TEKNIIKKA JA LIIKENNE

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

LAITTEISTOHALLINTATIETOKANNAN JA SEN KÄYTTÖLIITTYMÄN SUUNNITTELU

Työn tekijä: Jaakko Varjo
Työn valvoja: Juhani Rajamäki
Työn ohjaaja: Juuso Pesola

Työ hyväksytty: 27. 4. 2010

Juhani Rajamäki
lehtori

ALKULAUSE

Tämä insinöörityö tehtiin Envault Corporationille. Kiitän kaikkia projektin eri osa-alueiden toteutukseen osallistuneita henkilöitä ja toivotan uuden järjestelmän parissa työskenteleville henkilöille onnea ja menestystä. Erityisesti osoittan kiitokseni Envault Corporationin Juuso Pesolalle sekä Tuukka Autiolle, joiden ansiosta tämä projekti oli mahdollinen toteuttaa.

Helsingissä 2.3.2010

Jaakko Varjo

Metropolia

TIIVISTELMÄ

Työn tekijä: Jaakko Varjo

Työn nimi: Laitteistohallintatietokannan ja sen käyttöliittymän suunnittelu

Päivämäärä: 2.3.2010 **Sivumäärä:** 52

Koulutusohjelma: **Suuntautumisvaihtoehto:**

Tietotekniikka Ohjelmistotekniikka

Työn valvoja: Lehtori Juhani Rajamäki

Työn ohjaaja: Project manager Juuso Pesola

Tässä insinöörityössä esitellään suunnitelma laitehallintajärjestelmän toteutukseen tietokannan rakenteen, selainpohjaisen käyttöliittymän sekä hyödynnettävien ohjelmistojen ja ohjelmistokomponenttien osalta.

Järjestelmä suunnitellaan Envault Corporationille korvaamaan nykyinen käytössä oleva graafinen hallintaliittymä. Selainpohjaisen käyttoliittymän toteutukseen käytetään Java-teknologiaan perustuvaa Vaadin-sovelluskehystä ja tietokanta toteutetaan käyttämällä PostgreSql-tietokantaa.

Tämän työ yhteydessä on tarkoitus saada järjestelmän toteutuksen suunnitelma valmiiksi sekä tuottaa rajalliset toiminnallisuudet omaava testiversio.

Avainsanat: Vaadin-sovelluskehys, Envault Corporation, PostgreSQL, Java

Metropolia

ABSTRACT

Name: Jaakko Varjo

Title: User Interface and Database development plan for device management system

Date: 2.3.2010 **Number of pages:** 52

Department: **Study Programme:**

Information technology Software engineering

Supervisor: Lehtori Juhani Rajamäki

Instructor: Project Manager Juuso Pesola

This final thesis presents a development plan for implementing an device management system, which consist of database structure, web browser based user interface and definition of software and software components to be used in the implementation.

The information management system is designed for Envault Corporation in order to replace the management system and user interface that is currently in use. In the near future, the implementation of the web browser based user interface will be developed using Java-technology based web application framework Vaadin, and the database solution will be designed using PostgreSql database.

In the scope of this thesis, the goals to achieve are to finalize the development plan, and to develop a preliminary version with very limited functionality.

Keywords: Vaadin - framework, Envault Corporation, PostgreSql, Java

SISÄLLYS

1 JOHDANTO

Tämä insinöörityö esittelee laittestohallintajärjestelmän toteutuksen korvaamaan Envault Corporationin nykyisen graafisen hallintajärjestelmän osana Envault™ Removable Media Protectionin [1] tuotekokonaisuutta. Järjestelmän suunnitteluun kuuluu tietokannan sekä selainpohjaisen käyttöliittymän suunnittelu.

Envault Corporation on suomalainen tietoturvaan ja erityisesti ulkoisille tallenuslaitteille sijoitetun informaation turvaamiseen ja hallintaan erikoistunut yritys.

Envault™ Removable Media Protection mahdollistaa tietotekniikassa käytettyjen ulkoisten massamuistilaitteiden salasanattoman suojauksen, etähallinnan sekä käytönvalvonnan. Tämän järjestelmän perusideana on tiedon ”holvaaminen”, eli ulkoiselle massamuistilaitteelle tallennettavan materiaalin manipuloiminen sekä siitä poistettavan prosentuaalisen osan, eli ”fragmentin” tallentaminen turvattuun sijaintiin erilleen muusta materiaalista.

Tämän operaation tuloksena alkuperäinen tallennettu materiaali yksinään on lukukelvotonta ja materiaalin käyttäminen vaatii alkuperäisen ulkoiselle laitteelle tallennetun osan sekä siitä poistetun osan.

Tätä toimintamallia soveltamalla voidaan luoda tilanne, jossa estämme pääsyn materiaalista poistettuun osaan ja täten estetään koko materiaalin käytön. Näin voidaan määritellä, kenellä on oikeus käyttää tallennettua materiaalia sekä kerätä tietoa siitä, kuka on hyödyntänyt mahdollisuuttaan käyttää tallennettua materiaalia.

Envault™ Removable Media Protection tarjoaa selainpohjaisen käyttöliittymän, jolla loppukäyttäjä voi helposti hallita järjestelmän piirissä toimivia laitteita. Tämä käyttöliittymä perustuu tietokantaan, johon järjestelmä tallentaa tietoa laittesta ja käyttäjistä, jotka ovat käyttäneet järjestelmän suojaamaa materiaalia. Uuden käyttöliittymän sekä tietokannan alustava suunnittelu aloitettiin keväällä 2009, kun todettiin olemassaolevan käyttöliittymän sekä tietokannan olevan kykenemättömät vastaamaan järjestelmän nykyhetken asettamia vaatimuksia. Kesällä 2009 aloitettiin projektin varsinainen kehitystyö vanhan käyttöliitymän ylläpidon ohella.

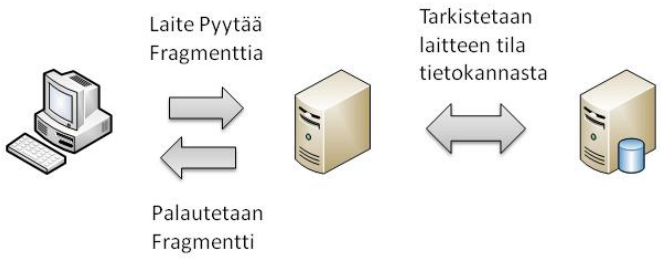
Tämä insinöörityö keskittyy järjestelmän suunnitteluun jonka aikana tuotettaan ensimmäinen rajallisen toiminnallisuuden omaava testiversio. Projekti ei noudata mitään erillisesti valittua ohjelmistotuotantomenetelmää, tosin projektin luonteesta, sekä jatkuvasti muuttuvista määrittelyistä johtuen lopputulos oli lähellä ketterien menetelmien yleisiä periaatteita.

Tämän insinöörityön tavoitteena on suunnitella laitteistohallintajärjestelmän rakenne ja tuottaa tämän suunnitelman mukainen testiversio. Järjestelmän on tarkoitus toimia käyttöliittymänä tietokannan hallintaan mahdollistaen lähes reaaliaikaisen toiminnan sekä miellyttävän yksinkertaisen käyttökokemuksen.

