Juha Patrikka

Selainkäyttöisen toimeksiantojärjestelmän suunnittelu ja toteutus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

28.11.2012

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Juha Patrikka Selainkäyttöisen toimeksiantojärjestelmän suunnittelu ja toteutus 20 sivua + 2 liitettä 28.11.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja(t)	asiakkuuspäällikkö Mia Halonen lehtori Olli Hämäläinen

Insinöörityössä suunniteltiin ja toteutettiin yrityksen mainososastolle vanhan toimeksianto-menettelytavan korvaava järjestelmä web-sovelluksena. Tarkoituksena oli sovelluksen avulla edistää mainostoimiston tuotantotiimin itseohjautuvuutta, nopeuttaa töiden läpimenoaikoja ja vähentää päivittäiseen tuotantotyöhön liittyvää suunnittelu- ja esimiestyötä. Sovellus tuli toteuttaa mahdollisimman paljon avoimen lähdekoodin ohjelmistoja hyödyntäen. Toteutuksessa sovellettiin nykyään laajalti käytettävää Scrum-menetelmää jakaen sovelluskehitysprojekti kahden viikon sprintteihin.

Sovelluksen tietokannaksi valittiin MySQL, web-palvelinohjelmistoksi Apache HTTP Server ja käytettäväksi ohjelmointikieleksi PHP, joka on dynaamisissa web-palveluissa nykyään käytetyin ohjelmointikieli. Sovellus jaettiin loogisiin osiin Model-View-Controller-arkkitehtuurin mukaisesti ja toteutettiin Zend Framework -ohjelmistokehyksen avulla.

Täsmällisen vaatimusmäärittelyn ja kunkin sprintin huolellisen suunnittelun ansiosta sovelluskehitysprojekti saatiin valmiiksi ja tuotantokäyttöön aikataulun mukaisesti. Tehtyjen laskelmien mukaan uuden sovelluksen kautta tulleiden toimeksiantojen määrä lisääntyi selvästi aiempaan toimeksiantoprosessiin verrattuna, joten sovellukselle asetetut tavoitteet toimeksiantojen käsittelyn tehostamiseksi toteutuivat.

Avainsanat	MVC, Zend Framework, Scrum, toimeksiantojärjestelmä

Author(s) Title Number of Pages Date	Juha Patrikka The design and implementation of web browser based assignment system 20 pages + 2 appendices 28 November 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Mia Halonen, Account Manager Olli Hämäläinen, Senior Lecturer

In this thesis a new web browser based assignment system was designed and implemented for an in-house advertising agency of a company to replace the old procedure of assigning tasks. By using the system, the aim was to enhance the self-guidance of the production team of the advertising agency, to speed up the lead time of production work and to reduce the everyday resourcing work. The system was intended to be implemented using open source software as much as possible. The Scrum framework, which is currently widely used, was utilized to split the software developing project into 2-week sprints.

MySQL was selected as the database and Apache HTTP Server as the web server. PHP, which is currently the most commonly used programming language in dynamic web services, was selected as the programming language. The application was split into logical parts according to Model-View-Controller architecture and was implemented using Zend Framework as the software framework.

Due to precise description of requirements and thorough design of each sprint, the software development project was made ready and deployed in schedule. According to calculations made, the count of assignments coming via the new assignment system was significantly increased compared to the old procedure of assigning tasks. This means that the aims to enhance dealing with assignments were achieved.

Keywords	MVC, Zend Framework, Scrum, assignment system

Sisällys

Lyhenteet

BSD	Berkeley Software Distribution. Yksi käytetyimmistä avoimen lähdekoodin lisensseistä.
CSS	Cascading Style Sheets. Rakenteisten dokumenttien, yleisimmin HTML-dokumenttien tyyliohje.
HTML	Hypertext Markup Language. Kuvauskieli, jolla voidaan kuvata hyperlinkkejä sisältävää tekstiä eli hypertekstiä.
HTTP	Hypertext Transfer Protocol. Hypertekstin siirtoprotokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon.
MIME	Multipurpose Internet Mail Extensions. Määrittelee joukon sisältötyyppejä eli MIME-tyyppejä, joita käytetään mm. HTTP:ssä ilmaisemaan välitetyn datan muotoa.
MVC	Model-View-Controller. Ohjelmistoarkkitehtuurityyli, jossa käyttöliittymä on erotettu sovellusaluetiedosta.
PHP	Hypertext Preprocessor. Ohjelmointikieli, jota käytetään erityisesti Web-palvelinympäristöissä dynaamisten web-sivujen luonnissa.
URL	Uniform Resource Locator. Yksilöllinen osoite Internetissä olevalle tiedostolle.

1 Johdanto

Insinöörityön tavoitteena oli kehittää SOK Markkinointi ja asiakasvuorovaikutus -yksikön Mainonnan tuotannon tuki ja kehitys -tiimille uusi, vanhan toimintamallin korvaava toimeksiantojärjestelmä web-sovelluksena. Järjestelmän avulla pyritään entisestään vähentämään päivittäistä töiden resursointia sekä lyhentämään töiden läpimenoaikoja edistämällä tiimin itseohjautuvuutta.

Sovellus toteutettiin MVC -mallin (Model-View-Controller) mukaisesti PHP-ohjelmointikielellä hyödyntäen Zend Framework -ohjelmointikehystä. Sovelluksen tietokannaksi valittiin MySQL. Työssä selitetään käytettävät tekniikat yleisellä tasolla sekä kuvataan projektin vaiheet ja projektissa syntyvän sovelluksen rakenne. Työssä ei käsitellä tarvittavia ohjelmistoasennuksia eikä web-palvelimeen (Apache) liittyvää konfiguraatiotyötä. Työssä kuvataan myös yleisellä tasolla työn toteutuksessa käytettävä Scrum-menetelmä, joka on eräs ketterissä ohjelmistokehitysprojekteissa käytetty projektinhallinnan menetelmä.

2 Toimeksiannot mainostoimiston tuotantotiimissä

Mainostoimiston tuotantotiimissä, jossa työskentelen, tuotetaan tuotantopalveluita yksikön suunnittelutiimeille. Käytännössä tuotantopalvelut ovat mainonnantuotannon prosesseihin liittyviä tuotantotöitä, kuten kuvankäsittelypalveluita, painoaineiston tuotantoa ja web-tuotantoa. Yksittäisten tuotantotehtävien lisääntyessä ja tuotantotiimin henkilökunnan kasvaessa on myös toimeksiantojen toteutuksen suunnittelun ja resursoinnin merkitys kasvanut. Samalla myös toimeksiantojen vaiheiden seurannasta on tullut entistä tärkeämpää myös toimeksiantajien näkökulmasta.

Tuotantotiimillä on käytössään PDF-lomakkeeseen ja yhteiskäyttöiseen sähköpostilaatikkoon perustuva toimeksiantomenettely. Tämän avulla tuotantotöiden jakaminen tiimiläisten välillä on helpompaa ja tehokkaampaa, mikä nopeuttaa töiden läpimenoaikoja sekä vaatii vähemmän päivittäiseen tuotantotyöhön liittyvää esimiestyötä. Toimeksiantomenettelyllä pyritään lisäämään tiimin itseohjautuvuutta toimeksiantojen resursoinnissa. PDF-lomake toimii toimeksiannon työmääräimenä sisältäen toimeksiantoon liittyviä tietoja. Sähköpostilaatikko toimii toimeksiannon vastaanottavalla osapuolella työ-

listana, josta vapaana oleva työntekijä poimii itselleen toimeksiannon suoritettavaksi. Käytännössä tuotantohenkilö toimeksiannon poimiessaan siirtää yhteiskäyttöiseen sähköpostilaatikkoon lähetetyn PDF-lomakkeella liitteistetyn viestin sähköpostipalvelimella omaan kansioonsa, jotta muut tuotantohenkilöt eivät poimisi samaa toimeksiantoa itselleen.

Tämän menettelytavan heikkoutena on havaittu olevan, että tuotantohenkilöiden käyttämissä sähköposti-clienteissa viestit eivät aina ole synkronissa sähköpostipalvelimen kanssa. Seurauksena voi mahdollisesti olla se, että sama toimeksianto on suorittavana samanaikaisesti useammalla tuotantohenkilöllä. Menettelytavasta ei toimeksiantajalle ole merkittävää hyötyä, sillä hän ei pysty seuraamaan itse toimeksiantonsa valmistumista.

3 Uudistettu toimeksiantoprosessi

Valintakriteerit ja toteutusmenetelmät

Edellä mainituista heikkouksista johtuen tuotantotiimissä pohdittiin vaihtoehtoja PDF-lomakepohjaisen menettelytavan korvaamisella jollain muulla tiimin itseohjautuvuutta edistävällä tavalla kustannustehokkaasti mielellään jo olemassa olevia laitteistoja ja sovelluksia hyödyntäen. Lisäksi koettiin tärkeäksi, että valittavalla sovelluksella saadaan kerättyä kaikki toimeksiannon suorittamiseksi tarvittava tieto ja että sovellus olisi tarvittaessa tätä varten räätälöitävissä. Myös toimeksiantajan mahdollisuus seurata työn etenemistä koettiin tärkeäksi. Pohdinnan seurauksena tuotantotiimissä päätettiin toteuttaa itse internet-selainkäyttöinen tietokantasovellus tätä tarkoitusta varten. Sovellusta varten tarvittavat palvelimet, palvelinohjelmistot ja varusohjelmat olivat mainostoimistolla jo ennestään käytössä. Sovellusprojektin esitutkimusdokumentaatio on liitteessä 1.

Työssä päätettiin myös kokeilla sovelluskehitystä käyttämällä ns. ketterää menetelmää. Menetelmäksi valittiin Scrum, jota tosin yhden henkilön projektiryhmän vuoksi jouduttiin soveltamaan. Web-palvelimeksi valittiin Apache ja tietokannaksi MySQL. Molemmat olivat jo ennestään asennettuna yksikön sisäverkossa olevalla palvelimella. Ohjelmointikieleksi valittiin PHP ja käytettäväksi ohjelmistokehykseksi Zend Framework. Tuotteen

omistaja, käytännössä tuotantotiimin tuotantoesimies, määritteli seuraavassa kappaleessa kuvatut vaatimukset.

Järjestelmän vaatimukset

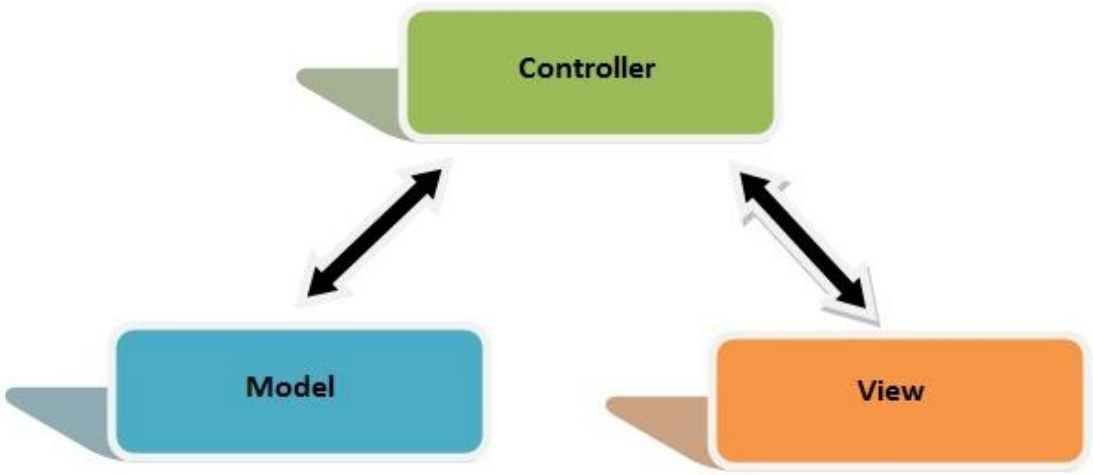
Sovelluksella voidaan hallinnoida tiimille tehtäviä toimeksiantoja (käytetään myös termiä ”tehtävä”). Tietojen tulee olla pysyviä. Sovelluksen perustoimintoja ovat tehtävän lisäys järjestelmään soveltuvilla lisätiedoilla liitetiedostot mukaan lukien, jo olemassa olevan tehtävän kopiointi uudeksi tehtäväksi, tehtävän tilan (toimeksianto, työn alla, valmis, peruttu) muuttaminen, tehtävien listaus käyttöliittymässä ja suodattaminen tilan mukaisesti sekä tehtävän poisto. Tehtävän tilan tai tietojen muuttamisen ajankohta tallentuu aikaleimana tietokantaan. Tehtävän tilojen tulee olla esitettynä käyttöliittymässä siten, että ne voidaan helposti erottaa toisistaan eri värien avulla. Sovellukseen tulee voida kirjautua, mutta käyttö ei edellytä kirjautumista. Kirjautuneilla käyttäjillä on käytössään toimintoja tehtävien hallinnointiin. Ei-kirjautuneet käyttäjät voivat lisätä tehtäviä, mutta eivät pysty muuttamaan tehtävän tilaa tai poistamaan tehtävää. Vain tietyissä tiloissa olevat tehtävät on mahdollista poistaa. Sovellus on internet-selainkäyttöinen, ja sen tulee olla käytettävissä sekä PC- että Mac-työasemalla. Vaatimukset on kuvattu tarkemmin liitteessä 2.

