
DOM	Document Object Model. Dokumenttiobjektimalli.
DHTML	Dynamic HyperText Markup Language. Yleinen termi tekniikoille, joiden avulla voidaan rakentaa rikkaita verkkosovelluksia.
ERP	Enterprise Resource Planning. Perinteinen yritysten toiminnanohjausjärjestelmä.
HRM	Human Resource Management. Henkilöstöhallinta.
HTML	HyperText Markup Language. Verkkosivujen rakennetta kuvaava standardisoitu kuvauskieli.
JSON	JavaScript Object Notation. Kevyt, helposti luettava tekstipohjainen tiedonsiirtoformaatti.
JSP	JavaServer Pages. Javaan perustuva menetelmä WWW-sivujen luontiin.
PSA	Professional Service Automation. Toiminnanohjausjärjestelmän variaatio palveluyrityksille.
SaaS	Software As A Service. Ohjelmisto palveluna.
SPA	Single Page Application. Yhden sivun ohjelmisto.
XML	Extensible Markup Language. Standardoitu merkintäkieli.

XSL	Extensible Stylesheet Language. Tyylimäärittely XML-dokumentille.

1 Johdanto

Tässä työssä suunniteltiin ja toteutettiin uusi moduuli PlanMill-toiminnanohjausjärjestelmään. Työ tehtiin PlanMill Oy:lle.

Nykypäivän palvelukeskeisillä työmarkkinoilla palveluprosessien toiminnan nopeus ja tehokkuus on todella tärkeää. Palveluyrityksen sisäisen kommunikaation ja tiedon kulun täytyy olla selkeää ja nopeaa. Moniosaisessa prosessissa yrityksen henkilöstön ja asiakkaiden on hyvä olla tietoisia siitä, mitä tapahtuu.

Projektissa luotiin uusi moduuli olemassa olevaan järjestelmään, joka nopeutti ja selkeytti sisäisen informaation kulkua käyttäjien välillä. Uusi moduuli mahdollistaa järjestelmän eri objektien kommentoinnin ja ilmoittaa uusista kommenteista sisäisesti muille käyttäjille.

Projektissa käytettiin myös järjestelmälle uutta JavaScript-kirjastoa, Backbone.js:ää. Backbonen avulla luotiin uusi dynaamisempi käyttökokemus kommenttien kirjoittamisessa sekä lukemisessa. Muusta järjestelmästä eroten kommenttien kirjoitaminen, tallentaminen eikä uusista kommenteista tiedotuksen saaminen vaadi uutta sivunlatausta.

Työ jakautui neljään osaan: käyttötarpeiden kartoitukseen, uuden teknologian tutkimukseen, käyttöliittymäsuunnitteluun sekä tekniseen toteutukseen. Projekti alkoi käyttötarpeiden kartoituksella, joka hoidettiin avoimella sisäisellä kyselyllä. Kartoituksen avulla hahmoteltiin järjestelmän parannustarpeita sekä projektin laajuutta. Kartoituksen ja projektin laajuusmäärittelyn jälkeen tarvitsi tutkia ja arvioida uusia teknologioita sekä valita uusi teknologia projektin toteutusta varten. Käyttöliittymäsuunnittelu ja tekninen toteutus toteutettiin samanaikaisesti. Käyttöliittymän suunnitteluun ja käytettävyyden hiomiseen käytettiin aikaa projektin osa-alueista eniten.

2 PlanMill Oy

PlanMill Oy on pk-yritys, joka on erikoistunut SaaS (Software as a service, ohjelmisto palveluna) -toiminnanohjaussovelluksen toimittamiseen. SaaS on helpoin toteuttaa

pilvipalveluna, jossa palvelun käyttäjät ottavat yhteyttä palveluun verkon yli käyttäen asiakasohjelmistoa. PlanMillin tapauksessa asiakasohjelmistona toimii käyttäjän oma verkkoselain. PlanMillin toiminnanohjausjärjestelmä on saanut alkunsa Nokian projektinhallintatyökalusta. Omistussuhteiden muutoksien yhteydessä projektinhallintatyökalun kehitysryhmä yhtiöityi omaksi yhtiökseen vuonna 2001. Nykyiseen omistukseen yritys siirtyi vuonna 2006.

Pilvipalveluna PlanMill-sovellusta voidaan käyttää mistä vain. PlanMill-sovelluksella on yli 10 000 käyttäjää päivittäin yli 25 maassa. PlanMill tarjoaa myös on-premise-ratkaisua, jolloin sovellus asennetaan asiakkaan ylläpitämään tuotantoympäristöön. Suurin osa asiakkuuksista koskee kuitenkin pilvipalvelua, jolloin myös sovelluspäivitykset tulevat asiakkaille automaattisesti. PlanMill-sovellus sisältää useita toimintokokonaisuuksia, joita voidaan aktivoida asiakkaille yhdessä tai erikseen. Näihin kuuluvat CRM (asiakkuudenhallinta), HRM (henkilöstöhallinta), PM (projektinhallinta), tuotteidenhallinta, tuntikirjausten tallennus ja seuranta sekä automoitu laskutus. Automoitu laskutus on keskeisessä osassa sovelluksen toimintaa. Laskueriä luodaan automaattisesti muun muassa myyntitilauksilta sekä projekteille kirjatuilta työtunneilta.

2.1 Toiminnanohjausjärjestelmä

Toiminnanohjausjärjestelmä, yleisesti lyhennettynä ERP, joka tulee englanninkielen sanoista Enterprise resource planning, on yrityksen ohjaukseen ja automaatioon käytetty järjestelmä. ERP:n yleinen tarkoitus on optimoida yrityksen toimintaa automoimalla useita toiminnallisuuksia kuten esimerkiksi laskutusta. Yleisiä ERP:n tarjoamia ominaisuuksia ovat

- CRM eli asiakkuuksienhallinta
- HRM eli henkilöstöhallinta
- kirjanpito
- laskutus
- projektinhallinta

- raportointi
- reskontra
- tuntikirjausten seuranta
- tuotannonohjaus
- varaston sekä tuotteiden hallinta.

ERP on yleisesti todella laaja kokonaisuus, joka sisältää yrityksen kaiken toiminnan. (Toiminnanohjausjärjestelmä, 2013.)

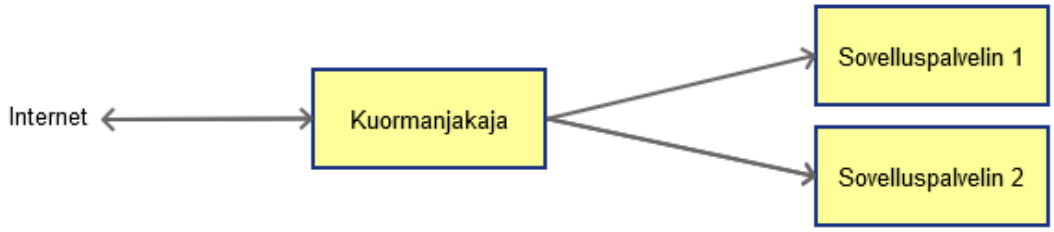
Toiminnanohjausjärjestelmistä on myös erikoistuneita variantteja, kuten palvelu- sekä asiantuntijayrityksien tarpeisiin suuntautunut PSA. PSA tulee englanninkielisistä sanoista Professional Service Automation. PSA-järjestelmät tarjoavat parempia ratkaisuja aloille joissa myytävä tuote on palvelumuotoista asiantuntemustyötä. Mahdollisiä käyttäjiä ovat esimerkiksi lakitoimistot sekä insinööritoimistot. PSA-järjestelmät rakentuvat yleensä suurilta osin samoista komponenteista kuin ERP:t, mutta sisältävät komponenteista räätälöityjä variantteja, kuten asiantuntemustyön laskuttamisen tuntien perusteella. (Professional services automation, 2014.)

2.2 PlanMill Cloud

PlanMill-toiminnanohjausjärjestelmä on PSA-variantti, joka sisältää myös yleisimmät normaalit ERP-toiminnot. Toiminnanohjausjärjestelmä toimii nimellä PlanMill Cloud. Nimen mukaisesti järjestelmää toimitetaan pilvipalveluna, eli asiakkaat käyttävät järjestelmää verkon yli verkkoselainten avulla. Jokaiselle PlanMill Cloud -asiakkaalle on oma sovellusinstanssi, johon on yksiselitteinen URL-osoite (Uniform resource locator).

Pilvipalvelun suuri etu on päivityksen helppoudessa verrattuna perinteiseen paikallisille tietokoneille asennettavaan järjestelmään. Asiakkaiden ei tarvitse asentaa tietokoneille mitään verkkoselaimen lisäksi, ja sovelluspäivitykset astuvat voimaan automaattisesti ilman lisäasennuksia.

Jatkuvan toimivuuden takaamiseksi tuotantoympäristö on klusteroitu kahdelle sovelluspalvelimelle.