Järjestelmän perustoiminnot ovat

- laitteiden hallinta
- laitteiden nykyisen tietosisällön tarkastelu
- tietosisältöön tapahtuneiden toimintojen tarkastelu.

Laitteiden hallita käsittää järjestelmään liitetyn laitteen tilan tutkimisen ja sen muuttamisen. Laitteella voi olla kolme tilaa: ’ACTIVE’, ’BLOCKED’ ja ’DELETED’. Järjestelmään liitetty laite, joka haluaa käyttää suojattua materiaalia, pyytää palvelimelta materiaalista poistettua osaa, joka taas tarkistaa tietokannasta kyseisen laitten tilan (kuva 1).



Kuva 1. Asiakaslaite haluaa käyttää suojattua materiaalia

Laite, joka on tilassa ’ACTIVE’, on normaalisti toimiva laite, jonka tietosisältö on käytettävissä järjestelmän piirissä. ’BLOCKED’ on laite, jonka tietosisältöön pääsy on estetty, ja ’DELETED’ on laite, jonka sisältämästä materiaalista eristetyt palaset on peruuttamattomasti tuhottu. Täten käytännössä sen tietosisältö on peruuttamattomasti tuhottu. Tämä tilatieto tallennetaan tietokantaan, jotta voidaan tarkastella myös järjestelmästä tuhottuja laitteita.

Laitteiden nykyisen tietosisällön tarkastelu käsittää järjestelmän piirissä olevan laitteeseen siirrettyn materiaalin tarkastelua nykyhetkellä. Laitteeseen siirretyistä tiedostoista kirjoitetaan lokitietoa tietokantaan, jonka avulla voidaan rakentaa kuva laittessa nykyhetkellä sijaitsevasta materiaalista.

Tietosisältöön tapahtuneiden toimintojen tarkastelu käsittelee laitteiden tietosisältöön kohdistuneiden operaatioiden kokonaisuutta ja sen osa-alueita. Käytännössä tämä tarkoittaa lokitietoja laitteeseen kohdistuneista operaatioista, joiden avulla voidaan luoda tarkka kuva laitteen käytöstä.

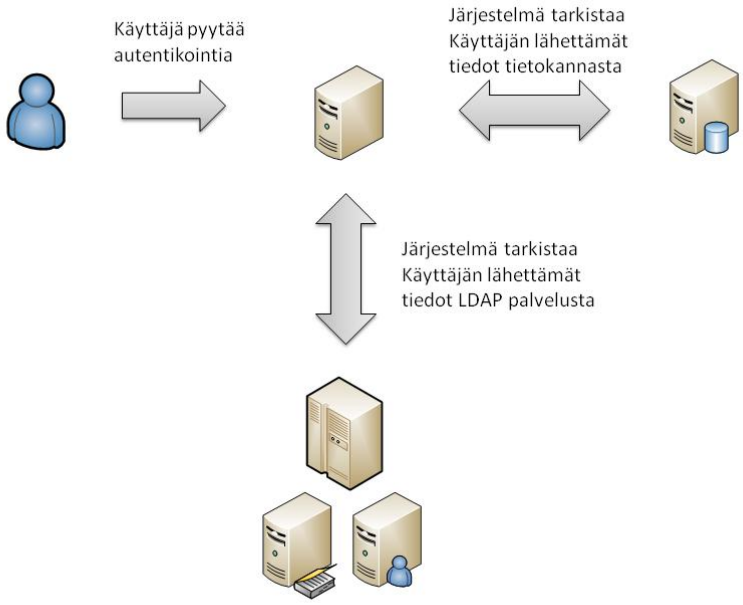
Käytännössä näiden kolmen toiminnallisuuden toteuttaminen tarkoittaa tietokannasta saatavan tiedon rakentamista ihmiselle luettavaan muotoon ja tarvittaessa sen muokkaamista.

Perustoimintojen lisäksi tuodaan esille kaksi järjestemän kannalta tärkeää toiminnallisuutta

- käyttäjän salasanan muuttaminen
- käyttäjän autentikointi käytettäessä graafista käyttöliittymää.

Tätä raporttia kirjoitettaessa näiden toimintamallien lopullinen toteutus on vielä auki. Käyttäjän autentikointiin on tosin esitetty kahta mallia, joista ensimmäisessä järjestelmän oma tietokanta ylläpitää käyttäjätietoja asianmukaisesti suojattuna ja toisessa autentikointi toteutetaan kolmannen osapuolen olemassa olevaa palvelua vasten. Käytännön toteutuksena tällä tarkoitettaisiin Lightweight Directory Access Protocol (LDAP) -standardiin [2] perustuvan järjestelmän hyödyntämistä (kuva 2).

Tässä raportissa ei käsitellä tämän yksityiskohtaisemmin käyttäjän autentikointia koskevia malleja tai niistä koituvia hyötyjä tai haittoja.



Kuva 2. Autentikoinnin vaihtoehtoiset mallit

Järjestelmän lokalisointi tullaan toteuttamaan CSV (Comma Separated Values)-tiedostoon [3] pohjautuvalla ratkaisulla, jonka sisällö on loogisesti ajateltuna taulukko, vaikka itse CSV-tiedosto sisältää pelkkiä alfanumeerisia merkkejä. Tätä tiedostoa on helppo muokata jälkikäteen esimerkiksi talukkolaskentaohjelmalla.

CSV-tiedoston yksi rivi pitää sisällään yhden tekstikentän sisällön eri kielivaihtoehdot puolipisteellä eroteltuina ja yksi sarake sisältää yhden kielen kaikki tekstikentät. Täten voidaan jälkikäteen lisätä CSV-tietostoon sarake, johon tallennetaan uuden kielen halutut merkkijonot, ja valitsemalla CSV-tiedostosta sarake, saadaan kaikki järjestelmän sisältämät merkkijonot valittua.

2 PROJEKTIN KUVAUS

Järjestelmän toteutus tämän projektin osalta rakennetaan käyttäen kolmea pääkomponenttia:

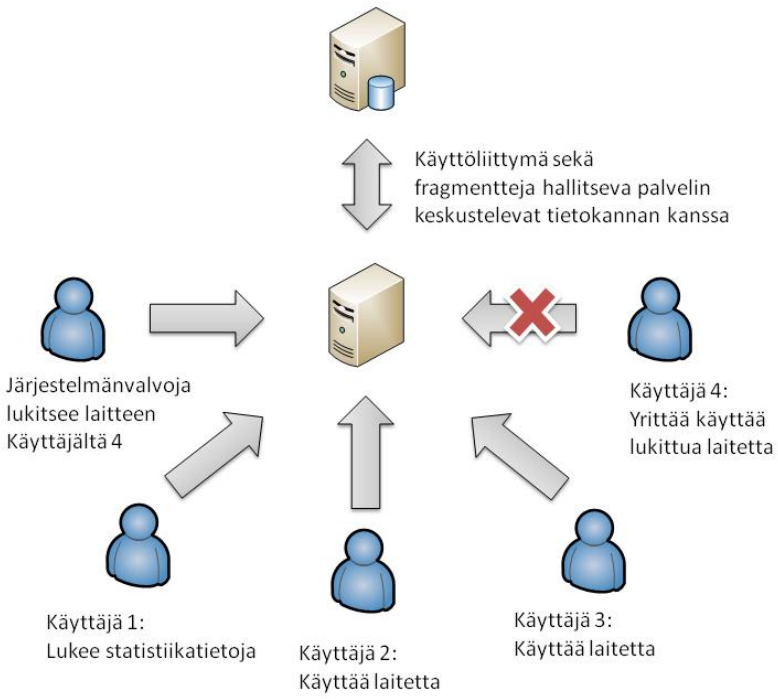
- Vaadin 6.1.0 Sovelluskehys (VF)[4]
- Apache Tomcat 6.0 -komponenttisäiliö Java-servlet -objekteille (ATSC)[5]
- PostgreSql 8.3 -tietokanta (PG)[6].