4 Insinöörityössä käytettyjen menetelmien teoriaa

4.1 Model-View-Controller -arkkitehtuuri

Model-View-Controller -arkkitehtuurilla, lyhennettynä MVC:llä, tarkoitetaan sovelluksen suunnittelumallia, jolla sovellus voidaan jakaa kolmeen loogiseen osaan. Sovelluksen jakaminen osiin yksinkertaistaa sovelluskehitystä ja sovelluksen ylläpitoa. Samalla ohjelmistokehittäjien työnkulku selkeytyy ja koodin uudelleenkäytettävyys sekä sovelluksen jatkokehitettävyys paranee (1, s. 201). MVC-arkkitehtuurin kehitti alun perin jo vuonna 1978 Trygve Reenskaug, ja se on säilynyt tähän päivään pääosin muuttumattomana (2; 3). MVC sopii erityisen hyvin interaktiivisiin dynaamisiin web-sovelluksiin, jossa sovelluksen käyttäjä on yhteydessä web-sivustoon useiden edestakaisten pyyntö-vastaustapahtumien kautta. (4.)

MVC-arkkitehtuurin (kuva 1) osat ovat Malli (Model), Näkymä (View) ja Ohjain (Controller). Malli vastaa sovelluksen liiketoimintalogiikasta mm. mahdollistaen pääsyn tietolähteisiin ja tarjoten uudelleenkäytettäviä kirjastoluokkia. Toisin sanoen mallikerros sisältää yleensä tarvittavan tiedon ja sen käsittelyn. Näkymä tarjoaa ulkoasun sovelluksen tarjoaman tiedon esittämiseksi ja tietojen keräämiseksi käyttäjiltä. Tämän kerroksen tehtävä on vain esittää muilta osilta saatuja tietoja, ei prosessoida niitä. Varsinainen sovelluslogiikka kuuluu ohjainkerrokselle, joka yhdistää näkymäkerroksen tarjoaman ulkoasun mallikerrokselta saataviin tietoihin. Ohjain vastaanottaa käyttäjän komentoja ja muuttaa mallin ja näkymän tilaa niiden mukaisesti.



Kuva 1. Model-View-Controller –arkkitehtuuri.

Osat ovat erillisiä, mutta yhteydessä toisiinsa ja siten myös niiden toteuttaminen voidaan eriyttää. MVC-arkkitehtuuri mahdollistaakin siten myös sovelluskehityksen jakamisen kolmeen osaan – kehitykseen, visuaaliseen suunnitteluun ja integrointiin. Kehittäjät työskentelevät mallikerroksen parissa, ja heillä on usein osaamista ohjelmoinnista, tietokantojen hallinnoinnista, algoritmeista, arkkitehtuurista ja tiedon validoinnista. Visuaaliset suunnittelijat taas rakentavat näkymäkerroksen tarjoamaa ulkoasua graafisen suunnittelun, HTML:n, CSS:n ja Javascriptin avulla. Visuaalinen suunnittelu usein johtaa näkymämallien prototyyppien kehittelyyn pyrkimyksenä sovelluksen ihanteellinen toiminnallisuus. Integraatio tapahtuu ohjainkerroksessa, jossa koostetaan kehittäjien ja visuaalisten suunnittelijoiden työ toimivaksi sovellukseksi. Integraattorit vastaavat staattisten näkymämallien pilkkomisesta dynaamisiin alueisiin. Lisäksi he vastaavat pyyntö-

tietojen vastaanottamisesta, niiden välittämisestä mallikerrokselle, mallin tuottamien tulosten tulkinnasta ja tämän informaation välittämisestä näkymäkerrokselle.

Yleisesti ottaen työnkulku etenee siten, että visuaaliset suunnittelijat luovat aluksi toiminnallisiin määrityksiin pohjautuvan staattisen prototyypin. Kehittäjät katselmoivat tämän, ja mikäli prototyypin toiminnallisuudet todetaan toteutuskelpoisiksi, luodaan kehityssuunnitelma rajapintoineen ja toimitetaan se integraattoreille. Integraatiossa prototyyppi analysoidaan ja mallinnetaan sopivalla ohjelmointikielellä dynaamisen tietojenkäsittelyn mahdollistamiseksi. Lopuksi toteutetaan ohjainkerros, joka käsittelee ja välittää pyyntöjä verkkopalvelimen ja mallin välillä. (1, s. 201–204.)

4.2 PHP-komentosarjakieli

PHP, joka on lyhennys sanoista "PHP: Hypertext Preprocessor", on laajasti käytetty avoimen lähdekoodin komentosarja- eli skriptikieli, ja se sopii erityisesti dynaamisten web-sivustojen luontiin web-palvelinympäristössä. Kyseessä on palvelinpään (server-side) ohjelmoinnissa käytettävä kieli, joka tulkitaan ohjelman suoritusvaiheessa ajonaikaisesti. Syntaksiltaan PHP muistuttaa C-, Java- ja Perl-ohjelmointikieliä. Nykyisin PHP on maailmalla käytetyin dynaamisten web-palveluiden toteutuksessa käytetty kieli. (5; 6.)

PHP-kielen kehitti alun perin Rasmus Lerdorf vuonna 1994 kirjoittamalla joukon C-kielisiä CGI-skriptejä (Common Gateway Interface). Nämä julkaistiin vuonna 1995 nimellä "Personal Home Page Tools". Kielen versio 2, joka alkoi muistuttaa ohjelmointikieltä, julkaisiin vuonna 1997. Andi Gutmans ja Zeev Suraski totesivat vuonna 1997 kielen riittämättömäksi elektroniseen kaupankäyntiin, kirjoittivat PHP-jäsentimen lähdekoodin uudestaan ja julkaisivat kielen version 3 vuonna 1998. Yksi version 3 merkittävistä uusista ominaisuuksista oli tuki olio-ohjelmoinnille. Vuonna 2000 julkaistiin PHP:n versio 4, joka käyttää Gutmansin ja Suraskin kehittämää Zend Engine -ydintä. Tämä ydin kehitettiin tukemaan aiempaa paremmin kolmansien osapuolten ohjelmointirajapintoja. Kirjoittamishetkellä käytettävän PHP:n versionumero on 5. Ytimenä on Zend Engine 2, jossa olio-ohjelmointitukea on kehitetty edelleen. (7.)

4.3 Ohjelmistokehys

Ohjelmistokehyksillä tarkoitetaan osittain abstraktiksi jätettyjä ohjelmistorunkoja, joita täydentämällä saadaan toteutettua kokonaisia uusia sovelluksia tai sovellusten osia. Näiden käytön tavoitteena on laajamittainen ja systemaattinen ohjelmakoodin ja ohjelmiston arkkitehtuuriin uudelleenkäyttö, joka edelleen nopeuttaa uusien ohjelmistojen toteutusta. Ohjelmistokehys tyypillisesti tarjoaa valmiiksi toteutettuja ohjelmistomoduuleja, jolloin kaikkia toteutettavan ohjelmiston vaatimia toiminnallisuuksia ei tarvitse ohjelmoida itse. Olioperustaisissa kehyksissä voidaan puhua myös ohjelmistorungosta, joka tarjoaa käyttäjälleen kokoelman luokkia, komponentteja ja rajapintoja.

Ohjelmistokehysten historia alkaa jo 1960-luvulta oliokieli Simula 67:n toteutuksesta. Kieli sisälsi valmiiksi määriteltyjä luokkakokoelmia, jotka voidaan tulkita ohjelmistokehyksiksi. SmallTalk-80 -ympäristön Model-View-Controller oli ensimmäinen laajasti käytetty ohjelmistokehys ja tämä aloitti ohjelmistokehysten vyöryn, joka jatkuu edelleen.

Kehysten käytön ilmeisten hyötyjen ohella on nähtävissä myös riskejä ja ongelmia, kuten se, että kehys saattaa kasvaa suureksi ja hallitsemattomaksi. Kehysten käyttö saattaa lisäksi vaikeuttaa ohjelmistojen laatuominaisuuksien variointia, kuten esimerkiksi tietoturvan lisäämisessä tai suorituskyvyn optimoinnissa. (8; 9.)

4.4 Zend Framework

Zend Framework on PHP:n kehittäjien perustaman Zend Technologies -yhtiön kehittämä avoimen lähdekoodin oliopohjainen ohjelmistokehys PHP:lle. Ohjelmistokehys tukee PHP-kieltä versiosta 5 alkaen. Ohjelmistokehys perustuu MVC-arkkitehtuuriin, ja sen avulla toteutettava sovellus jaetaan MVC-arkkitehtuurin mukaisesti loogisiin kokonaisuuksiin, joissa käyttäjänäkymät ja sovelluksen liiketoimintaosuus on eroteltu ja sijoitettu loogiseen paikkaan sovelluksessa. Zend Frameworkin käyttö selkeyttää ohjelmiston rakennetta, nopeuttaa ohjelmistokehitystä ja ohjaa suunnittelijaa noudattamaan hyviä käytäntöjä sovelluksen toteutuksessa.

Zend Framework koostuu toisiinsa löyhästi sidoksissa olevista ohjelmistokomponenteista, ts. tarjoaa luokkakirjaston sovellusten toteuttamiseksi. Komponenttien välinen

löyhä sidos mahdollistaa kehyksen komponenttien yksittäisen käytön, jolloin kehyksen tarjoamasta kirjastosta voidaan valita vain tarvittavat komponentit, mm. tietokantayhteydet, työkalut projektin rutiinitehtävien automatisointiin (hakemistorakenteen luonti, luokkien luonti), todennus ja HTTP-istunnon hallinta. BSD-lisensointi sallii ohjelmistokehyksen vapaan käytön. Ohjelmistokehittäjä voi halutessaan hyödyntää luokkakirjaston tarjoamia valmiita toiminnallisuuksia ja laajentaa niitä tarpeen mukaan. Valmiskomponenttien käyttö vähentää huomattavasti rutiiniohjelmointia sekä virheiden mahdollisuutta. (1, s. 215; 10; 11; 12.)

Zend Framework tarjoaa ohjelmakoodin kirjoittamiselle suositukset (coding standard), joita noudattamalla saadaan aikaan laadukasta ohjelmakoodia, vähemmän virheitä ja parempi ylläpidettävyys. Lisäksi sovellusten kehitystyö on helpommin hajautettavissa. Suosituksissa mainitaan mm. funktioiden ja metodien sekä muuttujien nimeäminen ja yleinen ohjelmointityyli mukaan lukien suositus ohjelmakoodin kommentointitavalle. Lisäksi kehys tarjoaa suositukset sovelluksen dokumentoinnille sekä sovelluksen hakemistorakenteelle.

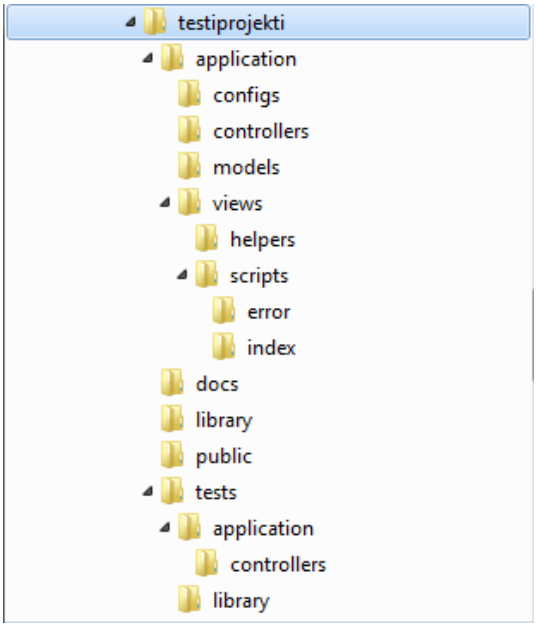
Kehys standardoi luokkien nimeämisen, jolloin luokkanimi viittaa suoraan hakemistoon, johon se on levyjärjestelmässä tallennettu. Esimerkiksi Zend_Application_Bootstrap_Bootstrap -niminen luokka viittaa tiedostoon Zend\Application\Bootstrap\Bootstrap.php. Luokkanimissä saa käyttää ainoastaan aakkosnumeerisia merkkejä. Alaviivoja käytetään luokan nimessä erottamaan hakemistot toisistaan. Zend-alkuiset luokkanimet on varattu kehyksen omille luokille. Omat luokat tulee siksi nimetä ilman sanaa "Zend" nimen alussa, esimerkiksi Application_Model_Task. Tiedostojen nimessä tulee käyttää tiedostotunnistetta .php.