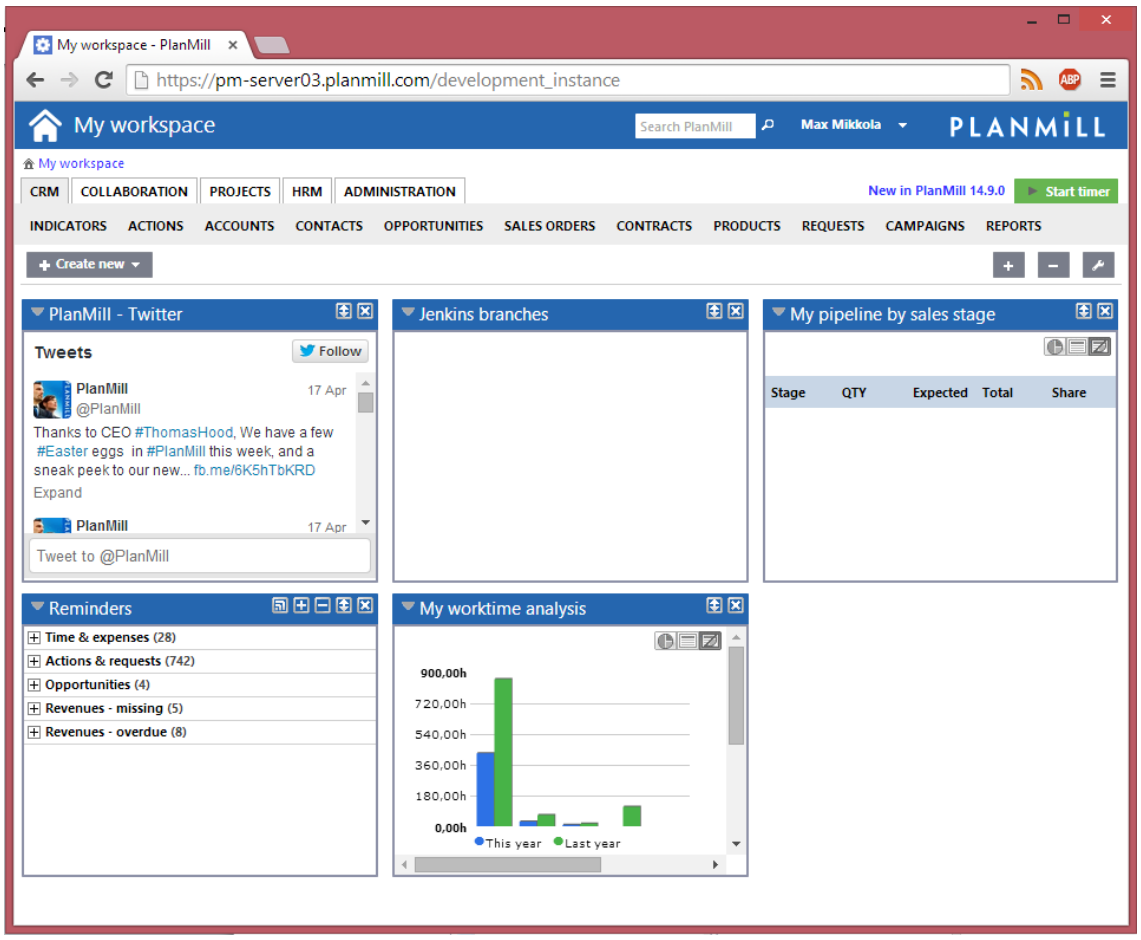


Kuva 1. Yksinkertaistettu esitys sisääntulevan liikenteen kuormanjaosta

Kuvassa 1 kuormanjakaja jakaa sisääntulevan liikenteen kahden sovelluspalvelimien välillä. Sovelluspalvelimia voi lisätä tarpeen vaatiessa lisää. Liikenne ohjataan vähiten kuormittuneelle palvelimelle, joten vasteajat ovat yleisesti optimaalisia.

Klusteroidusta ympäristöstä on myös paljon etua sovelluspäivityksissä. Palvelimet päivitetään yksi kerrallaan, ja päivityksen aikana käyttäjät ohjataan toiselle palvelimelle. Järjestelmää ei siis tarvitse ajaa alas päivityksen yhteydessä.

PlanMill Cloud tarjoaa asiakasyrityksilleen laajan valikoiman ERP:n ja PSA:n toiminnallisuuksia. Kaikki toiminnallisuudet ovat aktivoitavissa erikseen, joten asiakkaille voidaan räätälöidä sopiva kokonaisuus.



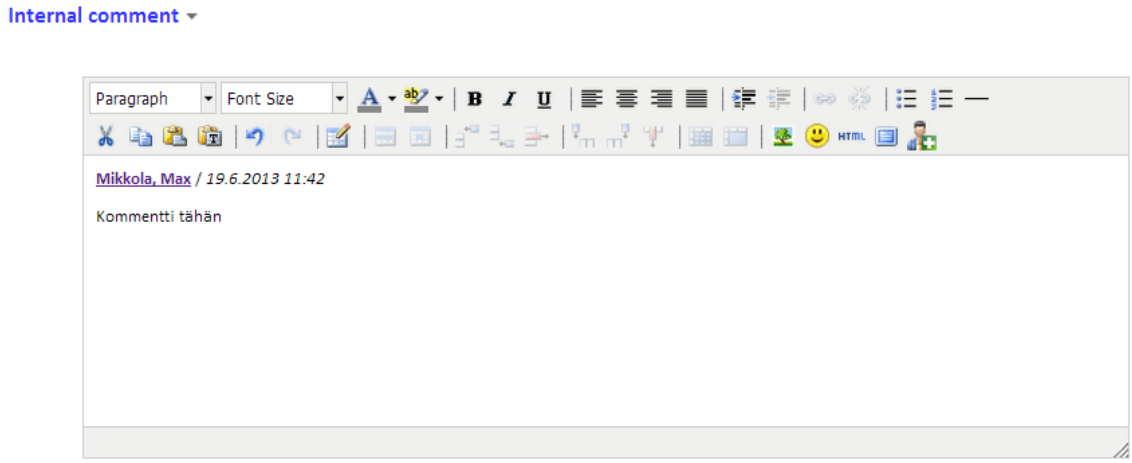
Kuva 2. PlanMill-toiminnanohjausjärjestelmän etusivu.

Kuvassa 2 nähdään PlanMill Cloudin etusivu. Kuva on otettu tehokäyttäjän käyttöoikeuksilla, joten kaikki järjestelmän toiminnot ovat käytettävissä.

3 Projektin tavoitteet

Planmill Cloudissa on useita moduuleita, joiden avulla voi luoda objekteja järjestelmään. Järjestelmän käytön aikana nämä objektit muuttuvat usein, ja muutoksista tallentuvat historiatiedot tietokantaan. Historiatietoihin tallentuu objektin tila ennen ylikirjoittamista, jolloin tietokantaan jää jälki kaikista tehdyistä muutoksista.

Täydellisiä muutostietoja ei voi kuitenkaan nähdä käyttöliittymästä. Tämä on ongelmallista tilanteissa, joissa objektiin liittyvä kommentointi sekä keskustelu tapahtuvat yhden lomakekentän avulla.



Kuva 3. Palvelupyyntö-moduulin sisäistä kommentointia varten varattu lomakekenttä

Kuvan 3 lomakekenttää voivat muokata kaikki henkilöt, joilla on oikeus muokata palvelupyyntöjä. Kuka tahansa voi esimerkiksi tyhjentää koko kentän tai muokata toisten kirjoituksia. Näissä tilanteissa peruskäyttäjällä ei ole mahdollisuutta nähdä muutoksessa mahdollisesti poistuneita tai muuttuneita kommenttitietoja.

Yksilöllisten kommentointitietojen säilyvyyden takaamiseksi järjestelmän kommenteille tarvittiin yksilöivä tallennustapa. Järjestelmään tarvitsi suunnitella uusi kommentteja käsittelevä moduuli, jonka pystyisi liittämään järjestelmän olemassaoleviin moduuleihin. Moduuli saisi hoitaakseen kommenttien yksilöllisen tallentamisen ja esittämisen kaikkialla järjestelmässä.

Uuden moduulin suunnittelussa tahdottiin nostaa esiin kommenttitietojen selkeys, saatavuus ja eheys. Projektille luotiin tarpeiden perusteella hyväksymiskriteerit:

1. Kommentointimoduulin tulee tukea kaikkia järjestelmän muita moduuleja, sekä tulevaisuudessa tuotettavia moduuleja.

2. Järjestelmään täytyy voida tallentaa yhtä tietuetta kohden monta kommenttia.

3. Kommentointiosion täytyy päivittyä dynaamisesti ilman uutta sivunlatausta.

4. Kommentointiosiosta täytyy voida tallentaa tietueita ilman uutta sivunlatausta.

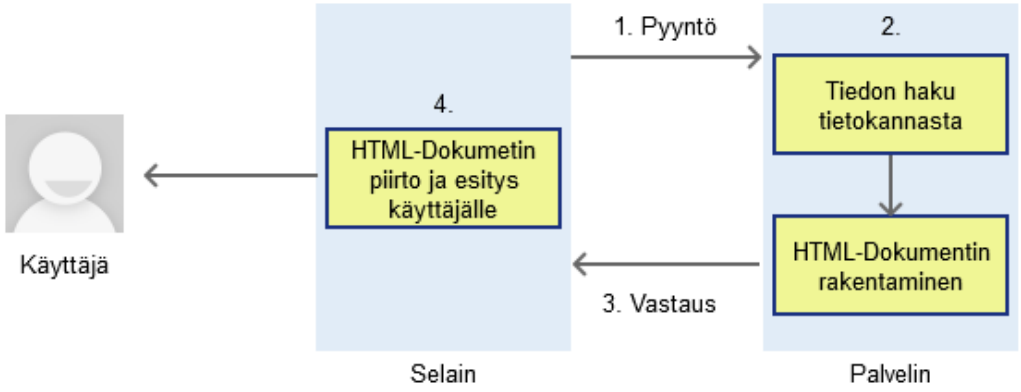
5. Kommentointiosion täytyy olla lokalisoitu.