Käyttöliittymän toteutukseen valittiin Java-teknologiaan [7] perustuva Vaadin -sovelluskehys, joka on Google Web Toolkit [8] toteutuksen pohjalta rakennettu avoimen lähdekoodin projekti. Tämän avulla on mahdollista rakentaa nykyistä käyttöliittymää skaalautuvampi ja dynaaminen järjestelmä, jonka keskeisenä ideakokonaisuutena on asiakkaalle näytettävien html-sivujen sekä ajax-komponenttien (Asyncronous Javascript And Xml) [9] luominen ohjelmallisesti java-ohjelmointikielellä kirjoitetusta koodisisällöstä.

Näin toteutetaan asiakkaan selaimessa dynaamisen sisällön hallinta javascript - teknologialla sekä täytetään vaatimus selain- ja asiakaspään riippumattomuudesta. Tämä toimintamalli myös hyödyntää Vaadin -sovelluskehyksen ominaisuutta optimoida tuotettu javascript-ohjelmakoodi [10] eri selaimille ja rakentaa selainkohtaiset javascript-komponentit, jolloin tuetut selaimet käyttävät ainoastaan niille optimoituja komponentteja ja tukemattomat selaimet käyttävät yleisiä komponentteja.

2.1 Projektin pitkäaikaiset tavoitteet ja vaatimukset

Projektin suunnitteluvaiheessa keskeiseksi tavoitteeksi asetettiin normaalin työpöytäsovelluksen kaltaisen web-sovelluksen [11] toteuttaminen. Käyttöliittymän on tarkoitus toimia monen yhtäaikaisen, järjestelmän näkökulmasta eriarvoisen käyttäjän työkaluna järjestelmän kautta salatun materiaalin hallintaan. Järjestelmän näkökulma käyttäjiin on hierarkinen, joten on otettava huomioon käyttäjien vuorovaikutus toisiinsa ja käyttäjän hierarkiatason vaikutus käyttäjälle sallituihin vapauksiin järjestelmän piirissä (kuva 3).



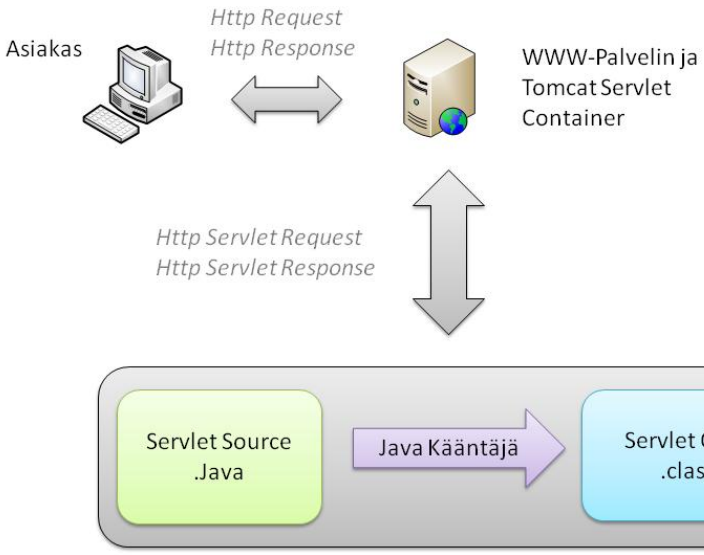
Kuva 3. Käyttäjien vuorovaikutus

Järjestelmän luonteeseen kuuluu olennaisena osana myös reaaliaikaisuus, jotta suojatun materiaalin käyttöoikeuksien muutokset tulevat voimaan heti muutoksen hetkellä. Tämä vaatimus asettaa tietokannan responsiivisuuden korkealle prioriteetille.

2.2 Käyttöliittymä

Järjestelmän käyttöliittymä toteutetaan yleiseen asiakas-palvelin (Client-server) -malliin [12] perustuvana web-sovelluksena, jossa palvelin on Apache Tomcat Serlet Container (TSC) [5] tai vaihtoehtoisesti joku muu www-palvelin, johon on mahdollista integroida Java-Servlet [13] objekteja tukeva komponenttisäiliö. Tässä toimintamallissa asiakkaan internetselain pyytää web-palvelimelta näytettävää sivua ja saadessaan HttpRequest [13] -pyynnön TSC Enkapsuloi kyseisen pyynnön HttpServletRequest [13] -pyynnöksi, ohjaten tämän haluttua sivua vastaavalle servlet-luokalle, joka käännetään pyynnön saapuessa ja siitä luodaan objekti palvelimen muistiin.

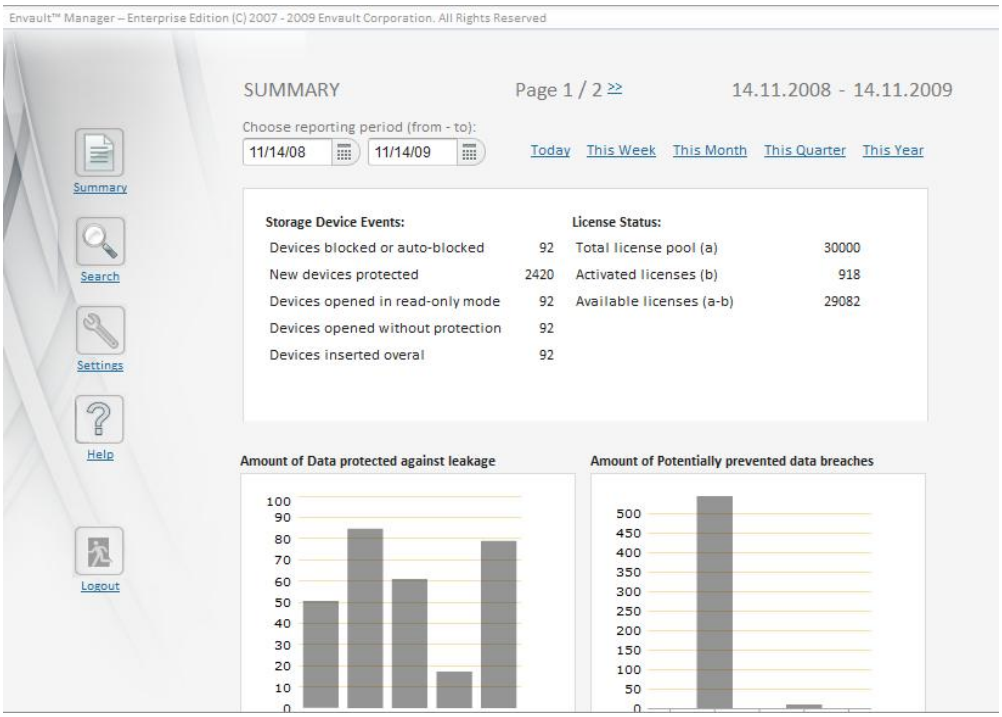
Suoritettuaan tarpeelliset toiminnot servlet-objekti ohjaa TSC:lle HttpServletResponse [13] -tyyppisen vastauksen, joka puolestaan muokkaa saamansa vastauksen HttpResponse [13] tyyppiseksi vastaukseksi, rakentaa tarvittavat otsikkotiedot sekä enkoodaa vastauksen lokalisoinnin mukaiseksi ennen sen palauttamista takaisin alkuperäisen pyynnön lähettämälle selaimelle.