Zend Framework käyttää autoload-mekanismia. Tämän avulla voidaan poistaa luokkien väliset riippuvuudet PHP-koodissa. Autoload-mekanismin ollessa käytössä, ns. "autoloader" lataa tarvittavat luokat käyttöön automaattisesti. Käytännössä ohjelmoijan ei tarvitse huolehtia PHP:n include- ja require-lauseista, joilla tarvittavia luokkia ladataan käytettäväksi, eikä siitä, missä tarvittavat luokat sijaitsevat projektissa. Autoload-mekanismilla on merkitystä myös tehokkuuden kannalta, sillä se varmistaa, että tarvittavat luokat ladataan käyttöön vain tarvittaessa ja kerran. (10.)

Zend-projekti aloitetaan luomalla Bootstrap-luokka, siihen liittyvä konfiguraatiotiedosto ja index.php -tiedosto. Näillä toteutetaan ns. bootstrapping-mekanismi, jolla sovellus

alustetaan – ladataan tarvittavat tiedostot, luetaan konfiguraatiotiedot, kuten tietokannan osoite, käyttäjätunnus ja salasana, jäsennetään URL:n tiedot ja alustetaan HTTP-pyyntö- ja -vastausoliot sekä ohjaimet. MVC-sivustot tyypillisesti toteutetaan käyttäen keskitettyä PHP-skriptiä, joka käsittelee kaikki pyynnöt sivustolla. Sen sijaan, että selaimen otsikkoriviltä kutsuttaisiin yksittäistä PHP-sivua, käytettävä osoite on muotoa host/controller/action, jolloin sovellus kutsuu annettua ohjainta ja tämän action-metodia. Käytännössä tuo keskitetty skripti sijaitsee index.php-tiedostossa, jonka suorituksessa luodaan Zend_Application-olio, joka puolestaan käynnistää bootstrapping- ja autoload-mekanismin. Bootstrapping-mekanismin, ts. sovelluksen alustustoimenpiteiden, toteutuksen jälkeen projektiin toteutetaan tarvittavat MVC-komponentit.

Projektin aloitukseen liittyvien rutiinitehtävien automatisoimiseksi voidaan käyttää ohjelmointikehyksen tarjoamaa komentorivityökalua. Työkalulla on mahdollista mm. luoda oletuskansiorakenne oletusarvoisella bootstrapping- ja autoload-mekanismilla komennolla *zf create project projektinimi*. Työkalun luoma hakemistorakenne on kuvattu kuvassa 2. Vastaavasti luokkatiedostot MVC-komponenteille voidaan luoda työkalulla komennoilla *zf create model mallinimi* ja *zf create controller ohjainnimi*. Myös ohjainluokan action-metodien esittely voidaan suorittaa työkalulla komennolla *zf create action actionnimi ohjainnimi*. Tällöin ohjelmoijan tarvitsee vain toteuttaa metodin toiminnallisuus. Ohjaimen action-metodin, esim. loginAction, luonti työkalulla luo samalla myös metodia vastaavan näkymätiedoston (login.phtml). (1, s. 215–232; 13.)



Kuva 2. Zend-sovelluksen oletushakemistorakenne.

4.5 Ketterät menetelmät

Ketterillä menetelmillä tarkoitetaan ohjelmistotuotantoprojekteissa käytettäviä menetelmiä. Niille yhteistä on inkrementaalisuus, tiivis yhteistyö asiakkaan ja kehittäjien välillä, suoraviivaisuus ja adaptiivisuus. Inkrementaalisuudella tarkoitetaan sitä, että ohjelmistosta toimitetaan pieniä julkaisuja nopeissa sykleissä. Suoraviivaisuus taas merkitsee sitä, että menetelmä itsessään on helposti opittavissa sekä muokattavissa ja lisäksi hyvin dokumentoitu. Adaptiivisuus puolestaan tarkoittaa, että menetelmä mahdollistaa nopean reagoinnin muutoksiin jopa hyvin myöhäisessä vaiheessa. (14.)

Ongelmana perinteisissä menetelmissä on ollut niiden kyvyttömyys vastata muuttuviin vaatimuksiin projektin aikana. Mitä myöhäisemmässä vaiheessa projektia vaatimuksia muutetaan, sitä enemmän muutokset tulevat maksamaan. Ketterissä menetelmissä etuna on kyky reagoida muutoksiin myös aivan projektin loppuvaiheessa. Toimivia versioita kehitettävästä ohjelmistosta toimitetaan asiakkaalle usein, vähintään muutaman kuukauden välein, ja asiakas tekee koko projektin ajan yhteistyötä kehittäjien kanssa. Siten asiakas pysyy ajan tasalla projektin kulusta. Kehittäjät taas saavat asiakkaalta välitöntä palautetta toteutettavasta sovelluksesta. Perinteisissä menetelmissä taas asiakas pääsee yleensä näkemään lopputuloksen vasta projektin loppuvaiheessa. Jos vaatimusmäärittely on puutteellinen, lopputulos ei välttämättä vastaa tarpeeksi hyvin asiakkaan tarpeita. Perinteisissä menetelmissä dokumentoinnilla on ollut tärkeä osa, ja dokumentteja tuotetaankin koko projektin ajan. Ketterissä menetelmissä taas toimivaa ohjelmakoodia arvostetaan enemmän kuin kattavaa projektidokumentaatiota. (15.)

4.6 Scrum

Scrum on ketterä menetelmä, joka muiden ketterien menetelmien tapaan näkee ohjelmistokehityksen rakentuvan erimittaisten syklien ympärille. Scrumissa tärkeimmät syklit ovat sprintti ja päivä. Sprintillä tarkoitetaan kehitysjaksoa, jonka jälkeen tuote tai sen osa on ainakin periaatteessa julkaisuvalmis. Sprintin tyypillinen kesto on kuukausi, mutta sen pituus voi vaihdella organisaation tarpeiden mukaan viikosta kahteen kuukauteen.

Perinteisessä vesiputousmallin mukaisessa projektissa on yleensä ainakin määrittelijä, suunnittelija, ohjelmoija, testaaja ja projektipäällikkö. Kussakin roolissa voi projektipääl-

likköä lukuun ottamatta olla useampia henkilöitä. Samoin yksi henkilö voi joissain tapauksissa kuulua useampaankin rooliin. Scrum-projektissa sen sijaan esiintyy vain kolme eri roolia: tuotteen omistaja, Scrum-mestari ja tiimi.

Tuotteen omistaja (product owner) “omistaa” tuotteen, ts. vastaa viime kädessä tuotteen ominaisuuksista. Asiakasprojekteissa omistaja on asiakkaan edustaja, tuotekehitysprojekteissa taas tyypillisesti tuotepäällikkö. Omistajan tehtävänä on tehdä kaikki päätökset tuotteen ominaisuuksista ja toiminnallisuuksista. Asiakkaan organisaatiossa projektin omistajan ja projektipäällikön tulisi olla sama henkilö. Jos tämä ei ole mahdollista, niin asiakkaan puolella projektipäällikkönä toimivalle tulisi ainakin voida antaa selkeät toimintavaltuudet tehdä päätöksiä.

Scrum-mestari (Scrum master) huolehtii siitä, että tiimi voi tehdä työtään optimaalisella tavalla. Mestarin tehtävänä on ratkoa ongelmat, jotka hidastavat töiden etenemistä. Lisäksi mestari järjestää päivittäiset aamupalaverit ja vastaa siitä, että Scrumia noudatetaan oikein.

Tiimii koostuu kaikista niistä henkilöistä, joilla on tarvittava osaaminen projektin kannalta. Tiimi yhdessä rakentaa tuotteen, millä halutaan korostaa kunkin tiimiläisen olevan projektin kannalta yhtä tärkeä. Tiimi yhdessä vastaa tuotteen kaikista puolista, ei koskaan yksittäinen henkilö. Tiimi vastaa itse yhteisöllisesti tehtävien jaosta, jolloin tehtävät jakautuvat tiimin jäsenille osaamisen mukaisesti.

Aluksi Scrumissa luodaan korkean tason visio siitä, mitä tulevalta projektilta halutaan. Visio vastaa kysymyksiin ”Miksi projekti tehdään?” ja ”Mitä projektin lopputuotteena saadaan?”. Tämän jälkeen muodostetaan alustava tuotteen työlista (product backlog), eli lista tuotteen ominaisuuksista. Tuotteen omistajan on priorisoitava tuotteen ominaisuudet ennen toteutuksen aloittamista.

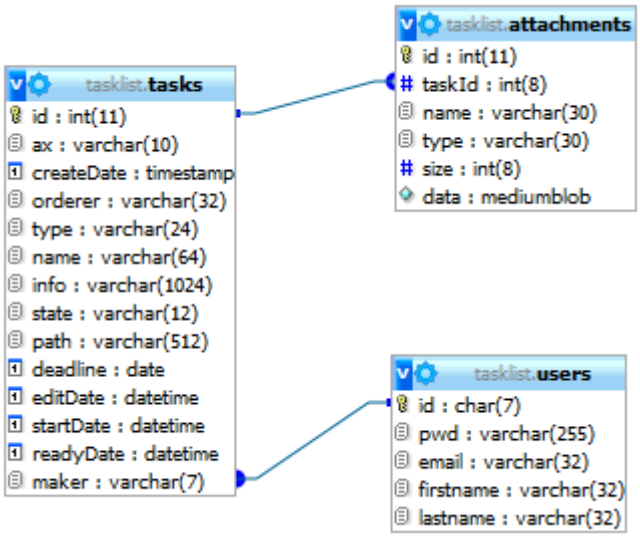
Sprintin suunnittelukokous (sprint planning) on työpaja, jossa valitaan tuotteen työlistasta seuraavan sprintin aikana toteutettavat toiminnallisuudet. Kokoukseen osallistuvat Scrum-mestari, tuotteen omistaja ja tiimi. Työpajan lopputuotteena on seuraavan sprintin työlista (sprint backlog).