6. Järjestelmän täytyy pystyä ilmoittamaan käyttäjälle uusista kommenteista.

Järjestelmä ei vielä tukenut tietojen dynaamista lataamista. Kolmannen sekä neljännen hyväksymiskriteerin saavuttamiseksi tarvittiin apua järjestelmälle uudelta JavaScript-kirjastolta. Uutta kirjastoa tarvittiin uusien dynaamisesti ladattujen kommenttien jäsennykseen ja hallinnointiin. Ongelman ratkaisemiseen oli saatavilla useita kirjastoja. Kaikki kirjastot olivat pääosin tarkoitettuja yhden verkkosivun applikaatioiden luomiseen.

4 Single Page Application

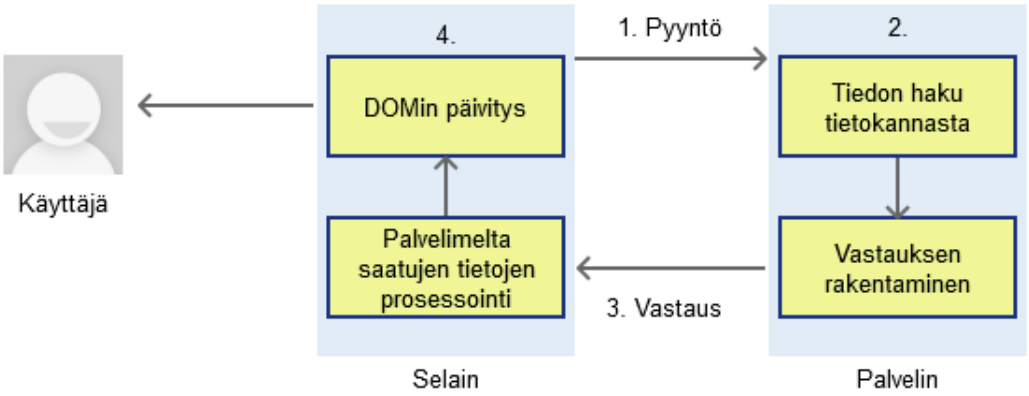
Single Page Application (SPA), eli yhden verkkosivun applikaatio on selaimeen ladattava applikaatio, joka ei ensimmäisen latauksen jälkeen tarvitse uutta sivunlatausta. Yhden verkkosivun applikaatiota on aikaisemmin toteutettu käyttäen muun muassa Flashia tai Javaa, mutta JavaScript on nykyään tehokkain natiivinen toteustapa. (Single Page Web Applications, 2013).



Kuva 4. Perinteinen sivunlataus

Kuva 4 esittää perinteistä sivunlatausprosessia, jossa asiakassovellus pyytää palvelimelta kokonaista HTML-dokumenttia. Palvelin rakentaa pyynnössä saatujen parametrien mukaisen HTML-dokumentin ja palauttaa sen vastauksena pyyntöön. Asiakassovellus rakentaa dokumentin pohjalta sivun ja esittää sen käyttäjälle. Kun käyttäjä navigoi uudelle sivulle, sama prosessi toistuu. Sivun rakentaminen kokonaan uudelleen on suurissa määrissä palvelimelle työlästä, ja asiakassovellukselle hidasta.

Yhden verkkosivun applikaatiossa kokonainen dokumentti ladataan vain sivustolle navigoitaessa. Ensimmäinen lataus on pidempi, sillä sovelluslogiikka ladataan kokonaisuudessaan selaimen muistiin. Myöhemmin sivua vaihdettaessa palvelimeen ei välttämättä tarvita palvelinta ollenkaan.



Kuva 5. DHTML yhden sivun applikaatiossa

Kuvassa 5 SPA päivittää esitettäviä tietoja pyytämällä palvelimelta uusia tietoja tai tietopaketteja Ajaxilla.

Ajax (Asynchronous JavaScript and XML) on kokonaisuus teknologioita, joiden avulla selaimesta voi kutsua palvelinta taustaprosessina (AJAX, 2005). Alunperin Ajaxilla pyydettiin palvelimilta XML-muotoista tietoa, mutta nykyään tiedot saa myös JSON-muodossa. JSON (JavaScript Object Notation) on kevyt tekstipohjainen tiedonsiirtoformaatti, joka on samanaikaisesti helposti luettava ihmisille sekä tietokoneille. (JSON, 2013.)

Palvelin palauttaa selaimen pyytämät tiedot XML- tai JSON-formaatissa. Applikaatio muokkaa sivua tietojen avulla lataamatta sitä kokonaan uudestaan. Sivua muokataan manipuloimalla dokumenttiobjektimallia.

Dokumenttiobjektimalli (DOM) on ohjelmointirajapinta, joka sallii dynaamisen pääsyn HTML-dokumentin rakenteeseen ja tyyliin. (Dokumenttiobjektimalli, 2009.) DOM:n avulla HTML-dokumenttia voi manipuloida lennosta, eikä sivua tarvitse ladata kokonaan uudestaan.

SPA:ta käytettäessä palvelimen kuormitus vähenee, ja käyttäjän käyttökokemus paranee. SPA:n seurauksena selaimen muistiin tallennetaan paljon enemmän tietoja kuin ennen. Käytettävien tietojen jäsennys on tärkeä askel järkevään kehitystyöhön. Jäsennykseen sekä tietojen esitykseen on saatavilla useita JavaScript-kirjastoja. Kirjastot seuraavat yleisesti joko MVC- tai MV*-suunnittelumalleja.

Model-View-Controller, eli Malli-Näkymä-Ohjain on sovellusarkkitehtoninen suunnittelumalli, joka erottelee käyttöliittymän siinä esitettävästä tiedosta. Malli kuvastaa yhtä osaa sovelluksen tiedoista sekä siihen liittyvää logiikkaa. Näkymä sisältää käyttäjälle esitettävän ulkoasun, siis käyttöliittymän. Verkkosovellusten kehityksessä näkymää vastaa HTML-dokumentti. Ohjain siltaa nämä kaksi osaa yhteen, toimien välittäjänä käyttöliittymästä saatavien käskyjen ja malleilta tulevien tietojen välillä. (MVC, 2009.)

MV* on MVC:stä kehittynyt suunnittelumalli, jossa ohjain ei ole pakollinen. Ohjaimen puuttuessa näkymä voi olla suoraan yhteydessä malleihin, jolloin ohjaimen toiminnallisuus jakautuu näkymän ja mallin kesken. (Developing Backbone.js Applications, 2013.)

SPA:n sivunsisäisiin sivuihin tai resursseihin ulkoa linkittäminen suoraan URL-osoitteessa on ongelmallista ilman apuvälineitä. Sovellusta avattaessa selaimessa palvelin lähettää aina saman HTML-dokumentin, mutta sivun täytyisi silti avautua oikealle alisivulle URL-osoitteessa sijaitsevan polun mukaisesti. Kaikissa SPA-kirjastoissa on jonkin tasoinen sisäinen reititin.

Projektissa katselmoitavia kirjastoja oli kaksi: AngularJS sekä Backbone.js. Kyseiset kaksi kirjastoa valittiin testattaviksi, koska yrityksen työntekijöillä oli kirjastoista kokemusta työprojektien ulkopuolelta.

4.1 AngularJS

AngularJS on Googlen kehittämä JavaScript-kirjasto, jonka päämääränä ovat laajennettavuuden sekä testattavuuden helpottaminen yhden verkkosivun ohjelmistoissa. Tämän saavuttamiseksi AngularJS pilkkoo toiminnallisuudet järkeviksi osiksi ohjainluokkiin. (AngularJS, 2014.)

Angularissa käyttöliittymäelementtien manipulointiin käytetään ohjainluokkia, jotka sidotaan elementteihin HTML-dokumentin päälle rakennetulla Angularille ominaisella syntaksilla. HTML-elementin tai sen alielementtien sisään sijoitetut muuttujat liitetään automaattisesti samannimisiin muuttujiin ohjainluokassa. Käyttöliittymässä esitettävän tiedon ja JavaScript-muuttujan sitomista toisiinsa kutsutaan kaksisuuntaiseksi sidonnaksi. Kaksisuuntaisessa sidonnassa molempien puolien arvo on aina sama. Jos käyttöliittymäelementin arvoa muutetaan, JavaScript-muuttujan arvo muuttuu samaksi automaattisesti. Sama toimii myös toisinpäin.

```
<html ng-app> <!-- 1 -->

    <script src="./angular.js"></script>
    <script src="./ElementController.js"></script>

    <body ng-controller="ElementController">    <!-- 2 -->
        <button ng-click="Button()">            <!-- 3 -->
            {{variableText}}                    <!-- 4 -->
        </button>
    </body>
</html>
```

Esimerkkikoodi 1. Angularin merkinnät HTML-dokumentissa

Esimerkkikoodissa 1 HTML-dokumentin elementteihin on lisätty ominaisuuksia, joiden avulla AngularJS hallinnoi sivua. Esimerkkiin on numeroitu 4 huomioitavaa kohtaa:

1. Merkintä ng-app määrittää osan sivustosta, jota AngularJS hallinnoi. Esimerkissä AngularJS hallinnoi koko sivua.
2. Merkinnällä ng-controller annetaan elementille (ja mahdollisille alielementeille) ohjainluokka, jonka muuttujien ja funktioiden avulla käyttöliittymä toimii.
3. ng-click sitoo painallustapahtuman tiettyyn ohjainluokan funktioon. AngularJS:ssä on myös merkinnät muille yleisille tapahtumille.
4. Kahdennettujen aaltosulkeiden sisälle voidaan lisätä ohjainluokkien muuttujia ja muuta logiikkaa. Esimerkissä aaltosulkeiden sisälle on asetettu muuttujanimi, joka on sidottu ohjainluokan muuttujaan.