Kuva 4. Asiakas pyytää sivua palvelimelta

TSC:n ja servlet-luokkien etu verrattuna vaihtoehtoisiin tekniikoihin on sen ominaisuus pitää kyseinen servlet-luokasta luotu objektiluokka tallessa vastauksen lähettämisen jälkeen. Järjestelmä hyötyy tästä ominaisuudesta, koska asiakkaalle ensimmäisenä esitettävä sivu muodostetaan yhdestä servlet-luokan objektista ja tämä sivu sisältää järjestelmän kontrollit. Kun asiakas liikkuu järjestelmässä, käytetään ajax- sekä javascript-tekniikoita päivittämään selaimessa näkyvää sivua dynaamisesti, jolloin alkuperäinen servlet-luokan objekti säilyy palvelimen muistissa ja on saatavilla nopeasti, jos asiakas esimerkiksi vahingossa lataa sivun uudelleen.

Suunniteltavassa käyttöliittymässä asiakaspuolen toteutus rakennetaan Vaadin sovelluskehystä [4] käyttäen. Käyttäen tätä työkalua toimitaan ohjelmointiteknisesti kuin rakennettaessa työpöytäsovellusta esimerkiksi java.awt (Abstrack Windowing Toolkit) [14] luokkakirjastoa hyödyntäen. Vaadin-sovelluskehyksen avulla saadaan toteutettua haluttu dynaamisuus käyttöliittymään sekä vaatimuksissa esitetty työpöytäsovelluksen kaltainen käyttötuntemus.

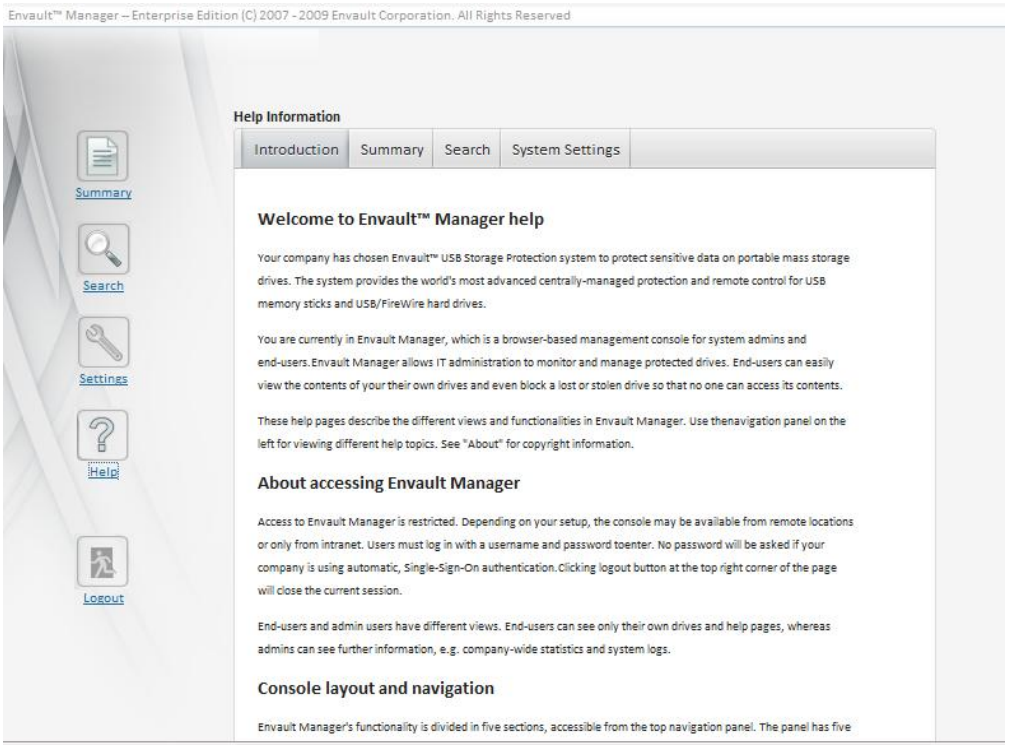


Kuva 5. Järjestelmän ”summary” -sivun prototyyppi

Käyttöliittymän suunnitteluvaiheessa yleisien toiminnallisuuksien lukumäärä haluttiin laskea minimiin, jolloin järjestelmän käyttö olisi mahdollisimman suoraviivaista, ja sen käyttölogiikka olisi helppo omaksua. Tästä johtuen toiminnallisuudet ryhmitellään neljään kategoriaan, joiden kautta navigointi olisi mahdollisimman yksinkertaista.

Järjestelmän neljä päätason navigointikomponenttia ovat

- ”Summary”-kooste järjestelmän eri osa alueiden toiminnasta (kuva 5)
- ”Search”-järjestelmän hakutoiminnallisuus (kuva 7)
- ”Settings”-järjestelmän toimintaan vaikuttavat toiminnallisuudet
- ”Help” -kooste järjestelmän käyttössä avustavasta tiedosta (kuva 6).



Kuva 6. ”Help” -sivun protyyppi

Tässä ryhmittelymallissa ”Summary”-osio esitettiin järjestelmään kirjauduttaessa esitettäväksi näkymäksi, joka jaettaisiin loogisesti kahteen osaan. Ensimmäinen osio sisältää järjestelmän kannalta kriittisimmät statistiikkatiedot ja toinen osio syvällisen katsauksen järjestelmän toimintaan.