Sprintin aikana tiimi toteuttaa sprinttiin kuuluviksi valittuja toiminnallisuuksia. Päivittäin tiimi kokoontuu lyhyeen tilannepalaveriin (daily scrum), jossa saadaan tietoa siitä, mi-

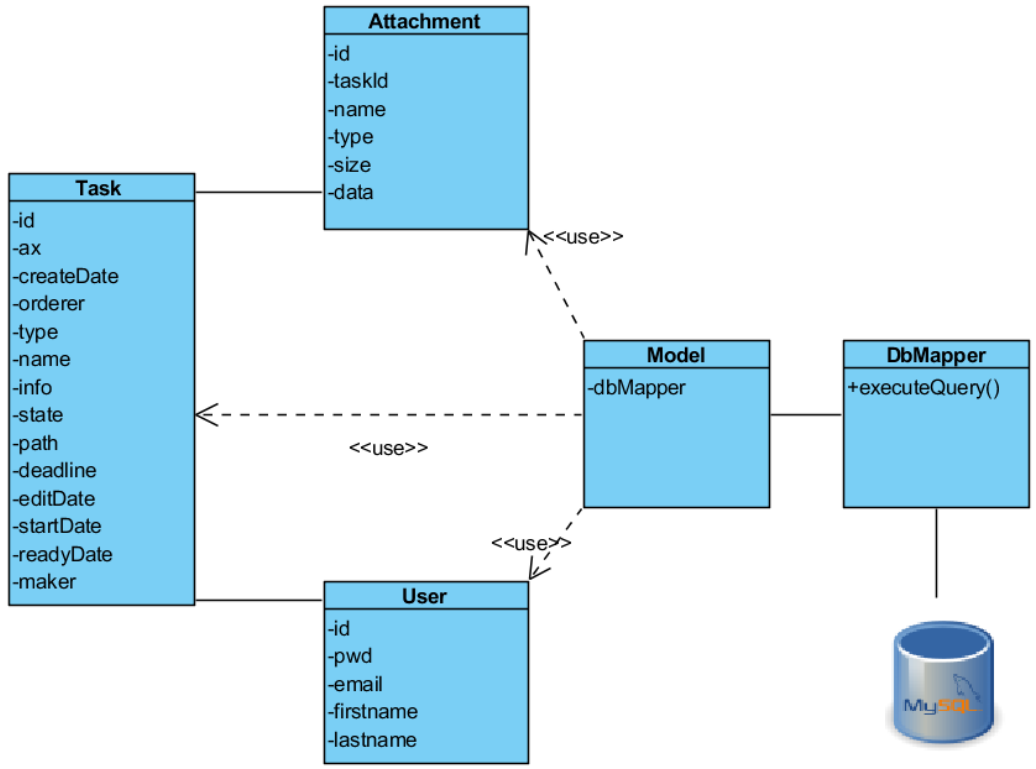
ten sprintti etenee. Sprintin lopuksi pidetään katselmointi (sprint review), jossa tiimi esittelee valmista tuotetta tuotteen omistajalle. Sprintin lopuksi tuotteen siis pitäisi olla periaatteessa käyttöönotettavissa: tuote on toteutettu, testattu ja dokumentoitu. Omistaja voi katselmoinnin perusteella päättää joko seuraavan sprintin tekemisestä tai tuotteen käyttöönottamisesta. Lisäksi kunkin sprintin lopuksi tiimi tarkastelee päättynyttä sprinttiä (sprint retrospective). Tiimin jäsenet kertovat omasta näkökulmastaan, mikä sprintissä meni hyvin ja mitä voisi olla parannettavissa. (16; 17.)

5 Insinöörityön toteutus

Sprintin pituudeksi asetettiin kaksi viikkoa. Ensimmäisen sprintin työlistalle valittiin tietokannan taulujen rakentaminen (kuva 3) sekä sovelluksen liiketoimintalogiikan ohjelmointi Model-, Task- ja DbMapper-luokkien osalta (kuva 4). Task-luokka on luokka sovellukseen tallennettaville tehtäville. Model-luokka sisältää metodit tehtävätietueiden hakemiseen tietokannasta, tehtävätietueen lisäämisen ja päivittämisen tietokantaan, tehtävän aikaleimojen ja tilan asettamisen. DbMapper-luokka huolehtii tietokantayhteyden luomisesta ja sille annetun SQL-lauseen suorituksesta. Luokat on testattu kutsumalla kutakin luokan metodia.



Kuva 3. Sovelluksen tietokantarakenne.



Kuva 4. Sovelluksen luokkakaavio.

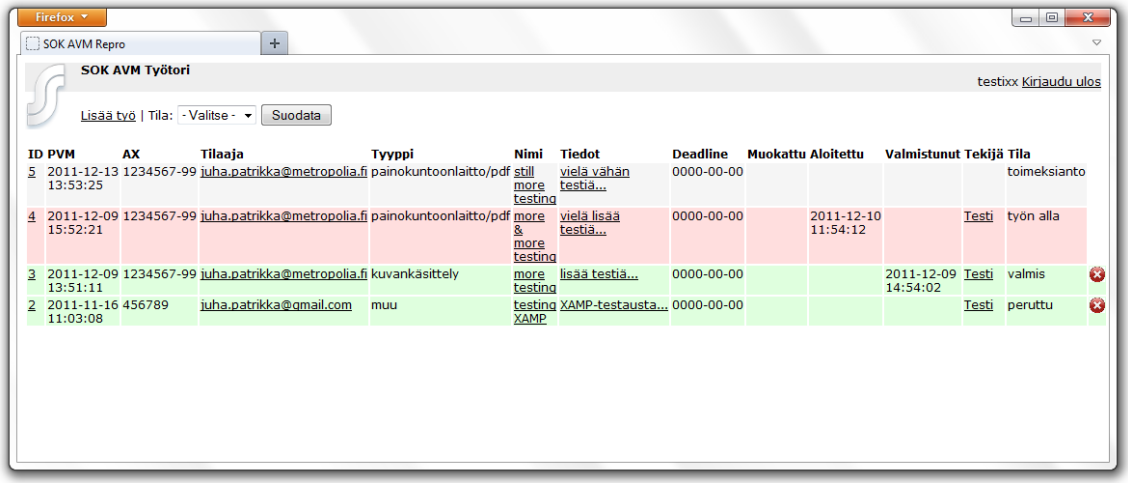
Toisessa sprintissä toteutettiin jo jotain tilaajalle näkyvää ensimmäisine käyttöliittymineen. Task-luokan ilmentymien hakemiseksi tietokannasta ja tallettamiseksi tietokantaan toteutettiin ohjaimet (controller) sovelluksen etusivua, tehtävän lisäyslomaketta, tehtävän tietosivua ja tehtävän tilan muuttamista varten.

Käyttöoikeuksien hallinnasta vastaava ohjain, samoin kuin tarvittavat lisäykset käyttöoikeuksien tarkistamiseksi edellisessä sprintissä toteutettuihin ohjaimiin, toteutettiin kolmannessa sprintissä. Lisäksi kolmannessa sprintissä päätettiin ja toteutettiin sovelluksen lopullinen ulkoasu.

Neljännessä sprintissä toteutettiin sovellukseen tallennettavia liitetiedostoja varten Attachment-luokka ja tätä vastaava ohjain. Ohjain huolehtii liitetiedoston latauksesta käyttäjän työasemalle sekä liitetiedoston poistosta tietokannasta. Liitetiedoston lisäys tehtävän lisäyksen tai päivityksen yhteydessä toteutettiin Task-luokan kontrolleriin. Projektin viidennessä eli viimeisessä sprintissä toteutettiin Task-luokan ohjaimeen toiminnallisuus jo olemassa olevien tehtävien kopioimiseksi uudeksi tehtäväksi käyttäjän valitsemilla tiedoilla.

Sovelluksen käyttöliittymä rakentuu erilaisista näytöistä eri toimintoja varten. Kaikki toiminnot liittyvät tehtävien tietojen tarkasteluun, tehtävien lisäämiseen ja tehtävien tietojen muokkaamiseen.

Sovelluksessa käytetään ns. layout-komponenttia (\application\layouts\scripts\layout.phtml) yhdenmukaistamaan eri näyttöjä. Lisäksi layout-komponentin käyttö mahdollistaa sovelluksen personoinnin helposti. Komponentin otsikko-osassa (head-elementti) ladataan käyttöön ulkoiset JavaScript-komponentit (\public\js\), joilla tarkistetaan, että pakolliset kentät lomakkeissa on täytetty sekä päivitetään sovelluksen etusivua halutulla aikavälillä. Runko (body-elementti) koostuu otsakeosiosta ja vaihtuvasta sisältöosiosta. Otsakeosiossa on käyttöliittymän vakioelementit, jotka toistuvat jokaisella sivulla. Tilaajalle toimitetussa sovelluksessa otsakkeen vakioelementtejä ovat logo, sovelluksen nimi ja linkki kirjautumissivulle. Mikäli käyttäjä on kirjautunut, otsakkeessa näytetään käyttäjän käyttäjätunnus ja uloskirjautumislinkki. Sivuston ulkoasun tyylit on määritelty ulkoisessa CSS-tiedostossa (Cascading Style Sheets) (\public\css\tasklist.css).



Kuva 5. Sovelluksen etusivun tehtävälistaus.

Sovelluksen etusivulla (kuva 5) listataan tietokannan tehtävätietueet käyttäjän valitseman suodatuksen mukaisesti. Tehtävien suodattamiseksi käytetään tehtävien tilakenttää. Etusivulta käyttäjä pääsee lisäämään uusia tehtäviä ja siirtymään kunkin tehtävän tietosivulle tarkastelemaan ja tarvittaessa muokkaamaan tietokantaan tallennetun tehtävän tietoja. Tietosivulta pääsee myös kopioimaan haluttuja tehtävän tietoja uuden tehtävän tiedoiksi. Kirjautuneet ja tehtävien ylläpitoon oikeutetut käyttäjät saavat tehtä-

vän tietosivulla näkyviin alasvetovalikon, josta tehtävän tilaa voidaan vaihtaa. Tällöin myös sovelluksen etusivulla tehtävien poisto tietokannasta on mahdollista.

Kuten edellä jo mainittiin, DbMapper-luokka (\application\models\DbMapper.php) huolehtii tietokantayhteyden avaamisesta ja sen ainoalle metodille parametrina annetun SQL-lauseen suorituksesta. Luokassa hyödynnetään Zend Frameworkin Zend_Config_Ini-luokkaa tietokantayhteyden luomisessa tarvittavien tietojen, kuten tietokantapalvelimen osoitteen, lukemiseksi ulkoisesta konfiguraatiotiedostosta (/application/configs/config.ini).

Logiikka tietojen hakemiseksi, lisäämiseksi, päivittämiseksi ja poistamiseksi on sijoitettu Model-luokkaan (\application\models\Model.php). Ohjaimet välittävät tietokantaoperaatioita vaativia toimenpiteitä Model-luokan olion suoritettavaksi. Model-luokan olio vastaavasti kutsuu DbMapper-luokan oliota varsinaisen tietokantaoperaation suorittamiseksi.

Task (\application\models\Task.php) on luokka sovelluksessa luotaville tehtäville. Luokassa on get- ja set-metodit luokan jäsenmuuttujien hakemiseksi ja asettamiseksi. Luokan ilmentymien luonti ja käsittely tapahtuu Model-luokassa.

Attachment (\application\models\Attachment.php) on luokka liitetiedostoille, joita voidaan liittää tehtävien yhteyteen. Luokassa on get-metodi luokan jäsenmuuttujien hakemiseksi. Task-luokan tapaan Attachment-luokan ilmentymien luonti ja käsittely tapahtuu Model-luokassa.

IndexController (\application\controller\IndexController.php) toimii ohjaimena sovelluksen etusivulle. Ohjaimen indexAction-metodissa kutsutaan Model-luokan metodia, joka palauttaa halutut tehtävätietueet, ja välittää tiedot edelleen ohjaimen view-komponentille (\application\views\scripts\index\index.phtml). Lisäksi ohjaimessa tarkistetaan, löytyykö käyttäjä HTTP-istunnosta tietoa kirjautumisesta, ts. käyttäjän käyttäjätunnusta, ja mikäli tieto istunnosta löytyy, tarkistetaan, onko käyttäjätunnuksella oikeuksia tehtävien ylläpitotoimintoihin. Tällä tavoin mahdollistetaan useamman rinnakkaisen sovellusinstanssin käyttö. Käyttäjä voi esimerkiksi olla tehtävien vastaanottaja yhdessä sovellusinstanssissa ja toimeksiantaja toisessa. Myös käytetty tehtävien tilan mukainen suodatus luetaan ja kirjoitetaan istuntoon. Istunnon hallinnassa käytetään koko sovelluksessa Zend_Session_Namespace-luokkaa.

TaskController-ohjain (\application\controllers\TaskController.php) hoitaa yksittäisiin tehtäviin liittyvät toiminnot, joita ovat tehtävän tietojen listaaminen, tehtävän tietojen päivittäminen, tehtävän tilan vaihtaminen, uuden tehtävän lisääminen, tehtävän tietojen kopiointi uudelle tehtävälle ja tehtävän poistaminen. Metodi indexAction poimii HTTP-pyynnössä parametrinä välitetyn tehtävän tunnisteen eli id:n ja pyytää Model-luokan oliolta id:tä vastaavaa tehtäväoliota. Tämän jälkeen tehtäväolion tiedot välitetään view-komponentille. Lisäksi metodissa tehdään vastaavat käyttöoikeuksiin liittyvät tarkistukset kuten IndexController-ohjaimessakin tehtävän tilan vaihtamista varten.

Tehtävän tilan vaihto hoidetaan changestateAction-metodissa, jossa ensin tarkistetaan, että HTTP-pyynnössä on käytetty POST-metodia. Tällä varmistetaan, että tehtävän tilaa päivitetään vain lomakkeen kautta eikä selaimen osoiteriviltä käsin. Tilan vaihtaminen päivittää tehtävän tilan lisäksi HTTP-istunnosta käyttäjän käyttäjätunnuksen tietokantaan.