Osa logiikasta, kuten tapahtumakäsittelijöiden kiinnitys, on nyt HTML-dokumentissa, jolloin itse ohjainluokan JavaScript-koodista tulee todella yksinkertaista:

```
function ElementController($scope){
    $scope.variableText = "Teksti.";

    $scope.Button = function(){
        $scope.variableText = "Vaihtunut teksti.";
    }
}
```

Esimerkkikoodi 2. Ohjainluokka ElementController.js

Esimerkin 2 ohjainluokka sisältää ainoastaan ohjaimen toiminnalle loogisesti kuuluvaa toiminnallisuutta. Esimerkiksi normaalisti nappuloihin liitettävät tapahtumakuuntelijat lisäävät koodin määrää ja vähentävät luettavuutta. Kaksisuuntaisen sidonnan ansiosta käyttöliittymässä esitetteviä tietoja ei myöskään tarvitse erikseen asettaa JavaScript-muuttujan muututtua. AngularJS piilottaa loogisesti ohjainluokkiin kuulumattoman koodin kehittäjältä, joten ohjainluokat ovat helpompilukuisia ja helpommin testattavissa.

$scope on Angularin ohjainluokalle antama tilaobjekti, jonka avulla sidottuja HTML-elementtejä voi manipuloida. Ohjainluokka siis näkee vain omassa tilaobjektissaan olevat muuttujat, joten yksittäisestä ohjainluokasta ei voi käsitellä toisten ohjainluokkien muuttujia. Tästä syystä sovelluksen kasvaessa ohjainluokkien määrä kasvaa, mutta ohjainluokat pysyvät yksinkertaisina.

Tämä katselmus on kuitenkin vain pintaraapaisu AngularJS:n maailmaan. AngularJS:ään löytyy useita lisäosia joilla voi hoitaa esimerkiksi animaatioihin (ng-animate), lokalisaatioon (ng-localization) sekä API-integraatiot (RESTangular).

4.2 Backbone.js

Backbone.js on kevyt MV*-mallin mukainen JavaScript-kirjasto, joka auttaa strukturoimaan selainohjelmistoja.

Backbone rakentuu kolmesta keskeisestä elementistä:

- Backbone.Model on yhden tietueen kapseloiva malliobjekti.

- Backbone.Collection on kokoelma malliobjekteja.

- Backbone.View on osa käyttöliittymää, johon yleensä sidotaan malli tai kokoelma malleja. Näkymillä voi olla alinäkymiä.

Backbone.js on AngularJS:ää huomattavasti pienempi kirjasto. Backbone.js itsessään lähentelee apukirjastoa, joka antaa hyvät rakennuspalat MV*-mallisen SPA:n rakentamiseen.

Backbonen toiminnan perusta on mallien ja kokoelmien laukaisemissa tapahtumissa. Näkymään liitetään malli, kokoelma malleja tai kokonainen alinäkymä, jolla on omat mallinsa. Näkymä voi tarvittaessa kuunnella alielementtien laukaisemia tapahtumia. Tapahtumia laukeaa useissa eri tilanteissa, kuten esimerkiksi mallin tai kokoelman muuttuessa. Yleensä näkymään liitettyjen elementtien muutokset laukaisevat näkymän piirtofunktion. Piirtofunktion voi itse kirjoittaa tahtomallansa tavalla.

```
var View = Backbone.View.extend({
    el: $(body),
    initialize: function () {
        this.collection = new Backbone.Collection;

        this.listenTo(this.collection, 'change', this.render);
    }
    render: function () {
        console.log("Kokoelma muuttui");
    }
});
```

Esimerkkikoodi 3 Näkymän määrittely

Esimerkkikoodissa 3 on yksinkertaistettu näkymän määrittely. Näkymään liitetään uutena luotu kokoelma ja näkymä asetetaan kuuntelemaan muutoksia kokoelmassa. Kokoelman muuttuessa näkymän piirtofunktio ajetaan, jolloin se kirjoittaa JavaScript-konsoliin.

Backbone on suurelta osin ”Tee-se-itse”-tyyppinen apukirjasto, joka jättää kehittäjälle monien osa-alueiden, kuten esimerkiksi piirtofunktioiden toteuttamisen. Backbone ei tarjoa AngularJS:n tapaista kaksisuuntaista sidontaa. Backboneen kannattaa liittää erillinen JavaScript-mallikirjasto käyttöliittymäelementtien piirtämiseen. Backbone toimii

kaikkien mallikirjastojen kanssa, ja Backbonen käyttämä apukirjasto Underscore.js tarjoaa mallitoiminnallisuuden. Underscore.js on JavaScript-kirjasto, joka sisältää joukon yleishyödyllisiä apufunktioita. Suuri osa Backbonen sisäisestä toteutuksesta on toteutettu käyttäen Underscoren apufunktioita. (Underscore.js, 2013.)

```
<html>
    <body>
        <div id="template">

        </div>
    </body>
</html>

<script type="text/javascript">
    var firstName = 'Max',
        lastName = 'Mikkola';

    var template = "Hello, <%- first %> <%- last %>!"

    $("#template").html(_.template(template, { first: firstName, last: lastName}));
</script>
```

Esimerkkikoodi 4. Underscore.js-mallien käyttö

Hello, Max Mikkola!

Kuva 6. Koodiesimerkistä generoitu HTML

Esimerkkikoodissa 4 on demonstroitu yksinkertaistettu tapa käyttää Underscore.js:n malleja. Mallifunktiolle annetaan mallinnettava teksti. Avain-arvo pareina käytettävien muuttujien nimet sekä arvot. Mallinnusfunktio korvaa kuvan 6 mukaisesti ”<%-” ja ”%>” merkkien välissä olevat muuttujanimet muuttujien arvoilla. Malleihin voi myös lisätä JavaScriptiä merkkien väliin, jolloin mallien rakennuslogiikkaa voidaan tehdä monimutkaisemmaksi.

4.3 Yhteenveto

Backbone.js valittiin pilotoitavaksi projektissa. Suurin valintaan vaikuttava tekijä oli projektin mittakaava: kommentointiosio on vain pienehkö osa käyttöliittymää. Backbone.js ei ota kantaa käyttöliittymän muuhun toteutukseen ja tarjoaa kevyet

apurakenteet projektin toteuttamista varten. AngularJS:n toiminta-alue voidaan rajoittaa vain tiettyihin elementteihin sivulla, mutta AngularJS:n muusta järjestelmästä eroavan toimintatyylin sekoittaminen osaksi muuta toimintaa voisi helposti sekoittaa tulevaa kehitystä tarpeettomasti.

Backbone.js itsessään on silti pieni ja kevyt kirjasto. Kirjasto ei tarjoa suurempaan projektiin suoraan yhtä paljon rakennetta kuin AngularJS. Backboneen on kuitenkin saatavilla suuri määrä laajennuksia, joilla vajavaisuudet voidaan paikata. Lukuhetkellä laajennuksia löytyi 209 kappaletta. (BackPlug, 2014.) Esimerkiksi AngularJS:ssä esiintyvä kaksisuuntainen sidonta voidaan toteuttaa ottamalla käyttöön Backbone.ModelBinder (Backbone.ModelBinder, 2013).

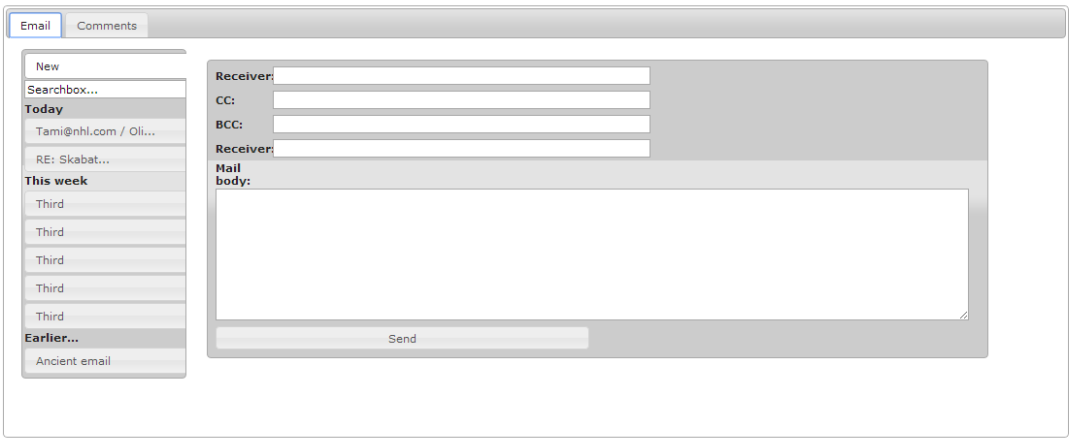
Kirjastot täytyy kuitenkin arvioida uudestaan uuden käyttöliittymäuudistusprojektin tullessa ajankohtaiseksi. Pienikokoiseen projektiin sopiva apukirjasto ei ehkä olekaan paras vaihtoehto kokonaisvaltaisempaa käyttöliittymän uusimista varten. Seuraavan projektin alkaessa kirjastojen toiminnallisuudet ovat myös voineet muuttua suurestikin.

5 Projektin toteutus

Projekti alkoi käyttäjätarpeiden kartoituksella. Kartoitus suoritettiin kyselemällä kaikilta yrityksen työntekijöiltä parannusehdotuksia palvelupyyntö-moduulin kommunikaatiotyökaluihin ja niiden käytettävyyteen. Projektin laajuutta ei oltu vielä rajattu millään tavalla, joten kaikki ehdotukset otettiin vastaan. Parannustarpeita nousi esiin kaksi kappaletta: pitkien sähköpostiketjujen luettavuuden parantaminen ja kommentointitietojen parempi yksilöiminen.