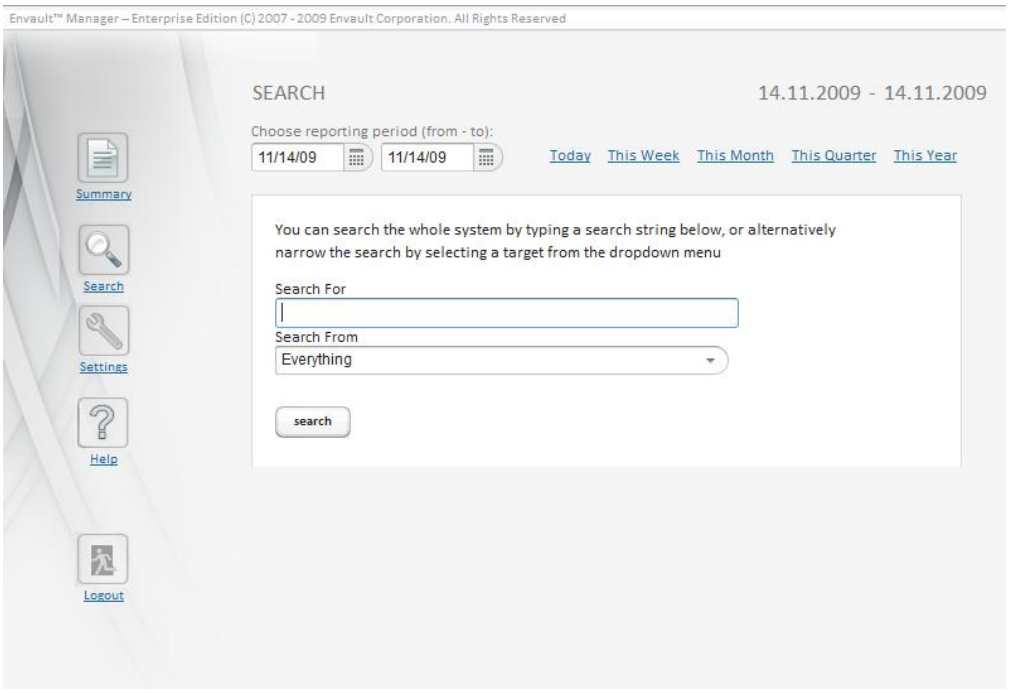
Järjestelmän statistiikkatiedon esittämiseen vaadittiin graafisia kuvaajia, näin päädyttiin ratkaisuun, jossa nämä kuvaajat rakennetaan Open Flash Chart 2 [15] -työkalulla, joka on Adobe System:in tuottamalla Actionscript 3 [16] -ohjelmointikiellellä toteutettu avoimen lähdekoodin projekti. Tämän komponentin käyttö mahdollistaa esitettävien kuvaajien sisällön dynaamisen päivittämisen javascript-teknologialla.

Toinen esitetty vaatimus tiedon esittemiselle oli sen sitominen aikaan. Tässä päädyttiin ratkaisuu jossa koko käyttöliittymällä on määriteltävissä oleva universaali aika. Koska järjestelmän ”Summary” -osiossa esitettävän informaation pienin erottava aikaväli on yksi päivä, päädyttiin universaalina aikana käyttämään kahta päivämäärää, jotka molemmat asetetaan järjestelmään kirjaduttaessa nykyhetkeen.

Universaalin ajan hallintaan rakennettiin kaksi valittaessa avautuvaa kalenterinäkymää sekä pieni joukko linkkityylisiä painikkeita, joilla voi pikavalita ajan esimerkiksi tähän kuukauteen.

Järjestelmän universaalia aikaa muuttaessa näytettävä statistiikkatieto generoidaan uudelleen ja päivitetään dynaamisesti. Järjestelmän universaali aika vaikuttaa myös hakutoimintojen tuloksiin. Tällä toteutuksella halutaan luoda vaikutelma, jossa käyttäjä voi järjestelmän informaatiossa liikkua vaapasti ajasta toiseen ja tutkia joko nykyhetkeä, eilistä, koko järjestelmän elinkaarta tai tiettyä spesifioitua hetkeä. Tämän toiminnalisuuden vieminen myös hakutuloksiin mahdollistaa järjestelmän tutkimisen pienissä osissa.

Kun nämä toiminnallisuudet toteutetaan Ajax-tekniikkaa hyödyntäen, saadaan luotua joustavan tuntuinen käyttökokemus.



Kuva 7. Hakutoiminnallisuus haettaessa koko järjestelmästä

Järjestelmän informatiivisesta luonteesta johtuen tiedon haku sekä esittäminen nousevat korkealle prioriteetille. Hakutoiminnallisuutta suunniteltaessa esitettiin toivomus hakukonemaisesta toimintamallista, jossa haku jolle ei ole erikseen määritelty kohdetta tuottaisi koosteen hakusanaa sivuavista tuloksista sekä mahdollisuuden käyttää tiedon haussa suodattimia, jolloin järjestelmään kohdistuvat hakutoiminnallisuudet saataisiin mahdollisimman tarkoiksi ja yksityiskohtaisiksi.

”Help” -osion sisältämät tekstit luetaan erikseen standardimuotoisesta xhtml-tiedostoista (Extended Hypertext Markup Language) [17]. Näin järjestelmän ylläpidon on helppoa lisätä materiaalia avustesivuille sekä muokata näytettävän avustesivun ulkoasua.

Järjestelmän ”Settings” -osion toiminnallisuudet ovat tätä raporttia kirjoittaessa vielä määrittelemättä lukuunottamatta käyttäjän oman salasanan vaihtamista.

2.3 Tietokanta

Järjestelmän toteutukseen valittiin avoimen lähdekoodin PostgreSQL[6] -tietokanta. Tietokannan valintaan vaikuttivat PostgreSQL:n laajat kustomointimahdollisuudet, aikaisempi käyttökokemus, olemassa olevien työkalujen saatavuus sekä PostgreSQL:n avoin lisenssointimalli, joka mahdollistaa tuotteen käyttämisen osana kaupallista sovelluskokonaisuutta. Tietokannan loogisen ja fyysisen mallin suunnittelemiseen käytetiin IBM Infosphere Data Architect ohjelmistoa [18]. Esimerkeissä käytetyt taulujen ja sarakkeiden nimet ovat konseptitason esimerkkejä ja tarkoitukseltaan sisältöään kuvaavia, eikä niiden odoteta huomioivan eri tietokantavalmistajien asettamia rajoitteita tietokannan objektien nimeämiselle.

2.3.1 Tietokannan suunnitteluprosessi

Tietokantasuunnittelun ensimmäinen vaihe on tietokannan informaation analysointi käsitteellisellä tasolla ilman konkreettista yhteyttä informaatiota ylläpitävän tietokannan fyysiseen toteutukseen. Tässä analyysissä määritellään tietokannan sisältämä informaatio yksittäisiin kokonaisuuksiin, *entiteetteihin* [19], ja näiden sisältämiin ominaisuuksiin, *attribuutteihin* [19]. Kaikkien tarvittavien entiteettien sekä attribuuttien määrittemisen jälkeen prosessin seuraava vaihe on informaation keskenäisten yhteyksien määritteleminen.

Tämän prosessin osa-alueen tuloksena yleisesti muodostetaan ns. relaatiomalli, joka kuvaa tallennettavaa informaatiota rakenteena. Relaatiomallia kutsutaan usein tietokannan *loogiseksi malliksi* [19].

Tietokantasuunnittelun kolmas vaihe on fyysisen mallin luominen, jolla tarkoitetaan tietokannan kokonaisvaltaisen rakenteellisen mallin luomista. Luotaessa fyysistä mallia tietokannasta otetaan huomioon käytettävän tietokantajärjestelmän mahdolliset rajoitukset sekä kaikki muut tarvittavat komponentit, joita tarvitaan suunniteltua informaatiota ylläpitävän tietokannan toteuttamiseen.