Metodeilla updateformAction ja updateAction hoidetaan tehtävän tietojen päivittäminen. updateformAction-metodissa välitetään tehtävän tiedot HTML-lomakkeelle ja updateAction käsittelee lomakkeelta lähetetyt tiedot ja välittää ne edelleen Model-luokan oliolle tietokantapäivitystä varten. Kuten changestateAction-metodissa, myös updateAction-metodissa tarkistetaan HTTP-pyynnössä käytetty metodi.

Uusi tehtävä lisätään insertformAction- ja insertAction-metodeilla, josta ensimmäinen päättelee lisäyslomakkeelle tultaessa käytetyn HTTP-metodin avulla, käytetäänkö valitun tehtävän haluttuja tietoja uuden tehtävän tietoina vai annetaanko käyttäjälle täysin tyhjä lomake täytettäväksi. Metodilla voidaan hoitaa siis sekä uuden tehtävän lisäys että tehtävän kopiointi uudeksi tehtäväksi. Mikäli pyynnön metodi on POST, täydennetään tehtävän lisäyslomake pyynnön parametreina annetuilla tiedoilla, muussa tapauksessa lomake on tyhjä. Lomakkeella syötetyt tiedot käsitellään pyynnöstä ja tallennetaan insertAction-metodissa Model-luokan olion avulla. Myös tässä metodissa tarkistetaan, että pyynnössä on käytetty POST-metodia väärinkäytösten estämiseksi.

Tehtävien poistaminen hoidetaan deleteAction-metodissa. Väärinkäytösten ehkäisemiseksi metodissa tarkistetaan käyttäjän käyttöoikeudet ennen kuin poisto annetaan Model-luokan olion suoritettavaksi.

Käyttäjien todennus on hoidettu UserController-ohjaimessa. loginformAction-metodilla käyttäjälle tarjotaan kirjautumislomake, jonka tiedot lähetään authAction-metodille tarkistettavaksi. Tässä metodissa todennuksessa käytetään apuna Zend Frameworkin valmiita Zend_Auth-, Zend_Db_Adapter_Pdo_Mysql- ja Zend_Auth_Adapter_DbTable-luokkia, jolloin todennuksen toteuttaminen on ollut helppoa. Todennus on kuvattu esimerkkikoodissa 1.

```
$auth = Zend_Auth::getInstance();
$db = new Zend_Db_Adapter_Pdo_Mysql($params);
$authAdapter = new Zend_Auth_Adapter_DbTable($db);
$authAdapter->setTableName('users')
                      ->setIdentityColumn('id')
                      ->setCredentialColumn('pwd');

//kirjautumistietojen käsittely
$user = $request->getParam('user');
$pwd = $request->getParam('pwd');
$authAdapter->setIdentity($user);
$authAdapter->setCredential(md5($pwd));


//autentikointikysely
$result = $auth->authenticate($authAdapter);
if($result->isValid()){
          ...
//jos validi käyttäjä, kirjoitetaan käyttäjätunnus sessioon
...
} else {
          ...
//muussa tapauksessa ohjataan takaisin kirjautumislomakkeelle
...
}
```

Esimerkkikoodi 1. Ote laaditusta koodista.

FileController-ohjain huolehtii liitetiedostojen lataamisesta käyttäjän työasemalle sekä liitetiedostojen poistosta. Tiedoston latauksessa ohjain pyytää Model-luokan oliolta haluttua liitetiedostoa ja välittää sen näyttökomponentilleen, jossa tiedoston tiedot, mm. tiedostokoko ja MIME-tyyppi, sijoitetaan HTTP-vastauksen otsaketietohin (header).

Tämän jälkeen tulostetaan liitetiedoston sisältö sivulle, jolloin internet-selain käynnistää tiedoston latauksen käyttäjän työasemalle. Liitetiedoston poistossa ohjain kutsuu Model-luokan metodia liitetiedoston poistamiseksi. Muista liitetiedostoihin liittyvistä toimenpiteistä poiketen liitetiedostojen liittäminen tehtävien yhteyteen on toteutettu TaskControllerin insertAction- ja updateAction-metodeissa.

Sovelluksen konfiguraatio on määritetty kahdessa konfiguraatiotiedostossa. Config.ini-tiedosto (\application\configs\config.ini) on sovelluksen ylläpitotoimenpiteitä varten tehty tiedosto, jossa määritetään internet-selaimen otsikkokentässä näytettävä nimi sovellukselle, käytettävän tietokannan osoite, tietokantaan kirjautumisessa käytettävä käyttäjätunnus ja salasana, tietokannan nimi, liitetiedostojen maksimitiedostokoko sekä sovelluksen tehtävälistauksen automaattisen päivittämisen aikaväli käyttöliittymässä. Varsinainen sovelluksen konfiguraatio, esimerkiksi luokkakirjastojen sijainnin määritys, on määritetty application.ini-tiedostossa (\application\configs\application.ini) oletusarvoisesti Zend Frameworkin mukaan.

6 Projektin tulokset ja yhteenveto

Sovellusprojektin läpivienti onnistui suunnitellussa ajassa johtuen täsmällisestä vaatimusmäärittelystä sekä kunkin sprintin huolellisesta suunnittelusta. Scrum-menetelmän täsmällinen noudattaminen, kuten päivittäisten tilannepalavereiden pitäminen, ei kuitenkaan tässä sovellusprojektissa olisi ollut tiimin koon vuoksi (yksi henkilö) järkevää. Scrum tarjosi kuitenkin mielestäni hyvän ja tehokkaan tavan suunnitella sovelluksen toteuttamista, koska toteutus saatiin näin jaettua loogisiin toiminnallisiin kokonaisuuksiin.

Sovelluksen vaatimuksiin ei tullut merkittäviä muutoksia tai lisäyksiä, jotka olisivat aiheuttaneet lisätyötä sovelluksen toteutuksessa. Työssä käytetty ohjelmistokehys nopeutti sovelluksen toteutusta, sillä sovelluksessa pystyttiin hyödyntämään paljon ohjelmistokehyksen tarjoamia valmiita luokkia. Samalla myös voidaan olettaa virheiden määrän vähentyneen merkittävästi.

Sovelluksen hyväksymistestauksessa ei havaittu sovellusvirheitä. Sovellus saatiin siten otettua käyttöön pian tuotantoympäristön konfiguraatiotöiden jälkeen tuotantotiimille

sopivana ajankohtana. Tuotantotiimi vastasi itse oman henkilöstönsä sekä toimeksiantajien opastuksesta sovelluksen käytössä.

Kuukausi käyttöönoton jälkeen tehtyjen laskelmien mukaan sovelluksen kautta tulleiden toimeksiantojen määrä oli 3,3-kertainen verrattuna edellisen vuoden kyseisen kuukauden vastaavaan määrään kokonaistuotantomäärien ollessa kutakuinkin samoja. Tämä merkitsee sitä, että töiden resursointiin käytetty työ on vastaavasti vähentynyt. Tällä on vaikutusta myös töiden läpimenoaikoihin, koska tiimi paremmin itseohjautuvana itse resursoi työt eivätkä ne jää työnjohdon resursoitaviksi.

Sovelluksesta saatu palaute toimeksiantajilta ja muulta organisaatiolta on ollut positiivista. Erityisesti tehtävien kopiointitoiminnallisuus on saanut kiitosta osakseen, sillä sen on koettu vähentävän toimeksiannoissa käsin syötettävän tiedon määrää. Muutama kuukausi käyttöönoton jälkeen myös yksikön toinen tiimi otti sovelluksen omaan käyttöönsä toimeksiantojen vastaanotossa ja hallinnoinnissa. Koska sovellus on toteutettu Model-View-Controller -arkkitehtuurin mukaisesti, sovelluksen käyttöliittymän räätälöinti toisen tiimin tarpeisiin on ollut helppoa. Sovellus- ja liiketoimintalogiikkaan muutoksia ei ole tarvinnut tehdä.

Käytön myötä sovellukselle on esitetty muutamia jatkokehitystoiveita liittyen sovelluksen käyttöliittymään. Toimeksiantojen määrän kasvaessa on toivottu, että toimeksiantaja voisi suodattaa tehtävälistauksesta vain omat toimeksiantonsa. Muut toiveet ovat liittyneet tehtävien lisäys- ja muokkauslomakkeisiin, joiden tekstikenttiin on toivottu merkkilaskuria ilmaisemaan käytössä olevaa merkkimäärää, koska tietokannassa tekstikenttien maksimimerkkimäärä on rajoitettu. Lisäksi lomakkeilla oleva "Työn tiedot" -kenttä toivottiin muutettavaksi alasvetovalikosta monivalinnaksi. Käytännössä muutostoiveet ovat toistaiseksi olleet varsin pieniä ja toteutettavissa pääosin näyttökomponentin muutoksilla.

Lähteet

1 McArthur, Kevin. 2008. Pro PHP: Patterns, Frameworks, Testing and More. New York City, USA: Apress.

2 Reenskaug, Trygve. 2003. The Model-View-Controller (MVC), Its Past and Present. Verkkodokumentti. Trygve M. H. Reenskaug. <http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf>. 20.8.2003. Luettu 10.2.2012.

3 Reenskaug, Trygve. 1979. Models - Views – Controllers. Verkkodokumentti. Trygve M. H. Reenskaug . <http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf>. 10.12.1979. Luettu 10.2.2012.

4 Sun Microsystems. Designing enterprise applications. Verkkodokumentti. Oracle Corporation. <http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html#1105855>. Luettu 10.2.2012.

5 The PHP Documentation Group. 2011. PHP manual. Verkkodokumentti. The PHP Group. <http://php.net/manual/en/>. 19.10.2011. Luettu 9.2.2012.

6 W3 Technology Surveys. Usage of server-side programming languages for websites. Verkkodokumentti. Q-Success. <http://w3techs.com/technologies/overview/programming_language/all>. Luettu 9.2.2012.

7 History of PHP and Related Projects. 2012. Verkkodokumentti. The PHP Group. <http://php.net/history>. Luettu 9.2.2012.

8 Ohjelmistokehys. 2012. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Ohjelmistokehys>. Luettu 4.6.2012.

9 Laine, Harri. 2010. Ohjelmistokehykset. Verkkodokumentti. Harri Laine, University of Helsinki, Department of Computer Science. <http://www.cs.helsinki.fi/u/laine/arkki/s09/pdf/oa-11-ohjelmistokehykset.pdf >. 12.5.2010. Luettu 4.6.2012.

10 Zend Technologies Inc. 2012. Programmer's Reference Guide. Verkkodokumentti. Zend Technologies Ltd. <http://framework.zend.com/manual/en/>. Luettu 10.2.2012.

11 BSD-lisenssi. 2011. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/BSD-lisenssi>. Luettu 10.2.2012.

12 Zend Framework. 2012. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Zend_Framework>. Luettu 10.2.2012.

13 Zend Technologies Ltd. 2012. Introduction. Verkkodokumentti. Zend Technologies Ltd. <http://framework.zend.com/manual/1.11/en/zend.application.introduction.html>. Luettu 10.2.2012.

14 Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. VTT Publications 478, 2002.

15 Kosonen, Sami. 2005. Ohjelmoinnin opetus Extreme Programming -hengessä, Tietotekniikan pro gradu -tutkielma. Jyväskylä: Jyväskylän yliopisto, Tietotekniikan laitos.

16 Sininen Meteoriitti. 2011. Ketteryys haltuun: Scrum pähkinänkuoressa. Verkkodokumentti. Sininen Meteoriitti Oy. <http://www.meteoriitti.com/fi-FI/tiedotteet/ajankohtaista/ketteryys-haltuun-scrum-pahkinankuoressa>. 9.12.2011. Luettu 9.6.2012.

17 Scrum Alliance. 2012. Scrum Principles. Verkkodokumentti. Scrum Alliance, Inc. <http://www.scrumalliance.org/pages/scrum_101>. Luettu 9.6.2012.

Esitutkimus

1. TUOTEIDEA

SOK Asiakkuus-, viestintä- ja markkinointipalvelut –yksikön sisäisessä tukitiimissä (Mainonnan tuotannon tuki ja kehitys) on tarvetta sähköiselle selainkäyttöiselle toimeksiantojärjestelmälle. Tukitiimi tuottaa tuotantopalveluita yksikön toisille tiimeille. Käytännössä tuotantopalvelut ovat mainonnantuotannon prosesseihin liittyviä tuotantotöitä kuten kuvankäsittelypalveluita, painoaineiston tuotantoa, web-tuotantoa tms. Toimeksiantojärjestelmän avulla tiimille voidaan tehdä toimeksiantoja, jotka tallentuvat järjestelmään. Sekä toimeksiantaja että toimeksiannon vastaanottava osapuoli näkee järjestelmän avulla toimeksiannon tilan. Järjestelmä toimii toimeksiannon vastaanottavalla osapuolella myös työlistana, josta vapaana oleva tuotantohenkilö poimii itselleen toimeksiannon suoritettavaksi. Näin järjestelmä toimii myös resursoinnin apuvälineenä (=tuotannon itseohjautuvuus).