Suurien sähköpostiketjujen lukeminen oli hankalahkoa, sillä sähköpostien lukemiseen ja kirjoittamiseen käytetty käyttöliittymä oli muun käyttöliittymän päällä, jolloin itse sähköpostilistausta ei nähnyt käyttöliittymän alta, eikä viimeksi avattua sähköpostia osoitettu millään tavalla listauksessa. Käyttäjän tarvitsi siis muistaa viimeksi avattu sähköposti, mikä oli välillä turhauttavaa suurien sähköpostiketjujen yhteydessä. Ensimmäinen hahmoteltu parannus oli tuoda sähköpostilistaus mukaan sähköpostien käyttöliittymään.

Sisäistä kommentointia hoidettiin kokonaisuudessaan yhden tekstikentän avulla. Kuka tahansa, jolla oli oikeus muuttaa kommenttikentän sisältöä pystyi esimerkiksi poistamaan toisten kirjoittamat kommentit. Yhdessä tekstikentässä kommenttien aikajärjestys saattoi myös olla sotkuinen.



Kuva 7. Ensimmäinen hahmotelma käyttöliittymästä

Kuvassa 7 on ensimmäinen hahmotelma käyttöliittymästä, joka nitoi sisäisen kommentoinnin sekä sähköpostiliikenteen yhteen käyttöliittymään.

Kommenttitietojen yksilölliseen tallennukseen ei tässä vaiheessa vielä otettu kantaa. Yksilöllinen tallentaminen tarvitsisi paljon muutoksia olemassaolevaan järjestelmään.

Ennen kommenttitietojen tallennuslogiikan tarkempaa mallintamista pidettiin ensimmäinen projektikokous. Kokouksessa tuli ilmi, että projekti oli nykyisessä muodossaan liian laaja. Projektin aikarajoitteiden vuoksi laajuutta piti supistaa, jotta uusien toiminnallisuuksien laadusta voitaisiin varmistua. Sähköposteihin liittyvät parannukset rajattiin pois projektin piiristä, ja sisäisen kommunikaation parannukseen keskityttiin kokonaisvaltaisemmin. Tehtiin päätös uudesta sisäisen kommentoinnin moduulista, joka hoitaisi tietuepohjaisesti kommentointitoiminnallisuudet järjestelmän muiden moduulien puolesta.

Kokouksessa nousi esiin idea uuden JavaScript-kirjaston käytöstä, jolla voisi esittää objektien kommentteja dynaamisemmin ilman erillisiä sivunlatauksia kommenttien päivittämiseksi. Mahdollisia JavaScript-kirjastoja uusien toiminnallisuuksien

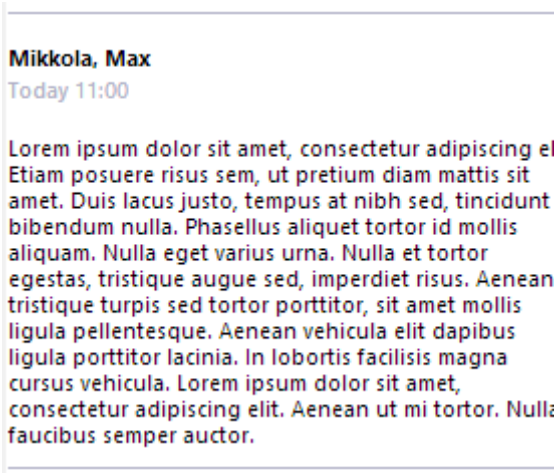
toteutukseen oli useita. Kahta kirjastoa kuitenkin ehdotettiin käytettäväksi projektissa: Backbone.js sekä AngularJS. Yrityksen muilla työntekijöillä oli kokemusta näistä kirjastoista projektin ulkopuolelta, joten oli luontevaa valita ne kokeiltaviksi.

Projekti oli nyt rajattu ja määritelty, joten uuden moduulin rakentaminen voitiin aloittaa.

5.1 Projektin suunnittelu

Ensimmäisen projektikokouksen jälkeinen käyttöliittymäsuunnittelu tapahtui portaittaisesti teknisen toiminnallisuuden kehityksen ohella. Aluksi suurin osa työpanoksesta kohdistuikin juuri palvelimen tallennus- ja hakulogiikan toteuttamiseen. Palvelimen toiminnallisuuksien toteuttaminen oli suoraviivaista, koska yrityksen koodimassassa oli normalisoidut toimintatavat tiedon tallentamisen ja haun toteuttamiseen. Tallennettavan tiedon mallinnus tietokantatauluiksi vaati hieman enemmän suunnittelua, mutta yksinkertaisen tekstidatan tallentamiseen oli myös yleisti käytetyt toimintatavat. Suurin osa suunnittelun tarpeesta kohdistuikin siis käyttöliittymään.

Käyttöliittymän suunnittelu ei ollut yhtä suoraviivaista. Uudelle käyttöliittymälle oli varattu tilaa ikkunan oikeasta laidasta, mutta muita määrityksiä tai rajauksia ei ollut. Aikaisemmin sovitut hyväksymiskriteerit käsittelivät vain moduulin teknistä toimintaa, joten käyttöliittymän ulkoasusta ei ollut vielä tarkkaa tietoa.

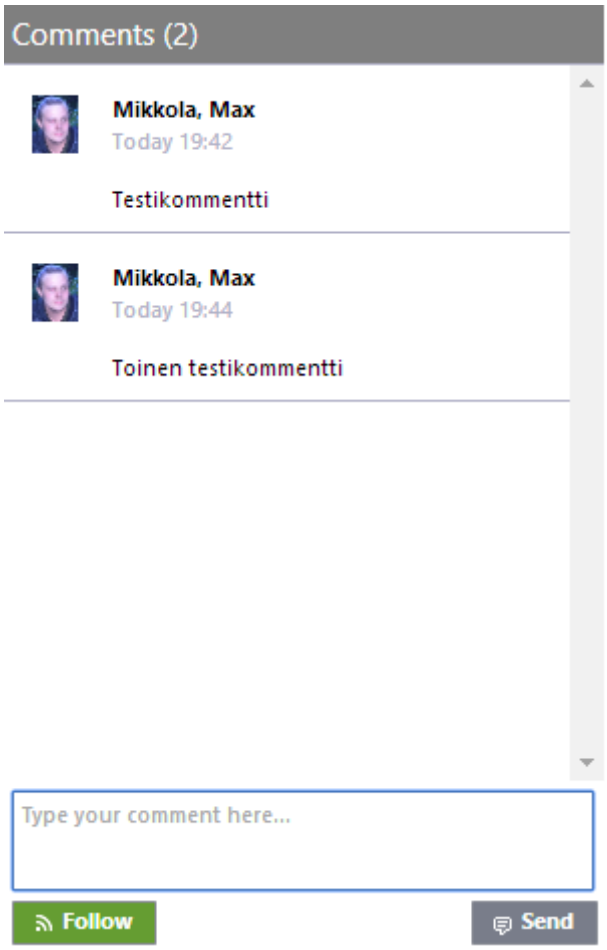


Kuva 8. Yhden kommentin kapseloiva osa käyttöliittymää kehityksen alkuvaiheilta

Kuvassa 8 kommentissa ei vielä ollut melkein mitään muotoilua. Alkuvaiheissa käyttöliittymää käytettiin pääosin palvelinkehityksen testausapuna. Käyttöliittymä alkoi muotoutua osissa muun kehityksen ohella. Käyttöliittymään tehtävistä muutoksista keskusteltiin jatkuvasti ja yksimielisesti hyväksytyt muutokset toteutettiin. Suureksi osin käyttöliittymäosille annetun rajatun sivuttaistilan vuoksi alkoi kehkeytyä pikaviestintää muistuttava käyttöliittymä. Lyhyitä ja nopeita kommentteja suosivaan käyttöliittymään ammennettiin vaikutteita yleisistä toiminnallisuuksista internetin muista pikaviestinsovelluksista, kuten esimerkiksi Skypen pikaviestitintoiminnallisuuksista ja sosiaalisten verkostojen chateista. Vaikutteiden haku oli tärkeää, koska käyttöliittymän intuitiivinen käyttö nojautui yleisesti tuttuun käyttötuntumaan.

Kuva 9. Kommentointiosion suuntaa antava käyttöliittymävedos kehityksen alkupuolelta

Kuten kuvan 9 rosoisesta käyttöliittymävedoksesta huomaa, käyttöliittymän suunnitteluprosessi olikin asteittainen ja valintoja tehtiin ns. yhteisellä tunnepohjalla. Muutama asia pidettiin kuitenkin koko ajan kirkkaina mielessä. Uuden käyttöliittymäosion tuli olla uudistusluontoinen mutta kuitenkin eheä osa nykyistä käyttöliittymää, ja yleisistä parhaista toimintatavoista piti pitää kiinni. Käyttöliittymäosion piti siis näyttää PlanMilliltä, mutta toimia yhtenevästi internetissä yleistyneiden pikaviestinten tavoin. Näin osio ei erottuisi liikaa muusta käyttöliittymästä ja olisi intuitiviinen loppukäyttäjille.