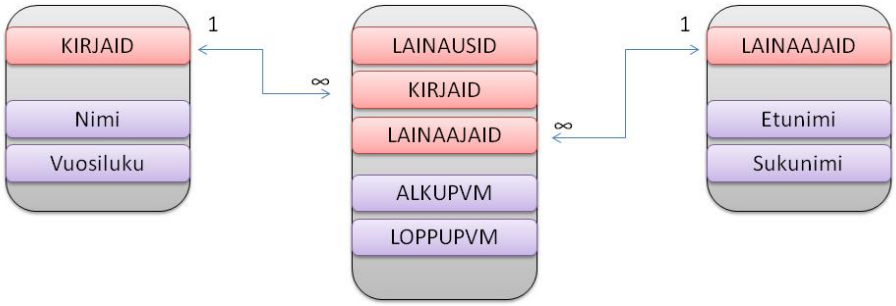
Projektin luonteesta johtuen voidaan katsoa tietokannan käsitteellisen analyysin tehdyksi jo ennen tämän projektin alkamista. Täten näitä projektin osa-alueita ei käsitellä tässä dokumentaatiossa.

2.3.2 Relaatiotietokannan normalisointi

Järjestelmän informatiivinen luonne ja tietokantaan tukeutuminen tietoturvaa tarjoavassa palvelumallissa asettaa järjestelmän tietokantasuunnittelulle korkeat vaatimukset. Täten vaatimuksia määriteltäessä esitettiin toivomus tietokannan normalisoinnista kolmanteen normaaliin asti.

Tietokannan normalisoinnilla tarkoitetaan tietokannan loogista rakennetta suunniteltaessa läpikäytävää vaiheittaista prosessia, jossa tutkitaan relaatiotietokannan rakennetta ja sen kykeneväisyyttä täyttämään eri normaaliasteille asetetut kriteerit [19]. Näiden vaiheiden tarkoituksena on relaatiotietokannan rakenne parhaiten tukemaan tiedon eheää ja tehokasta saatavuutta.

Normaalimuodot ovat järjestetty tavalla, jolloin seuraava normaalitaso sisältää aina edellisen.



Kuva 7. Normalisoitu tietokantarakenne

Ensimmäinen normaali määrittelee tietokannan sisältämän tiedon atomiseksi. Tämä tarkoittaa tietokannan taulujen sisältämien sarakkeiden attribuuttien määrittämistä siten, että ne ovat yksiarvoisia. Tällä toimintamallilla esimerkiksi ihmisen nimi koostuu kahdesta tietokannan solusta, etu- ja sukunimestä. Toinen ja kolmas normaali määrittelevät rajoituksia tietokannan attribuuteille, jotka eivät ole avainattribuutteja.

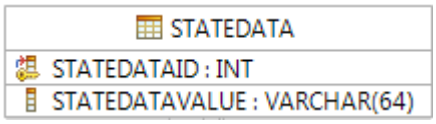
Toisessa normaalissa mikään ei-avainattribuutti ei saa olla osittain riippuvainen avainattribuuteista. Jos tietokannan jokaisen taulun avaimet ovat rakennettu vain yhdestä attribuutista, on tietokanta toisen normaalin mukainen. Jos taas taulujen avaimia rakennetaan useammasta attribuutista tulee tarkistaa että ei-avainattribuutit eivät riipu avainattribuutien osista, vaan koko avaimesta. Kolmas normaali määrittelee rajoituksen, jossa ei-avainattribuutit saavat olla riippuvaisia vain ja ainoastaan avainattribuuteista.

Relaatiotietokannan normalisointi on yleisesti suositeltu hyväksi todistettu lähestymistapa tietokantojen suunnitteluun. On kuitenkin huomattava, että monista nykyaikaisista reaalitietokannoista puuttuu täydellinen erottely tietokannan loogisen rakenteen ja varsinaisen informaation fyysisen tallennusmetodiikan väliltä. Tällöin on mahdollista ilmentyä skenaario, jossa tietokannan normalisointi itseasiassa heikentää tietokannan suorituskykyä.

Tästä skenaariosta johtuvaa tietokannan normalisoinnin tahallista rikkomista sanotaan denormalisoinniksi. Tämän kaltaiseen käyttäytymisen huomiointi on suunnitteluläheisesti hyvin vaikeaa ja varsinainen denormalisoinnin tarve ilmeneekin usein vasta tietokannan rasitustesteissä. Tässä työssä ei oteta huomioon mahdollista denormalisoinnin tarvetta.

2.3.3 Järjestelmän tietokannan looginen rakenne

Järjestelmän keskeisiin ominaisuuksiin kuuluu objektien tilan tutkiminen sekä muuttaminen. Siksi käsitellään ensimmäisenä järjestelmän staattiset taulut. Näiden komponenttien avulla määritellään monet dynaamisten tietokantakomponenttien attribuuteista. Ensimmäinen staattinen taulu on objektin yleistä tilatietoa osoittava taulu STATEDATA.



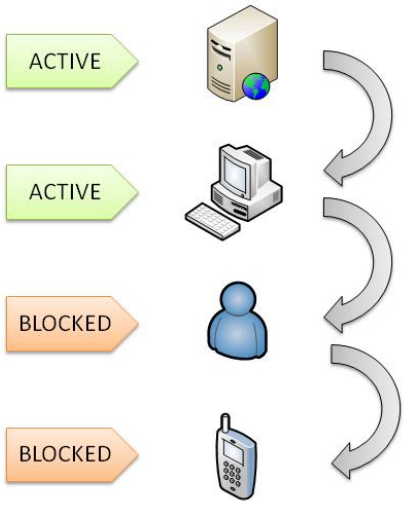
Kuva 8. Objektin tilaa kuvaava taulu

Taulu rakennetaan kahdesta sarakkeesta, joista ensimmäinen on kokonaisluku sekä taulun avain-attribuutti STATEDATAID ja toinen on STATEDATAVALUE, joka sisältää varsinaisen tilatiedon merkkijonona.

Kuten aikaisemmin todettiin, järjestelmään liitetty laite tai objekti voi olla kerrallaan vain yhdessä tilassa, joka on ”ACTIVE”, ”BLOCKED” tai ”DELETED”. Kun määritellään objektien tilan käsite näin korkealla abstraktiotasolla, saadaan pidettyä tilatietoa ylläpitävän taulun koko hyvinkin pienenä, jolloin tilatiedon hakeminen on nopeaa.

Tilatieto on myös hierarkinen samalla tavalla kuin järjestelmän näkökulma käyttäjiin. Tällä tarkoitetaan ajattelumallia jossa tietokannan objektit asetetaan tärkeysjärjestykseen tyyppinsä mukaan, vastaavasti kuin käyttäjät asetetaan järjestykseen roolinsa mukaan. Tilatiedon hierarkiassa ylimpänä on asiakasyritys, seuraavana on työasema, kolmantena käyttäjä ja viimeisenä laite.

Tämän lisäksi määrittellään järjestelmän hierarkian toimintamallin olevan restriktiivinen. Voidaan ajatella, että jos ylemmän objektin tila on ”ACTIVE”, voi alemman objektin tila olla mikä vaan, mutta jos ylemmän objektin tila on ”BLOCKED” tai ”DELETED” on myös alempi objekti pakotettu samaan tilaan.