2. PROJEKTIN ORGANISOINTI

Projektiryhmänä sekä asiakkaana toimii Mainonnan tuotannon tuki ja kehitys –tiimin tuotantoesimies Petteri Paunila, asiakkuuspäällikkö Mia Halonen sekä tuotantopäällikkö Juha Patrikka, joka myös vastaa projektista sekä projektissa kehitettävän sovelluksen teknisestä toteutuksesta. Projektin sidosryhmiä ovat Mainonnan tuotannon tuki ja kehitys –tiimin tuotantohenkilöstö sekä yksikön muut tiimin tukipalveluita tarvitsevat tiimit.

3. NYKYINEN JÄRJESTELMÄ

Tällä hetkellä Mainonnan tuotannon tuki ja kehitys –tiimille toimeksiannot tehdään PDF-lomakkeella, joka lähetetään sähköpostitse tuotannon yhteiskäyttöiseen sähköpostiosoitteeseen. Tiimin tuotantohenkilö poimii toimeksiannon (PDF-lomakkeella liitteistetty sähköposti) suoritettavaksi ja lopuksi merkitsee toimeksiannon valmiiksi sen valmistuttua käytettävän sähköposti-clientin omilla työkaluilla. Lisäksi tuotantohenkilö kuittaa toimeksiannon suoritetuksi vastaamalla lähetettyyn sähköpostiin. Käytännössä tuotantohenkilö toimeksiannon poimiessaan siirtää liitteistetyn sähköpostin omaan kansioonsa, jotta muut tuotantohenkilöt eivät poimi samaa toimeksiantoa itselleen. Nykyisen järjestelmän ongelmia ovat mm. yhteiskäyttöisen sähköpostilaatikon viestien päivittyminen sähköposti-clienteissa sekä se, ettei toimeksiantaja näe toimeksiantonsa tilaa, ts. onko toimeksianto otettu tehtäväksi.

4. HAVAITUT ONGELMAT JA RISKIT

Projektilla ei havaittu olevan riskejä.

5. TAVOITTEET JA VAATIMUKSET

Sovelluksella voidaan hallinnoida tiimille tehtäviä toimeksiantoja. Sovelluksen perustoimintoja ovat toimeksiannon (myöhemmin käytetään termiä ”tehtävä”) lisäys järjestelmään soveltuvilla lisätiedoilla sekä jo olemassa olevan tehtävän kopiointi uudeksi tehtäväksi, tehtävän tilan (toimeksianto, työn alla, valmis, peruttu) muuttaminen, tehtävien listaus käyttöliittymässä ja suodattaminen tilan mukaisesti sekä tehtävän poisto. Sovellukseen tulee voida kirjautua, mutta käyttö ei edellytä kirjautumista. Kirjautuneilla käyttäjillä on käytössään toimintoja tehtävien hallinnointiin. Ei-kirjautuneet käyttäjät voivat lisätä tehtäviä, mutta eivät pysty muuttamaan tehtävän tilaa tai poistamaan tehtävää. Tietojen tulee olla pysyviä (=tietokantaratkaisu). Sovellus on internet-selainkäyttöinen (IE, Firefox ja Safari) ja tulee olla käytettävissä sekä PC- että Mac-työasemalla.

6. RAJAUKSET

Sovellusta voidaan käyttää vain organisaation sisäverkossa.

7. YMPÄRISTÖ JA LIITTYMÄT

Projektissa toteutettavaa sovellusta varten tarvitaan organisaation sisäverkkoon web-palvelin sekä tietokantapalvelin. Palvelinohjelmistot voivat olla asennettuna samassa palvelinlaitteistossa kuten myös sovelluksen testiympäristö.

8. HYÖDYT

Projekti toteutetaan opinnäytetyönä. Tarvittavat määritykset tehdään tilaajan toimesta normaalin työn ohessa. Projektilla ei ole tiukkaa aikataulua, koska tilaajalla nykyisen toimintamallin käyttö on mahdollista projektin aikana.

Projektissa toteutettavalla sovelluksella saadaan edistettyä tiimin itseohjautuvuutta sekä vähennettyä huomattavasti päivittäisen tuotantotyön organisointiin/resursointiin liittyvää esimies/suunnittelutyötä.

9. AIKATAULU

Projekti toteutetaan 08 – 12/2011 aikavälillä ja valmistuu 31.12.2011 mennessä.

10. KUSTANNUKSET

Projektista ei aiheudu normaalin työajankäytön lisäksi muita henkilöstökustannuksia. Sovelluksen toteutuksessa pyritään hyödyntämään vapaan lähdekoodin ohjelmistoja (Apache –web-palvelin, MySQL-tietokanta) mikäli tilaajan puolesta mahdollista. Mahdollisia lisäkustannuksia sen sijaan aiheutuu tarvittavan palvelinlaitteiston hankinnasta.

11. TOTEUTUSVÄLINEET

Projektissa toteutettava sovellus olisi mahdollista toteuttaa esimerkiksi PHP-ohjelmointikielellä. Sovelluksen tietokantana voisi toimia MySQL.

12. PROJEKTIN KANNATTAVUUS

Kts. luvut 8 ja 10.

Toiminnallinen määrittely

1. JOHDANTO

1.1 Tarkoitus ja kattavuus

Tässä dokumentissa määritellään Toimeksiantojärjestelmä-projektissa toteutettavan sovelluksen toiminnalliset vaatimukset. Dokumentti on tarkoitettu koko projektiryhmälle. Sovelluksen käyttöliittymä on kuvattu alustavassa käyttöohjeessa.

1.2 Tuote

Toimeksiantojärjestelmä (käytetään myös nimeä ”Työtori”) on sisäverkossa toimiva sovellus toimeksiantojen hallinnointiin. Sovelluksen avulla voidaan vähentää merkittävästi päivittäisten tuotantotöiden organisointiin ja resursointiin liittyvää esimies/suunnittelutyötä edistämällä tuotannon itseohjautuvuutta. Sovellus toimii sekä PC- että Mac-työasemilla tämän dokumentin hyväksymisajankohtana käytetyissä IE-, Firefox- ja Safari –internet-selaimissa.

1.3 Määritelmät, termit ja lyhenteet

Käyttäjien syötteet, muut tietokantaan tallennettava merkkijonot ja painikkeidet nimet/tekstit on merkitty dokumenttiin lainausmerkkien sisällä, esim. ”Tallenna”-painike, ”Valmis”-tila, jne.

1.4 Viitteet

Ei merkitystä tässä dokumentissa.

1.5 Yleiskatsaus dokumenttiin

Luku 2 kuvaa järjestelmän toiminnan yleisellä tasolla: siihen kuuluvan laitteiston, käyttäjät, järjestelmän riippuvuudet ja rajoitukset.

Luvussa 3 kuvataan järjestelmän tietosisältö eli tietokanta ja tietovirrat.

Luvussa 4 määritellään järjestelmän toiminnot. Kustakin toiminnosta on kuvattu mitä se tarkoittaa, mitä se saa syötteeksensä ja toiminnon suorittamisesta tapahtuvat toiminnot ja/tai vaikutukset.
Luku 5 kertoo järjestelmän ulkoiset liittymät, eli laitteiston, tietoliikenteen ja ohjelmistoliittymät.

Lukuun 6 on kuvattu järjestelmän ei-toiminnalliset ominaisuudet, kuten suorituskyky, vasteajat, käytettävyys ja ylläpidettävyys.

Lukuun 7 on kirjattu suunnitteluun vaikuttavat rajoitteet, kuten standardit

sekä ohjelmisto- ja laitteistorajoitteet.

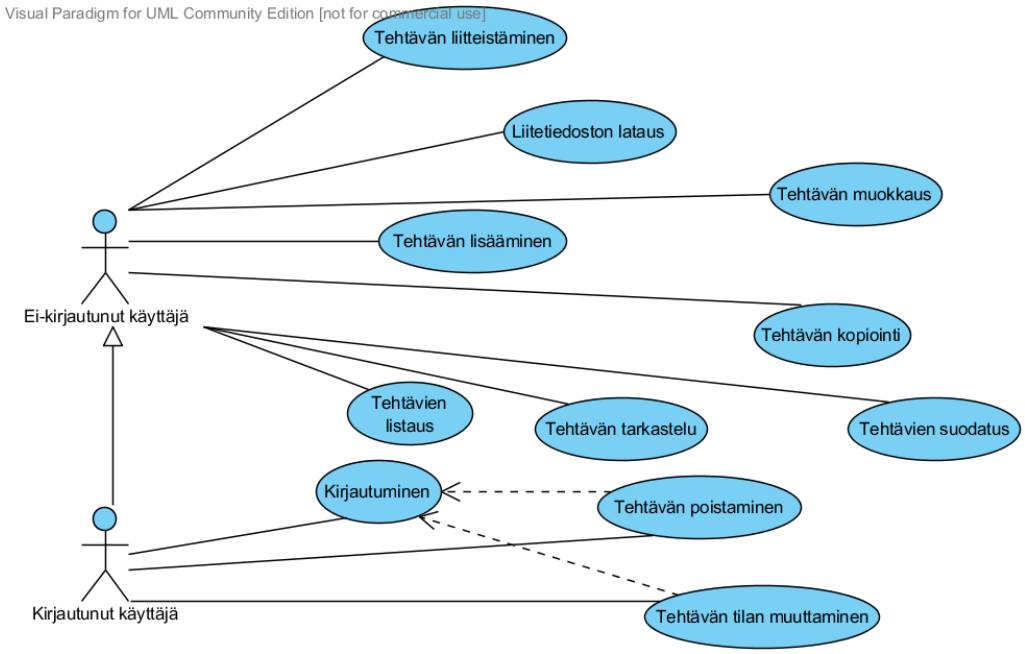
Luvussa 8 esitellään vaihtoehdot, jotka ovat olleet esillä, mutta on syystä tai toisesta hylätty.

Luku 9 on varattu jatkokehitysajatuksille.

Lukuun 10 on kerätty avoimia kohtia, ts. asioita, jotka ovat jääneet selvittämättä.

2. YLEISKUVAUS

Toimeksiantojärjestelmällä voidaan tallentaa toimeksiannettuja tehtäviä tarvittavine tietoineen sekä hallinnoida tehtäviä, niiden tietoja sekä tilaa. Järjestelmä koostuu web-palvelimesta sekä tietokantapalvelimesta.



Kuva 2.1: Käyttötapaukset

2.1 Ympäristö

Sovellusta käytetään sekä PC- että Mac-työasemilla internet-selaimilla (IE, Firefox ja Safari). Järjestelmän palvelinlaitteistona käytetään Mac XServe-palvelinta, jossa palvelinohjelmistona Mac OSX Server –käyttöjärjestelmä, webpalvelimena toimii Apache ja tietokantana MySQL. Kaikki palvelinohjelmistot, kuten myös sovelluksen testiympäristö, sijaitsevat samassa palvelinlaitteessa. Sovellus toimii omana erillisenä kokonaisuutenaan, joten integraatioita muihin järjestelmiin ei tarvita.

2.2 Toiminta

Sovelluksen käyttöliittymässä voidaan lisätä toimeksiantoja (myöhemmin käytetään termiä ”tehtävä”) tietokantaan tarvittavilla tiedoilla, hallinnoida niiden tietoja sekä tilaa ja poistaa tehtäviä tietokannasta. Käyttäjät syöttävät tiedot lomakkeille, joista ne tallennetaan/päivitetään tietokantaan. Käyttöliittymässä listataan tietokantaan tallennetut tehtävät olennaisilla tiedoilla kuten tilaaja, tehtävän nimi ja kuvaus, tehtävän deadline sekä tehtävän tila. Tehtävälistausta on mahdollista suodattaa tehtävien tilan mukaisesti. Käyttöliittymän tehtävälistauksen päivittäminen ei vaadi käyttäjältä toimenpiteitä vaan tehdään automaattisesti määräajoin.