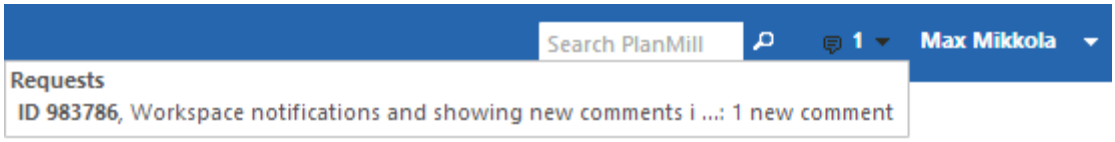
Kuva 10. Käyttöliittymä kehityksen lopussa. Kuvaa on supistettu korkeussuunnassa.

Kuvan 10 vasemmassa alalaidassa esiintyvä kommenttiketjujen seuraustoiminnallisuus nostettiin projektin piiriin aivan viimeisellä viikolla. Tekninen toiminnallisuus kommenttiketjujen seuraamiseen toteutui nopeasti, mutta uusista kommenteista

ilmoittavan valikon suunnitteluun ei jäänyt paljoa aikaa. Suunnittelu jäi siis ilmoitusvalikon osalta vähälle.

Kuva 11. Ilmoitus yhdestä uudesta viestistä sovelluksen yläpalkissa

Kuvassa 11 näkyy yläpalkkiin lisätty uusi ilmoitusvalikko, joka esittää ilman kommenttilistauksen avaamista objektien määrän, joilla on uusia kommentteja.



Kuva 12. Yläpalkista avautuva kommenttilistaus, jossa yhden palvelupyynnön kommentti

Kuvan 12 avattu valikko listaa kaikki objektit, joilla on uusia kommentteja. Rivit ryhmitellään objektityypin mukaan aakkosjärjestykseen.

Kommenttilistauksen esittävään ilmoitusvalikkoon ei ehditty panostamaan kovin paljon projektin puitteissa. Kuitenkin, järjestelmän sisäinen ilmoituskanava uusista kommenteista koettiin tärkeäksi osaksi toimivaa kokonaisuutta. Ilmoitusvalikko erillisenä kokonaisuutena oli tärkeä jatkokehitysprojekti.

5.2 Kehitysympäristö

Yrityksessä uusia toiminnallisuuksia kehitetään sisäverkossa sijaitsevalla kehityspalvelimella. Uutta toiminnallisuutta tai korjausta aloitettaessa automaattisen integraation palvelin Jenkins luo Subversionin pääkehityshaarasta uuden kehityshaaran ja luo samalla samalla nimellä sovellusinstanssin kehityspalvelimelle sekä uuden tietokannan tietokantapalvelimelle. Projektin kehityshaara oli nimeltään **max_thesis_982435**. Numerosarja kehityshaaran nimen lopussa vastaa yhden palvelupyynnön tunnusta, jossa on tarkempia tietoja kehityshaarassa tehtävästä kehitystyöstä.

Yrityksen versionhallintapalvelimena toimii Visual SVN, ja asiakassovelluksena käytetään TortoiseSVN:ää. Java-kehittimenä on Eclipse IDE. Kehityspalvelimen palvelinohjelmistona toimii Tomcat 7, ja tietokantapalvelimena toimii Microsoftin MS SQL Server 2012. Tietokantakehittimenä käytössä on Microsoft SQL Studio 2012.

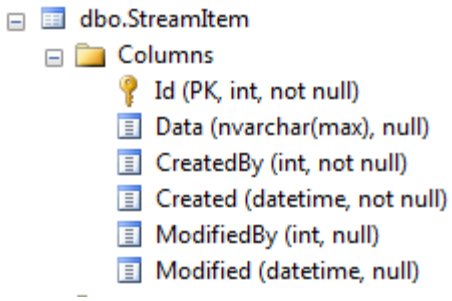
5.3 Tekninen toteutus

Projektin tekninen toteutus vaati useita muutoksia ja lisäyksiä olemassaolevaan järjestelmään. Käyttöliittymäelementtien toiminnallisuudet toteutettiin JavaScriptillä, ja palvelinpuolen tallennus- ja hakuoperaatiot Javalla ja T-SQL-tietokantakyselyillä. Muut uusiin toiminnallisuuksiin liittyvät tekijät kuten esimerkiksi luku- sekä kirjoitusoikeuksien hallinta toteutettiin järjestelmän parametreja hyödyntäen.

Tietokanta

PlanMillin tuotantoympäristössä on kaksi tietokantapalvelinta, jotka on peilattu kahdelle erilliselle tietokantapalvelimelle. Jokaisella asiakkaalla on oma tietokanta yhdellä palvelimella.

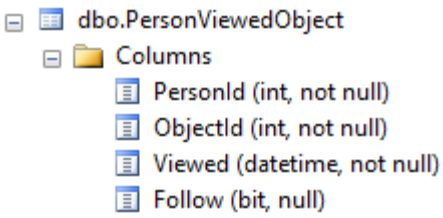
Projektissa luotiin kolme uutta tietokantataulua: StreamItem, RequestHasStreamItem sekä PersonSeenObject. StreamItem-tauluun tallennetaan yksilöidysti kommenttien tiedot.



Kuva 13. StreamItemin tietokantasarakkeet

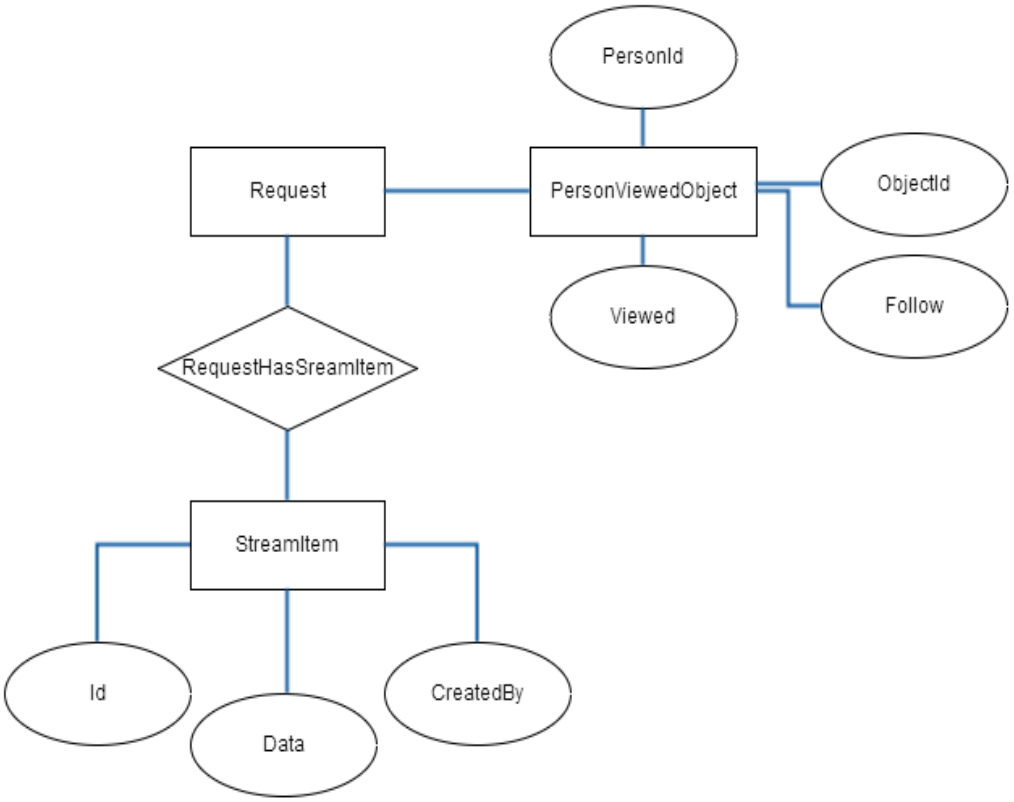
Kuva 13 on kuvakaappaus StreamItem-taulusta kehitystietokannassa. Tietueen yksilöivänä tietona käytetään inkrementaalisesti generoitua ID:tä, StreamItem.Id:tä. Tauluun tallennetaan myös kommenttiteksti, luomispäivämäärä sekä viittaus kommentoijaan.

PersonViewedObject-tauluun tallennetaan tietoja kommenttien näkemisestä ja seuraamisesta. Taulun avulla pidetään kirjaa ajankohdista, jolloin järjestelmää käyttävät henkilöt ovat nähneet objektien kommenttitietoja.



Kuva 14. PersonViewedObjectin tietokantasarakkeet

Kuvassa 14 alimpana esiintyvään Follow-sarakkeeseen tallennettaan tieto siitä, haluaako käyttäjä seurata järjestelmän objektiin. Järjestelmän yläpalkissa sijaitseva ilmoitusvalikko valitsee näytettäväksi ne objektit, joita käyttäjä seuraa ja joilla on uusia kommentteja.



Kuva 15. ER-kaavio uusista tietokantatauluista

Kuvan 15 ER-kaavio demonstroi uusien tietokantataulujen relaatiota muuhun järjestelmään. StreamItem-taulu voidaan liittää muihin tauluihin relaatiotaululla. Relaatiota varten projektissa luotiin RequestHasStreamItem-taulu, joka tallentaa sidokset palvelupyyntöjen ja kommenttien välillä. Muita relaatiotauluja ei lisätty, mutta kommenttimoduuli tukee kaikkia muita järjestelmän moduuleita ja uusia relaatiotauluja voidaan lisätä tietokantaan tarpeen mukaan.

Parametrit

Parametrit ohjaavat PlanMill-toiminnanohjausjärjestelmän toimintaa. Parametreja on käytössä yli 30 000 kappaletta, ja niiden avulla määritellään miltei kaikki järjestelmän toiminta. Järjestelmän muokkaaminen ja räätälöinti asiakkaiden tarpeisiin on nopeaa, sillä parametreja voi muokata sekä lisätä lennosta.