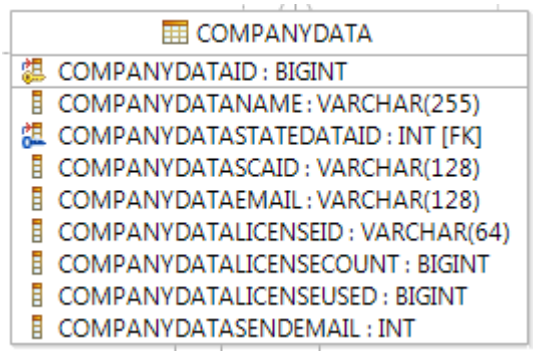


Kuva 9. Objektien hierarkia

Seuraavaksi käsitellään käyttäjän hierarkista asemaa kuvaavan taulun ROLEDATA, sekä laitteen tyyppiä kuvaavan taulun TYPEDATA. Molemmat näistä tauluista rakennetaan samalla periaatteella kuin aikaisempi STATEDATA taulu eli numeerisesta avainkentästä sekä merkkijonosta.

Kuva 10. Käyttäjän rooli ja laitteen tyyppi tietokantatoteuksena

Näiden kolmen staattisen taulun avulla voidaan hallita eri objektien käyttäytymistä järjestelmässä. Ensimmäinen dynaaminen objekti on asiakasyritystä esittävä COMPANYDATA taulu. Asiakasyritys on järjestelmän näkökulmasta yksi instanssi, jolla on oikeus käyttää järjestelmää.

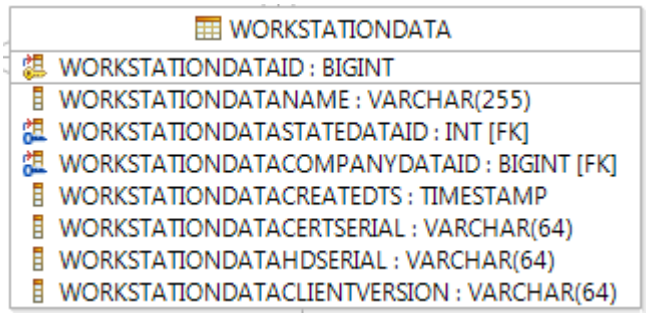


Kuva 11. Asiakasyritys

Asiakasyritystä kuvaava taulu on loogisesti ajateltuna järjestelmän sisäinen toimialuemääritelmä, joka kerää yhteen asiakkaan nimen ja sähköpostiosoitteen, jonne voidaan lähettää tieto asiakastietokoneen asennuksen onnistuessa, tiedot asennetuista lisensseistä sekä tietokentät COMPANYDATASCAID ja COMPANYDATALICENCEID, joita tarvitaan, kun järjestelmään asennetaan uusi asiakastietokone.

COMPANYDATASTATEDATAID on vierasavain STATEDATA -taulun numeeriseen avainattribuuttiin. Tämä määrittelee asiakasyrityksen nykyisen tilan.

Seuraava määriteltävä dynaaminen komponentti on asiakastietokonetta kuvaava WORKSTATIONDATA taulu.

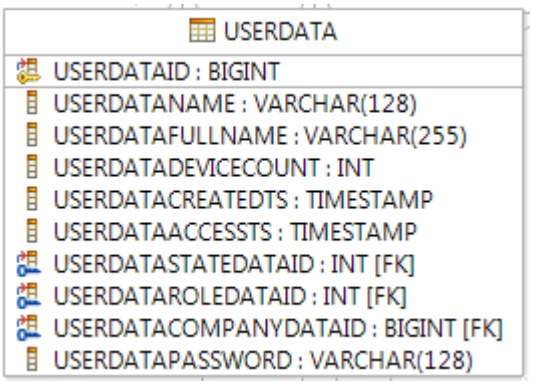


kuva 12. Asiakastietokonetta kuvaava taulu

Asiakastietokonetta kuvaava taulu kokoaa yhteen tiedot asiakastietokoneen nimestä, asennetun asiakasohjelmiston versiosta sekä asennuksessa käytetyn sertifikaatin sarjanumerosta, asiakaskoneen kiintolevyn sarjanumerosta ja asennusajankohdasta. Näitä tietoja käytetään identifioidessa asiakastietokoneita.

Taulun kentät WORKSTATIONDATASTATEDATAID ja WORKSTATIONDATACOMPANYDATAID määrittelevät asiakastietokoneen nykyisen tilan sekä liittävät asiakastietokoneen asiakasyritykseen.

Kolmantena dynaamisena komponenttina määritellään käyttäjätietoja ylläpitävä taulu USERDATA.



Kuva 13. Käyttäjätietoja kuvaava taulu

Järjestelmään luodaan uusi käyttäjä joko Microsoft Windows -käyttöjärjestelmään kirjautuneen käyttäjän tiedoista tai vaihtoehtoisesti manuaalisesti syöttämällä uusi käyttäjä graafisesta hallintajärjestelmästä.

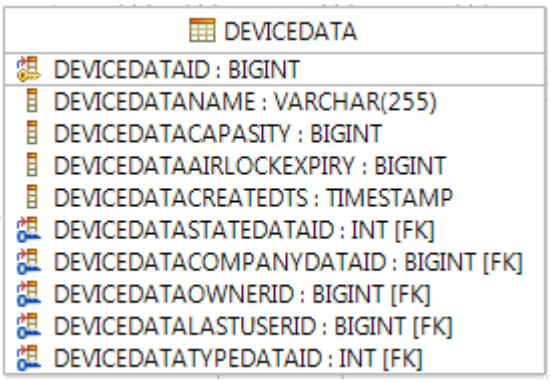
Luodun käyttäjän tiedot pitävät sisällään käyttäjätunnuksen, USERDATANAMEn, jota käytetään kirjauduttaessa hallintajärjestelmään sekä kryptatun salasanan USERDATAPASSWORD, jota verrataan käyttäjän kirjautuessa antamaan salasanaan. Jos järjestelmä asetetaan käyttämään ulkopuolista palvelua käyttäjien autentikointiin, ei USERDATAPASSWORD -kentällä ole järjestelmän kannalta merkittävää toiminnallisuutta.

Muita asetettavia attribuutteja ovat USERDATAFULLNAME, joka on käyttäjän kirjautumisen jälkeen käyttöliittymässä näytettävä nimiteksti. Tämä ei ole varsinaisesti käyttäjän erisnimi, vaikka voi sisältää senkin, vaan ajatuksena on enemmänkin *user@company* -tyylinen kooste. Tämän kentän toteutus sekä tarpeellisuus ovat järjestelmää suunniteltaessa vielä auki.

USERDATACREATEDTS on käyttäjän luomisen hetkellä asetettu aikaleima ja USERDATAACCESSTS on käyttäjän viimeisimmän järjestelmään kohdistuneen operaation yhteydessä asetettu aikaleima. USERDATADEVICECOUNT on laskuri käyttäjän laitteista. Tätä attribuuttia hyödyntämällä voidaan rajata käyttäjän laitteiden lukumäärää.