2.3 Käyttäjät

Järjestelmällä on useita kymmeniä (<100) samanaikaisia käyttäjiä, jotka käyttävät sovellusta samanaikaisesti internet-selaimella. Käytännössä käyttäjä voi pitää sovellusikkunaa auki internet-selaimessaan läpi työajan. Sovellusta käytetään sekä kirjautuneena että ei-kirjautuneena käyttäjänä. Kirjautuneilla käyttäjillä on laajemmat oikeudet päivittää tietokantaa kuin ei-kirjautuneilla käyttäjillä. Käytännössä ei-kirjautuneet käyttäjät palveluita tarvitessaan tallentavat tehtäviä järjestelmään ja kirjautuneet käyttäjät vastaanottavat niitä suoritettavaksi. Järjestelmällä on ylläpitokäyttäjä/käyttäjiä, jotka ylläpitävät palvelinohjelmistoja, ml. tietokanta, sopivilla varusohjelmilla. Sovelluksen käyttö ei vaadi koulutusta tai erityisosaamista (pl. ylläpitäjä).

2.4 Oletukset ja rajoitteet

Järjestelmällä ei ole yhteyksiä muihin järjestelmiin, ts. käyttäjien syötteiden oikeellisuutta ei tarkisteta eikä tietoja haeta muista järjestelmistä.

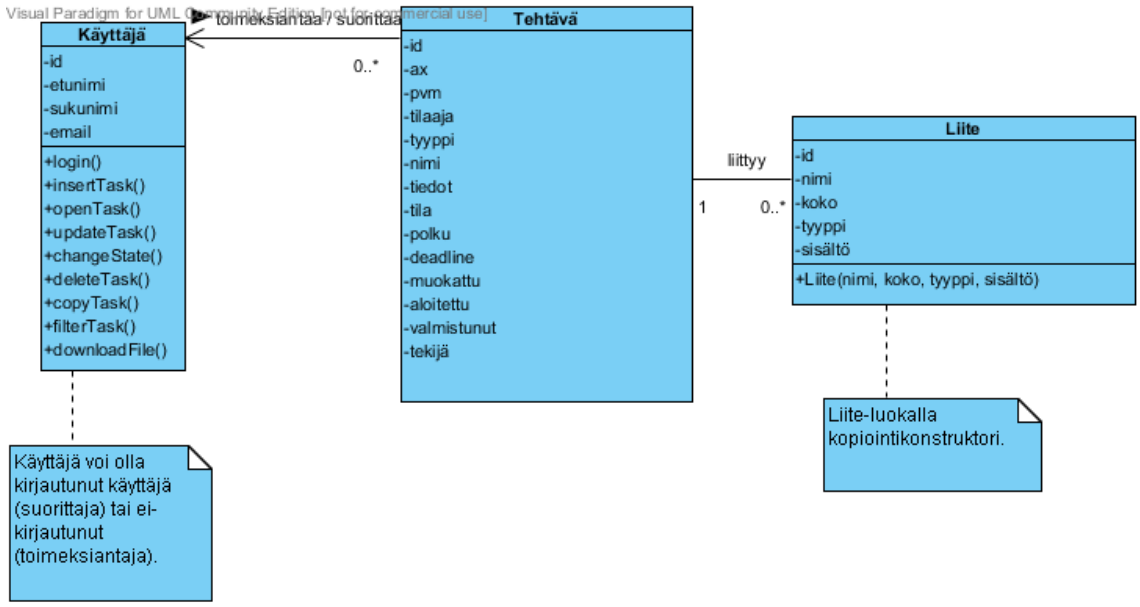
3. TIEDOT JA TIETOKANNAT

3.1 Tietosisältö

Tehtävä on synonyymi toimeksiannolle ja sisältää seuraavat tiedot: ID, AX-projektinumero, talletuspvm, tilaaja, tyyppi, nimi, tiedot/kuvaus, tila, lähdeaineiston tiedostopolku, deadline-pvm, muokkauspvm, aloituspvm, valmistumispvm ja tehtävän suorittaja. Tehtävällä voi olla mahdollisesti yksi tai useampi liitetiedosto. Liitetiedoston tiedoista järjestelmään tallennetaan nimi, tyyppi ja koko. Kirjautuvien käyttäjien tiedoista tallennetaan käyttäjätunnus, etunimi, sukunimi ja sähköpostiosoite.

Tämän dokumentin tietohakemistossa käytetään seuraavia merkintätapoja ja termejä:

Merkintä	*Selitys*
+	*Ja*
( )	*Optionaalinen (voi puuttua)*
{ }	*Toisto (0-N kertaa)*
n{ }m	*Toisto n-m kertaa*
n-m	*Väli n:stä m:ään*
[ ]	*Vaihtoehtoja*
|	*Vaihtoehtojen erotin*
@	*Avainominaisuus*
***	*Selite muodossa * teksti **
/	*Automaattisesti täytettävä tai laskettava kenttä*
M	** 8-bittisen ascii-merkistön kirjain, numero tai muu kirjoitusmerkki. **
K	*A-Ä|a-ä*
N	*0-9*
P	** Päiväys, josta pv, kk ja vuosi selviävät yksikäsitteisesti **



Kuva 3.1: Luokkakaavio

3.1.1 Tehtävä

ID	*@ / * Tehtävän id-numero, auto-increment, pakollinen tieto **
AX	*(0{M}10) * AX-projektinumero, voi olla tyhjä kenttä **
Pvm	*P * Tehtävän tallennuspäivämäärä ja kellonaika **
Tilaaja	*0{M}32 * Tehtävän tilaajan sähköpostiosoite **
Tyyppi	*0{M}24 * Tehtävän tyyppi: ”kuvankäsittely”, ”painokuntoonlaitto/pdf”, ”MAPPI-talletus”, ”siirto painoon”, ”WTOP”, ”muu toimeksianto” **
Nimi	*0{M}64 * Tehtävälle annettava kuvaava nimi **
Tiedot	*0{M}1024 * Tehtävän lisätiedot / kuvaus **
Tila	*0{M}12 * Tehtävän tila: ”toimeksianto”, ”työn alla”, ”valmis”, ”peruttu” (oletus = ”toimeksianto”) **
Polku	*0{M}512 * Tehtävän lähdeaineiston tiedostopolku, pakollinen tieto **
Deadline	*P * Päivämäärä, jolloin tehtävän on viimeistään oltava valmis **
Muokattu	*P * Tehtävän tietojen muokkauspäivämäärä ja kellonaika **

Aloitettu	*P * Tehtävän aloitusajankohta, kun tehtävän tila vaihtuu ”toimeksisanto” -> ”työn alla”, ei muissa tilavaihdoksissa ****
Valmistunut	*P * Tehtävän valmistumisajankohta, kun tehtävän tila vaihtuu ”valmis”, viimeisin tilavaihto tallennetaan ****
Tekijä	*(0{M}7) * Tehtävän muokkaajan käyttäjätunnus, muokkaus = tilan muuttaminen, oletusarvoisesti tyhjä ****
Liite	*** Tehtävään mahdollisesti liittyvä tiedosto ****

3.1.2 Liite

ID	*@ / * Liitetiedoston id-numero, auto-increment, pakollinen tieto ****
Nimi	*0{M}30 * Liitetiedoston tiedostonimi ****
Koko	*0{N}8 * Liitetiedoston tiedostokoko ****
Tyyppi	*0{M}30 * Liitetiedoston tiedostotyyppi ****
Sisältö	*** Liitetiedoston data ****

3.1.3 Käyttäjä

ID	*0{M}7 * Käyttäjätunnus ****
Pwd	*0{M}255 * Salasana, kryptattu MD5-tiivistefunktiolla ****
Etunimi	*0{M}32 * Käyttäjän etunimi ****
Sukunimi	*0{M}32 * Käyttäjän sukunimi ****
Email	*0{M}32 * Käyttäjän sähköpostiosoite ****

3.2 Käyttöintensiteetti

Sovellusta käytetään arkipäivisin pääasiassa klo 07-17 välillä. Yhtäaikaisia käyttäjiä on enimmillään 30, jotka suorittavat hakuja 1 min välein.

3.3 Kapasiteettivaatimukset

Järjestelmään on tallennettuna maksimissaan 1000 tehtävää liitteistettynä maksimissaan 1Mb liitetiedostolla. Järjestelmän ylläpitäjä tarvittaessa ylläpitää tietokantaa poistamalla vanhoja tehtäviä tallennuspäivän mukaisessa järjestyksessä.

3.4 Tiedostot ja asetustiedostot

Ei merkitystä tässä dokumentissa.

4. TOIMINNOT

4.1 Yleistä

Sovelluksen käyttöliittymä on suomenkielinen ja käyttäjän syötteissä on mahdollista käyttää skandinaavisia merkkejä. Sovellus toimitetaan tilaajalle oletuskäyttöliittymällä, mutta sovellus toteutetaan siten että tilaajan on itse mahdollista toteuttaa kohtuullisen helposti haluamansa käyttöliittymä sovellukselle. Oletuskäyttöliittymä skaalautuu eri kokoisille näytöille, jotta vältytään sivuttaissuuntaiselta näytön vierittämiseltä. Oletuskäyttöliittymä on kuvattuna alustavassa käyttöohjeessa.

Sovelluksen lähdekoodissa muuttujien nimissä käytetään pääosin englanninkielisiä sanoja tai lyhenteitä. Lähdekoodissa toiminnot kommentoidaan riittävällä tasolla ja kommentit kirjoitetaan suomenkielisinä.

4.2 Järjestelmän toiminnot

4.2.1 Kirjautuminen sovellukseen

Kuvaus: Käyttäjä, joka suorittaa sovellukseen tallennettuja tehtäviä, kirjautuu sisään sovellukseen syöttämällä kirjautumisikkunaan käyttäjätunnuksen ja salasanan ja painaa ”Kirjaudu”-painiketta. Kirjautuneella käyttäjällä on oikeudet muuttaa tehtävän tilaa tai poistaa tehtävä listalta.

Käsittely: Sovelluksessa tarkistetaan että käyttäjä löytyy sovelluksen käyttäjärekisteristä (=tietokannan taulu). Kirjautumisen onnistuessa käyttäjätunnus tallennetaan käyttäjän sessiotietoihin.

Virhetilanteet: Kirjautumisikkunassa tarkistetaan ennen kirjautumistietojen lähettämistä ovatko sekä käyttäjätunnus että salasana syötetty tekstikenttiin. Mikäli jompikumpi tai molemmat kentät ovat tyhjiä, saa käyttäjä heti ilmoituksen ”Syötä käyttäjätunnus/salasana” tms. Mikäli kirjautuminen ei onnistu väärin syötetyn käyttäjätunnuksen tai salasanan tai käyttöoikeuksien puuttumisen vuoksi, saa käyttäjä tästä ilmoituksen ja mahdollisuuden palata takaisin kirjautumissivulle.

4.2.2 Tehtävän lisääminen

Kuvaus: Käyttäjä painaa tehtävälistasivulla ”Lisää”-painiketta, josta käyttäjä ohjautuu tyhjälle tehtävälomakkeelle. Käyttäjä syöttää tehtävän tiedot lomakkeen tekstikenttiin ja painaa ”Tallenna”-painiketta, jolloin tiedot tallentuvat tietokantaan. Käyttäjä saa palautteen onnistuneesta tallennuksesta ja pääsee palaamaan tehtävälistaukseen.

Käsittely: Lomakkeella tarkistetaan ennen tietojen lähettämistä ovatko kaikki pakolliseksi määritetyt kentät täytetty. ”Tallenna”-painikkeella lomakkeen tiedot lähetetään palvelimelle, jossa suoritetaan määritetty SQL-lause lomakkeen tiedoilla.

Virhetilanteet: Mikäli pakollisiksi määritetyt kentät eivät ole täytetty, käyttäjä saa tästä palautteen (”Syötä [kentän nimi].”) eikä tietojen tallentaminen onnistu. Kenttiin syötetyt tiedot eivät häviä vaikkei lähettäminen vajaasti täytetystä lomakkeesta johtuen onnistu. Käyttäjä saa lisäksi virheilmoituksen mikäli palvelimella SQL-lauseen suorittaminen epäonnistuu.

4.2.3 Tehtävän liitteistäminen liitetiedostolla

Kuvaus: Käyttäjä voi halutessaan liittää tehtävään liitetiedostoja tehtävän lisäys- ja muokkauslomakkeella painamalla ”Lisää”-painiketta ja valitsemalla tiedoston avautuvasta tiedostoselausikkunasta.