Kaikki asiakasinstanssit hakevat parametrit muistiin jaetulta palvelimelta. Instansseihin voi tallentaa parametreja myös lokaalisti, jolloin niiden määritelmät yliajavat jaetut parametrit. Näin jaetulla palvelimella sijaitsevat perusparametrit takaavat järjestelmien normaalin toiminnan, ja lokaalit versiot mahdollistavat yksilölliset asiakasräätälöinnit.

Projektissa lisättiin järjestelmään useita parametreja. Parametrit määrittelevät kommenttien haussa sekä tallennuksessa käytettävät SQL-lauseet, lokalisoituvat tekstit eri kielillä, sekä millä objekteilla osio on käytössä ja kenellä on oikeus käyttää osiota.

System.Stream.SQL

Local	Name	Modified	Data	Request ID	Description	Comment	Enabled	Role	User
	Get object follow state		SELECT Follow FROM PersonViewedObject WHERE ObjectId = ? AND PersonId = ?				Yes		

	Toggle follow		DECLARE @ObjectId int = ?,
			@PersonId int = ?,
			@Follow int = ?;
			
			IF EXISTS(SELECT 1 FROM PersonViewedObject WHERE ObjectId = @ObjectId AND PersonId = @PersonId)
			BEGIN
			UPDATE PersonViewedObject
			SET Follow = @Follow, Viewed = GETDATE()
			WHERE ObjectId = @ObjectId AND PersonId = @PersonId
			END
			ELSE
			BEGIN
			INSERT INTO PersonViewedObject (ObjectId,PersonId,Viewed,Follow)
			VALUES (@ObjectId,@PersonId,GETDATE(),@Follow)
			END
							Yes		

Kuva 16. Kaksi SQL-lausetta kommentointimoduulin parametreissa

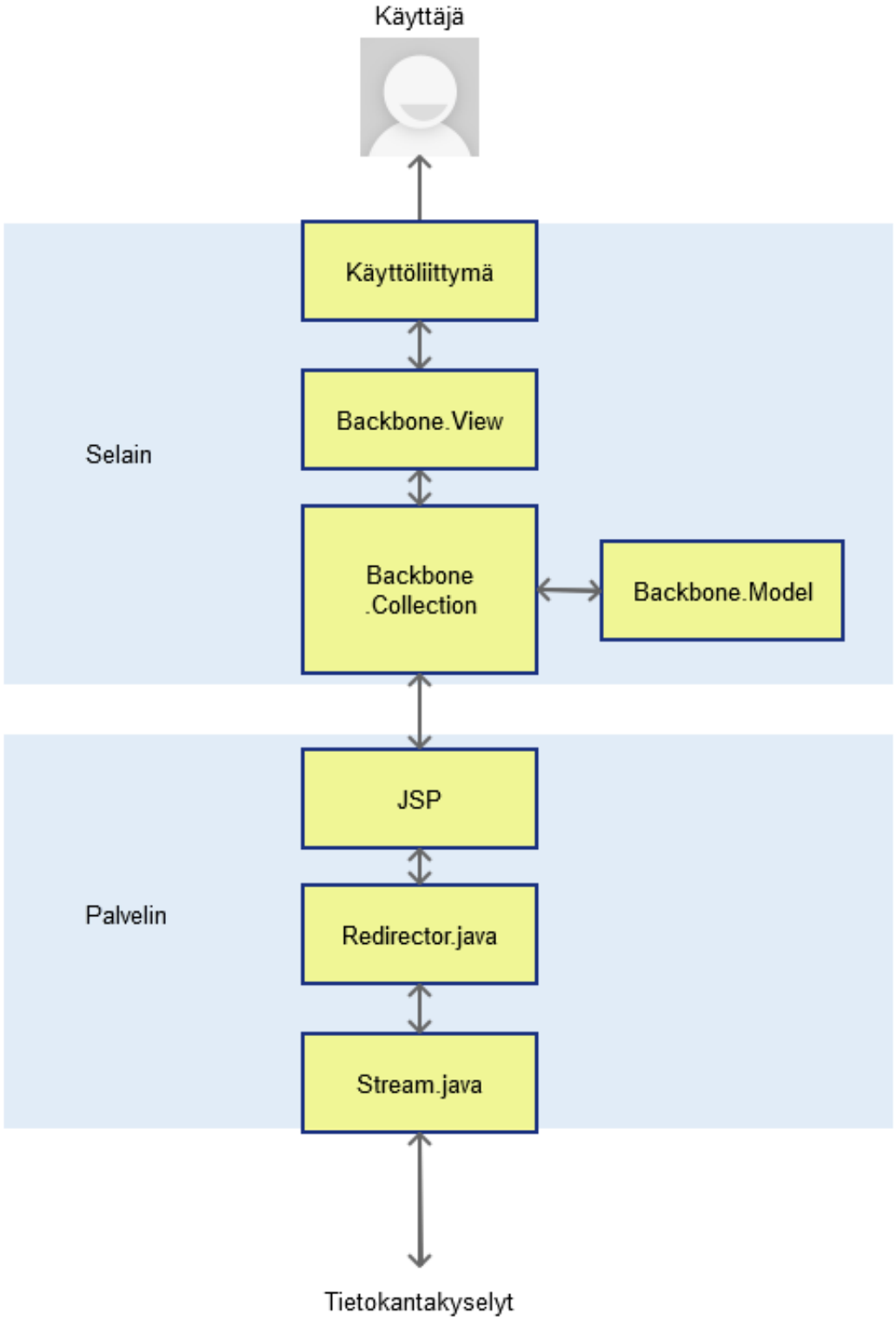
Kuvassa 16 esiintyy kaksi parametria, joiden avulla määritetään kommentointimoduulin tietokantakyselyt. Kuvassa esiintyy kommenttien seurausominaisuuteen liittyviä SQL-lauseita. Tietokantakyselyiden määrittelyyn on käytössä 8 parametria.

Parametrien tietoa esittävän sarakkeen harmaa teksti tarkoitaa sitä, että parametrit on ladattu instansseille yhteiseltä parametripalvelimelta, eivätkä ne sijaitse sovellusinstanssin omassa tietokannassa. Tietokantaan paikallisesti tallennetut parametrit lukevat mustalla.

Käyttöliittymä

PlanMillin käyttöliittymä rakentuu parametrien määritelmien mukaisesti XML-muodossa, jonka avulla muodostetaan HTML-dokumentti, joka lähetetään käyttäjän

selaimeen. HTML-dokumentin latauduttua kommenttiosion luovaa JavaScript-funktiota kutsutaan muun alustuslogiikan ohessa. Funktio luo osion elementit uuteen HTML-fragmenttiin, joka lisätään sivun oikeaan laitaan. Kun fragmentti on lisätty DOM:iin, alustetaan osion toiminta Backbonen avulla.



Kuva 17. Tiedon kulku kommenttien haku- sekä tallennusoperaatioissa

Kuva 17 esittää yksinkertaistetun version tiedon kulusta kommenttitietojen tallennus- sekä hakuoperaatioissa. Backbone.Collection pitää yllä listaa malleista, jotka sisältävät kommenttitietoja. Kokoelmassa näkyvät mallit piirtyvät näkymän piirtofunktion kautta

käyttöliittymään. Kokoelma päivittyy, kun käyttäjä lisää uuden kommentin tai kun palvelimelta haettava kokoelma eroaa selaimen kokoelmasta. Selain kutsuu palvelimen moduuleita JSP:n, eli Java-palvelinsivun (Java Server Page) avulla. Kommentteja ladatessa sekä tallennettaessa kutsu JSP:lle toteutetaan asynkronisena taustaprosessina Ajaxilla.

JSP kutsuu kommentointimoduulin toimintoja selaimelta tulevien ohjeiden mukaisesti. Kommentointimoduuli joko tallentaa uuden kommenttitietueen objektille tai hakee kaikki objektille kuuluvat kommentitietueet ja palauttaa ne selaimelle JSON-muodossa.

Palvelimelta saadut uudet kommenttitiedot lisätään Backbone.js:n ylläpitämään kommenttikokoelmaan. Muutokset kokoelmassa saavat aikaan tapahtumaketjun, jonka seurauksena kommenttiosio piirtyy uudestaan. Piirtofunktio käyttää hyväkseen Underscore.js:n tarjoamia kevyitä HTML-malleja, joihin kommenttien tiedot asetetaan.

5.4 Testaus ja laadunvarmistus

Koodin laadun varmistamiseksi sekä ongelmien välttämiseksi käytettiin staattista analyysiä. Palvelinkoodin analysointiin käytettiin FindBugs-liitännäistä Eclipseen, tietokantakyselyiden optimointiin ja analysointiin SQLEnlight-työkalua, ja JavaScriptin analysointiin JSHint-työkalua.

Kehityksen aikana palvelimen sekä selaimen toiminnallisuuksia testattiin yksikkötesteillä, ja kokonaisuuden toimintaa testattiin funktionaalisesti. Projektin myöhemmässä vaiheessa testasimme tarkemmin rajatapauksia sekä käyttäjäsyötteiden puhdistusta.

Myös toiminnallisuuden stressinsietokykyä testattiin. Kymmenen henkilöä lisäsi samalle objektille kommentteja niin nopeasti kuin mahdollista. Selain hidastui huomattavasti vasta, kun kommentteja oli useita satoja. Testissä lisättiin yhdelle objektille noin tuhat kommenttia, joka on paljon enemmän kun oletettava normaali määrä.