Käsittely: Valittu tiedosto tallennetaan tietokantaan tehtävän lisäys- tai päivitystoiminnon (luvut 4.2.2 ja 4.2.4) yhteydessä.

Virhetilanteet: kts. luvut 4.2.2 ja 4.2.4.

4.2.4 Tehtävän tietojen tarkastelu

Kuvaus: Käyttäjä painaa tehtävälistasivulla halutun tehtävän nimeä (linkki), josta käyttäjä ohjautuu kyseisen tehtävän yhteenvetosivulle. Yhteenvetosivulta pääsee palaamaan takaisin tehtävälistaukseen.

Käsittely: Tietokannasta haetaan ko. tehtävän kaikki tiedot sivulle.

Virhetilanteet: Käyttäjä saa virheilmoituksen mikäli tietojen hakeminen ei onnistu.

4.2.5 Tehtävän tietojen muokkaus/päivitys

Kuvaus: Käyttäjä painaa tehtävälistasivulla halutun tehtävän nimeä (linkki), josta käyttäjä ohjautuu kyseisen tehtävän yhteenvetosivulle. Yhteenvetosivulla on ”Muokkaa”-linkki, josta avautuu tehtävän tietolomake. Käyttäjä syöttää tehtävän tiedot lomakkeen tekstikenttiin ja painaa ”Tallenna”-painiketta, jolloin tiedot päivittyvät tietokantaan. Käyttäjä palaa automaattisesti tietojen tallennuksen jälkeen yhteenvetosivulle, josta pääsee palaamaan tehtävälistaukseen.

Käsittely: Kts. luvun 4.2.2 ”Käsittely”.

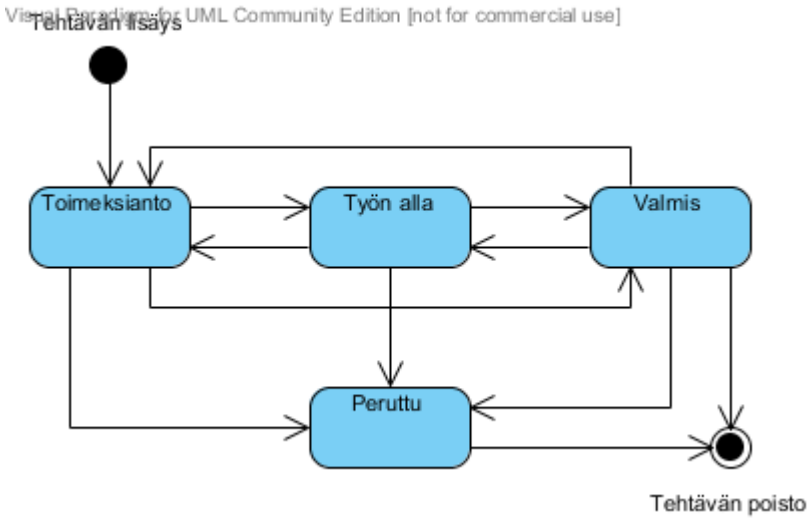
Virhetilanteet: Kts. luvun 4.2.2 ”Virhetilanteet”.

4.2.6 Tehtävän tilan muuttaminen

Kuvaus: Käyttäjä valitsee tehtävän yhteenvetosivulla alasvetovalikosta halutun tilan tehtävälle ja päivittää tehtävän tilan. HUOM! tehtävän tilan päivittäminen vaatii kirjautumisen sovellukseen.

Käsittely: Kts. lukujen 4.2.2 ja 4.2.5 ”Käsittely”. Sivulle siirryttäessä tarkistetaan onko käyttäjä kirjautunut sovellukseen ja mikäli on, lomakkeella näytetään tilan alasvetovalikko. Muussa tapauksessa alasvetovalikko ei ole näkyvissä. Tilan vaihtaminen tallentaa käyttäjän käyttäjätunnuksen tietokantaan tehtävän tietoihin. Tilan vaihto ”toimeksianto” -> ”työn alla” asettaa tehtävän aloitus- ja ”valmis” valmistumisajankohdan. HUOM! Aloitusajankohta tallennetaan vain kun tila vaihtuu ”toimeksianto” -> ”työn alla”, ei muissa tapauksissa. HUOM! Valmistumisajankohta tallennetaan aina kun tehtävä saa tilan ”valmis”.

Virhetilanteet: Käyttäjä saa virheilmoituksen mikäli tilanvaihto ei onnistunut.



Kuva 4.1: Tehtävän elinkaari tiloineen kuvattuna tilakaaviona.

4.2.7 Tehtävän poistaminen

Kuvaus: Käyttäjä painaa tehtävälistasivulla halutun tehtävän kohdalla ”Poista”-painiketta ja saa palautteen onnistuneesta poistosta sekä pääsee palaamaan tehtävälistaukseen. HUOM! tehtävän tilan poistaminen vaatii kirjautumisen sovellukseen. HUOM! tehtävän tulee olla ”Valmis”- tai ”Peruttu”-tilassa.

Käsittely: Tehtävälistaukseen tultaessa tarkistetaan onko käyttäjä kirjautunut sovellukseen ja mikäli on, sivulla näytetään tehtävän poistopainike. Muussa tapauksessa painike ei ole näkyvissä. Tehtävän poistotoiminto tarkistaa ensin onko tehtävä valmis tai peruttu ja ehdon täyttyessä suorittaa tarvittavat SQL-lauseet palvelimella, jolloin tietokannasta poistuu sekä tehtävä että tehtävään liittyvät liitetiedostot.

Virhetilanteet: Käyttäjä saa virheilmoituksen mikäli palvelimella SQL-lauseen suorittaminen epäonnistuu ja pääsee palaamaan tehtävälistaukseen.

4.2.8 Tehtävän kopiointi uudeksi tehtäväksi

Kuvaus: Käyttäjä painaa tehtävän muokkaussivulla ”Kopioi”-painiketta, jolloin siirrytään tehtävän lisäyslomakkeelle. Lomakkeelle kopioidaan seuraavat kentät vanhasta tehtävästä: ”AX”, ”Nimi”, ”Tiedot”, • ”Polku”, ”Liite”.

Käsittely: Kts. luvun 4.2.2 ”Käsittely”. Vanhan tehtävän liitteet kopioidaan uudelle tehtävälle.

Virhetilanteet: Kts. luvun 4.2.2 ”Virhetilanteet”.

4.2.9 Tehtävien listaus

Kuvaus: Tehtävälistaussivulla tehtävät listataan allekkain ja lajitellaan tehtävän tallennuspäivämäärän mukaan laskevasti (viimeisin tehtävä on listan ensimmäisenä). Listauksessa näytetään seuraavat kentät: ”Pvm”, ”AX”, ”Tilaaja”, ”Nimi”, ”Tiedot”, ”Deadline”, ”Muokattu”, ”Aloitettu”, ”Valmistunut”, ”Tekijä” ja ”Tila”. Näiden riveittäin kirjautuneille käyttäjille näytetään tehtäväkohtainen ”Poista”-painike. Pitkistä tekstikentistä tulostetaan listaukseen osamerkkijono (max ? merkkiä). ”Tilaaja”- ja ”Tekijä”-kentissä on linkki sähköpostilähetykseen (mailto:). Tehtävien ”Työn alla”- ja ”Valmis”-tilat korostetaan väreillä (”Työn alla” = punainen, ”Valmis” = vihreä). • hNimi • h-kentässä on linkki tehtävän yhteenvetosivulle.

Käsittely: Tehtävät listataan taulukkoon, joka muotoillaan css-tyylitiedoston mukaisesti. ”Tekijä”-kenttään tulostetaan tekijän käyttäjätunnus, mutta sähköpostiosoite haetaan tekijätaulusta sähköpostilähetystä varten. Oletuksena tehtävätaulusta haetaan ne tehtävät listaukseen, joiden tila on ”Toimeksianto” tai ”Työn alla” (suodatuksena ”Ei-valmis”, kts. tehtävien lajittelu luvussa 4.2.9).

Virhetilanteet: Käyttäjä saa virheilmoituksen mikäli tietokannasta ei saada haettua tietoja listaukseen.

4.2.10 Tehtävien suodattaminen tilan mukaisesti

Kuvaus: Käyttäjä valitsee tehtävälistaussivulla alasvetovalikon vaihtoehdoista haluamansa suodatuksen ja painaa ”Suodata”-painiketta, jolloin listanäkymään suodatetaan näkyville vain tietyissä tiloissa olevat tehtävät. Suodatusvaihtoehtoja ovat ”Ei-valmis” (tilat ”Toimeksianto” ja ”Työn alla”), ”Valmis” (tila ”Valmis”), ”Peruttu” (tila ”Peruttu”) ja ”Kaikki” (kaikki tilat). Valittu tila jää ”muistiin” käyttäjäkohtaisesti kunnes selain suljetaan.

Käsittely: Suodatustoiminto suorittaa palvelimella SQL-kyselyn, jossa tietokannasta palautetaan vain valitun tilan mukaiset tehtävät. Valittu suodin tallennetaan esim. käyttäjän sessioon, jolloin valittu tila pysyy muistissa. Mikäli valitulla tilalla ei löydy tehtäviä, listaus jää otsikkoja

lukuun ottamatta tyhjäksi.

Virhetilanteet: Kts. luvun 4.2.8 ”Virhetilanteet”.

4.2.11 Liitetiedoston lataus

Kuvaus: Käyttäjä painaa tehtävän muokkaussivulla tai tehtävän muokkaustoimintoa seuraavalla yhteenvetosivulla liitetiedostoon osoittavaa linkkiä, jolloin tiedosto latautuu työasemalle.

Käsittely: Liitetiedoston linkin painallus suorittaa kyselyn, joka palauttaa tiedoston tietokannasta.

Virhetilanteet: Käyttäjä saa virheilmoituksen mikäli lataus ei onnistu.

5. ULKOISET LIITTYMÄT

5.1 Laitteistoliittymät

Järjestelmässä ei liittymiä ulkoisiin laitteistoihin.

5.2 Ohjelmistoliittymät

Järjestelmässä ei liittymiä toisiin ohjelmistoihin.

5.3 Tietoliikenneliittymät

Järjestelmä toimii organisaation sisäverkossa. Sovellusta ei ole mahdollista käyttää sisäverkon ulkopuolelta. VPN-yhteyden (Virtual Private Network) avulla sovelluksen käyttö on tarvittaessa mahdollista myös julkisesta verkosta käsin.

6. MUUT OMINAISUUDET

6.1 Suorituskyky ja vasteajat

Järjestelmällä on kymmeniä yhtäaikaisia käyttäjiä (kts. luku 3.2). Tehtävälistauksen hakujen saantiajat tulevat olla %:ssa alle 1 sekunti, enintään 5 sekuntia.

6.2 Käytettävyys, toipuminen, turvallisuus, suojaukset

Järjestelmä on käytettävissä arkisin 07 – 17 välisenä aikana lukuun ottamatta mahdollisia lyhyitä ylläpitokatkoksia, joista tiedotetaan organisaatiota etukäteen. Järjestelmän tiedot on mahdollista varmuuskopioida normaalin palvelinvarmistuksen yhteydessä. Koska kyseessä on organisaation sisäverkossa toimiva sovellus, ei järjestelmällä ole erikseen omia tietoturvavaatimuksia.

6.3 Ylläpidettävyys

Järjestelmän ylläpitäjä ylläpitää tietokantaa (esim. käyttäjien lisäys) valitsemallaan varusohjelmalla. Sovelluksen mahdolliset uudet ominaisuudet toteutetaan omissa jatkokehitysprojekteissaan. Sovellus toteutetaan siten että sovelluksen käyttöliittymää on mahdollista päivittää ilman muutoksia itse sovellukseen.

6.4 Siirrettävyys ja yhteensopivuus

Ei tarvetta siirrettävyydelle tai yhteensopivuudelle esim. muille päätelaitteille.

6.5 Operointi

JavaScript tulee olla käytössä käyttäjän selaimessa.

7. SUUNNITTELURAJOITTEET

7.1 Standardit

Ei merkitystä tässä dokumentissa.

7.2 Laitteistorajoitteet

Ei merkitystä tässä dokumentissa.

7.3 Ohjelmistorajoitteet

Ei merkitystä tässä dokumentissa.

7.4 Muut rajoitteet

Ei merkitystä tässä dokumentissa.

8. HYLÄTYT RATKAISUVAIHTOEHDOT

Ei merkitystä tässä dokumentissa.

9. JATKOKEHITYSAJATUKSIA

Sovellukseen kehitetään raportointiominaisuus, joka määritellään erikseen.