5.5 Siirto tuotantoon

Ennen siirtoa tuotantoon projektissa tuotettu koodimassa katselmoitiin kahdessa vaiheessa. Ensimmäisessä vaiheessa koodimassa katselmoidaan pintapuolisesti. Koodin syntaksi, kommentointi, luettavuus sekä yleinen ilme tarkistetaan. Koodin laatua, kehittäjien sopimia konventioita ja parhaita toimintatapoja pidetään näin yllä.

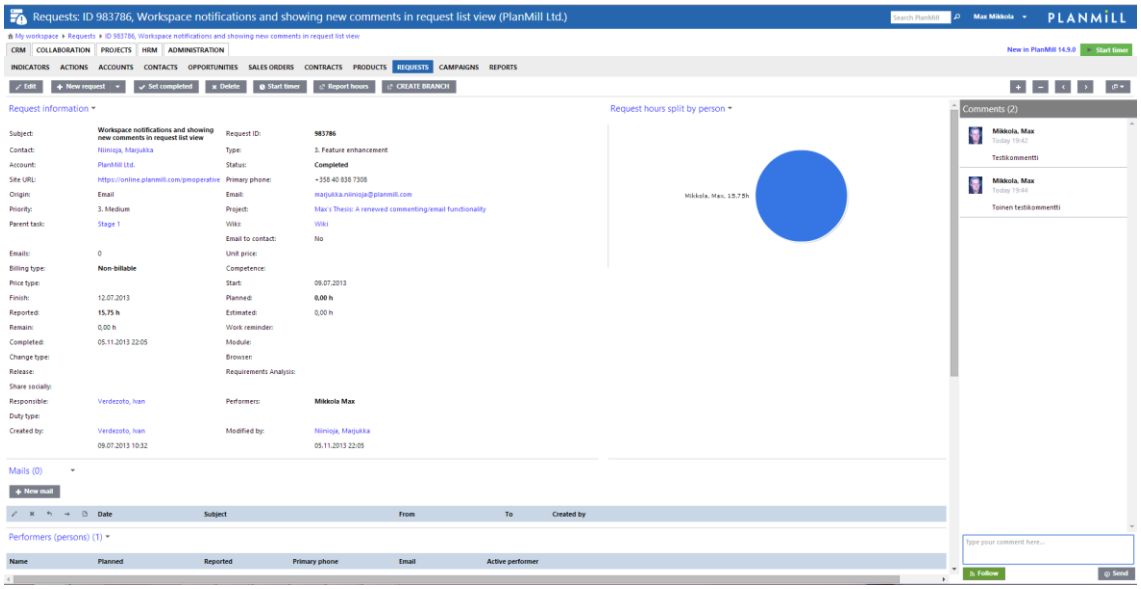
Katselmoitu toiminnallisuus testattiin funktionaalisesti kehittäjän laatiman testisuunnitelman mukaisesti. Kehittäjä oli itse yksikkötestannut moduulin sisäiset yksittäiset toiminnallisuudet kehitysvaiheessa, joten katselmoinnin jälkeinen testaus oli kokonaisuuden yhteisen toiminnan testaamista. Järjestelmälle ajettiin myös perustoiminnallisuudet kattava regressiotestaus.

Läpäistyjen testien jälkeen koodimassa katselmoitiin uudestaan, syvemmin. Toisen kierroksen katselmoinnissa otetaan kantaa tehtyihin koodimuutoksiin, käytettyihin algoritmeihin sekä logiikkaan. Toisen katselmuksen läpäissyt toiminnallisuus yhdistettiin PlanMillin pääkehityshaaraan. Ongelmatilanteessa kehittäjä olisi korjannut ilmenneet viat, ja prosessi alotettaisiin alusta.

Pääkehityshaaraan tehdyt muutokset päivitettiin tuotantopalvelimille elokuun alussa vuonna 2013.

6 Lopputulos

Toimiva kokonaisuus siirrettin tuotantoon ja aktivoitiin käytettäväksi PlanMillin omassa käytössä olevaan tuotantoinstanssiin.



Kuva 18. Kommentointi osana käyttöliittymää yksittäisen palvelupyynnön yhteenvetosivulla.

Kuvassa 18 nähdään uusi kommentointiosio käyttöliittymässä. Moduuli aktivoitiin toimivaksi vain järjestelmän palvelupyyntöjen yhteydessä. PlanMillin sisäisessä käytössä kommentoinnille ei ollut tarvetta palvelupyyntöjen ulkopuolella. Mahdollisten sisäisten- tai asiakastarpeiden vaatiessa moduuli voidaan aktivoida muualle helposti ja nopeasti.

Jatkokehitys

Uusista kommenteista ilmoittava valikko tarvitsee jatkokehitystä. Osion informatiivinen hyöty ei itsessään ole vielä kovin suuri. Nyt kun ilmoitusten dynaaminen lataus on teknisesti toteutettu, voidaan käyttöliittymään panostaa. Dynaaminen ilmoitus uusista viesteistä, objektimuutoksista tai esimerkiksi puuttuvista tuntikirjauksista olisi suurenmoinen parannus nykyiseen verrattuna. Ilmoitusvalikkoa voitaisiin käyttää myös järjestelmän sisäisenä ilmoituskanavana.

Long polling, eli pitkät pyynnöt, voisi vähentää palvelimelle lähetettävien pyyntöjen määrää. Long pollingissa palvelin ei palauta selaimelle heti tyhjää vastausta, vaan

odottaa tietyn aikamäärän, esimerkiksi puoli minuuttia. Jos puolen minuutin jälkeen ei ole vieläkään mitään palautettavaa, palvelin lähettää selaimelle tyhjän vastauksen. Long polling lisäisi jossain määrin palvelinkuormitusta, mutta parantaisi relevantin datan vasteaikoja ja vähentäisi palvelimelle lähetettyjen pyyntöjen määrää. Long polling on hyvä kandidaatti jatkotutkimukselle myöhemmissä projekteissa. (Even faster websites, 2009.)

Vaikka AngularJS:n laaja-alaisempi prototypointi ei oikeastaan olekaan osa projektin jatkokehitystä, kannattaisi kirjastoa kuitenkin kokeilla enemmissä määrin. AngularJS:n tapa pilkkoa ja rakentaa kokonaisia selainsovelluksia on mainio, ja se on itsessään Backbone.js:ää kattavampi paketti.

7 Yhteenveto

Projektin alkuperäinen tavoite oli parantaa järjestelmän sisäistä sekä ulkoista kommunikointia. Sisäinen kommentointi koettiin tärkeämmäksi kehityskohteeksi, ja ulkoinen kommunikaatio rajattiin ulos.

Järjestelmään toteutettiin uusi moduuli, jonka tarkoitus oli mahdollistaa järjestelmän muiden objektien kommentointi. Moduulin toteutus vaati muutoksia Java-luokkiin, kolme uutta tietokantataulua sekä muutoksia käyttöliittymän toimintaan ja ulkonäköön. Moduulin täytyi pystyä lataamaan sekä tallentamaan kommenttitietoja ilman uutta sivunlatausta. Dynaamisemmin toimivan käyttöliittymäosion jäsentelyn toteuttamiseen valittiin Backbone.js. Kommentointiosio toteutettiin ajallaan ja kokonaisuus toimi mainiosti.

Uusista kommenteista ilmoittava valikko lisättiin projektin piiriin vasta viimeisellä viikolla. Ilmoitusvalikon tekninen toiminnallisuus saatiin kuitenkin toteutettua hyvin, ja käyttöliittymäkin oli käytettävässä kunnossa. Ilmoitusvalikko olikin lopulta ehkä kokonaisuuden parhaiten vastaanotettu toiminnallisuus. Käyttäjät olivat todella tyytyväisiä siihen, ettei palvelupyyntöihin tulleita kommentteja tarvinnut enää tarkistaa aktiivisesti.

Projekti oli kokonaisuutena onnistunut. Kaikki kaavaillut toiminnallisuudet toteutettiin ajallaan, eikä toteutuksessa ilmennyt suuria ongelmia.

Lähteet

AJAX (2005). Jesse James Garret. Verkkodokumentti. https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf. Luettu 22.4.2014.

AngularJS. Green,B. ja Seshadri, S. O'Reilly, 2013.

Backbone.js. Verkkodokumentti. http://www.backbonejs.org. Luettu 15.6.2013.

Backbone.ModelBinder, Verkkodokumentti. https://github.com/theironcook/Backbone.ModelBinder. Luettu 22.4.2014.

BackPlug.io. Verkkodokumentti. http://backplug.io/. Luettu 20.4.2014.

Developing Backbone.js Applications. Osmani. A. O'Reilly, 2013.

Dokumenttiobjektimalli. Verkkodokumentti. World Wide Web Constortium W3C http://www.w3.org/DOM/. Luettu 22.3.2014.

Even faster websites. Steve Sounders. O'Reilly, 2009.

JSON. Verkkodokumentti. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. Luettu 23.4.2014.

MVC. (2009) Verkkodokumentti. http://www.jdl.co.uk/briefings/MVC.pdf. Luettu 21.4.2013.

Professional services automation. (16.3.2014) Verkkodokumentti. Wikipedia http://en.wikipedia.org/wiki/Professional_services_automation. Luettu 22.4.2014.

Restangular. Verkkodokumentti. https://github.com/mgonto/restangular. Luettu 22.4.2014.

Single Page Web Applications. Mikowski M. S., ja Powell J.C (2013). Manning.

Toiminnanohjausjärjestelmä (30.3.2013) Verkkodokumentti. Wikipedia. http://fi.wikipedia.org/wiki/Toiminnanohjausjärjestelmä. Luettu 22.4.2014.

Underscore.js. Verkkodokumentti. http://www.underscorejs.org. Luettu 15.6.2013.