Käyttäjälle määritellään tilatieto ja asiakasyritys samoin kuin esimerkiksi asiakastietokone liitetiin asiakasyritykseen ja tilatietoon. Näiden lisäksi käyttäjälle määritellään rooli käyttäjähierarkiassa. Tämä tapahtuu USERDATAROLEDATAID -kentän määrittelemisellä, joka on viite ROLEDATA -taulun numeeriseen avainindeksiin.

Seuraavana dynaamisena komponenttina määritellään itse suojattava laite. Järjestelmän piirissä toimivaa laitetta kuvaa taulu DEVICEDATA.



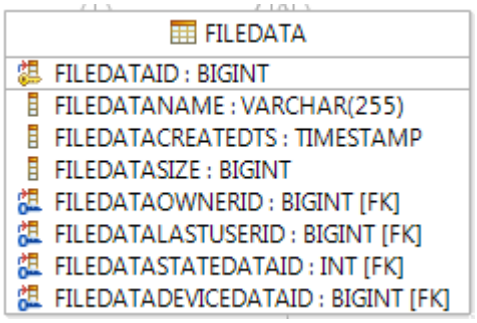
Kuva 14. Laitetietoa kuvaava taulu

Kun järjestelmän piirissä olevalle asiakastietokoneelle tuodaan suojaamaton massamuistilaite, tarjotaan käyttäjälle mahdollisuus suojata laite tai käyttää sitä vain sisällön lukemiseen oikeuttavassa tilassa. Kun laite halutaan suojata luodaan uusi laitetieto DEVICEDATA -tauluun. DEVICEDATANAME on suojattavan laitteen nimi, joka saadaa käyttöjärjestelmältä. DEVICEDATACAPACITY on laitteen tallennuskapasiteetti. DEVICEDATACREATEDTS on aikaleima, joka asetetaan laitteen suojaamisen hetkellä.

Suojaamisen yhteydessä laitteelle asennetaan erillinen ohjelma Airlock™. Tämä ohjelma on eräänlainen säiliö, johon voidaan tallentaa tiedostoja, jotka eivät päädy järjestelmän suojausoperaatioiden piiriin. Laitetiedoissa kenttä DEVICEDATAAIRLOCKEXPIRY on numeerinen arvo, joka kuvaa säiliöön vietyjen tiedostojen elinikää sekunteina. Kun tämä aika on kulunut, tiedostot tuhoutuvat automaattisesti. Airlock™-ohjelma ei kuulu tämän insinöörityön piiriin, eikä sen toimintaa käsitellä tarkemmin.

Laitetiedoille määritellään asiakasyritys ja tilatieto samoin kuin käyttäjälle ja asiakastietokoneelle. Lisäksi laitetiedolle määritellään laitteen tyyppi, omistajakäyttäjä sekä viimeisin käyttäjä. Omistajakäyttäjä on laitteen suojannut käyttäjä. Viimeisin käyttäjä järjestelmän piiriin määritelty käyttäjä, joka on viimeiseksi hyödyntänyt mahdollisuuttaan käyttää laitteen suojattua materiaalia.

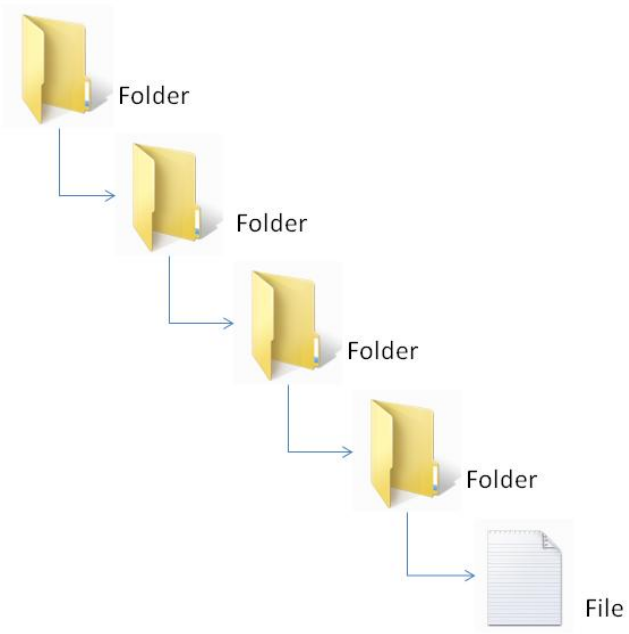
Viimeinen järjestelmän dynaaminen komponentti on järjestelmän piirissä toimivien laitteiden sisältäviä tiedostoja kuvaava taulu, FILEDATA.



kuva 15. Tiedostoinformaatiota kuvaava taulu

Kun järjestelmän piirissä toimivaan laitteeseen viedään tiedosto, siitä luodaan automaattisesti tiedosto-objekti tietokantaan.

FILEDATANAME on hakemistopolku, joka määrittelee tiedoston sijainnin laitteen tietostojärjestelmässä. Kuvan 16 mukaisessa tilanteessa FILEDATANAME muodostuisi muotoon ”\Folder\Folder\Folder\Folder\File”.



Kuva 16. Tiedostopuu

FILEDATACREATEDTS on aikaleima, joka asetetaan tiedoston saapuessa järjestelmän suojauksen piiriin. FILEDATASIZE on tiedoston koko nykyhetkellä. Tiedostoa kuvaavalle tiedolle määritellään asianmukaiset tilatiedot periyttämällä ne tiedoston luoneelta käyttäjältä ja laitteelta, jossa tiedosto nykyhetkellä sijaitsee. FILEDATAOWNERID osoittaa tiedoston luoneen käyttäjän ja asiakasyrityksen jonka vaikutusalueeseen tiedosto kuuluu. FILEDATALASTUSERID osoitaa puolestaan tiedostoa viimeisen toteutetun tiedosto-operaation käynnistäneen käyttäjän.

FILEDATADEVICEDATAID on järjestelmän piirissä toimivan laitteen yksilöivä tunniste. Tämä yhdistää tiedoston laitteeseen, jossa tiedosto nykyhetkellä sijaitsee. Tätä yhteyttä hyödyntämällä kopioidaan laitteen tilatieto tiedoston tilaa kuvaavaan FILEDATASTATEDATAID -kenttään. Tällä mallilla käyttäjä tilassa ”ACTIVE” voi käyttää tiedostoa normaalisti, mutta ”BLOCKED” -tilassa oleva käyttäjä ei (kuva 9).

2.3.4 Järjestelmän lokitietojen tallentaminen tietokantaan

Edellä esitetyt taulukokonaisuudet muodostavat mallin, jonka avulla järjestelmän objekteja hallitaan reaaliaikaisesti. Toinen tietokannasta riippuva huomattava osakokonaisuus järjestelmän toiminnassa on lokitus.

Järjestelmästä kerätyn tiedon tallentamista lähestytiin periaatteella, jossa kaikki tieto lokitetaan tietokantaan. Tästä periaatteesta johtuen tietokannan lokitietoja sisältävät taulut kasvavat ajan kuluessa massiivisiin kokoluokkiin ja tämä tulee vaikuttamaan järjestelmän suorituskykyyn. Tätä insinöörityötä kirjoitettaessa on vielä avoinna lopullinen toimintamalli, jolla tämä ongelma ratkaistaan, eikä tätä mahdollista tiedon arkistointimallia käsitellä tässä raportissa.

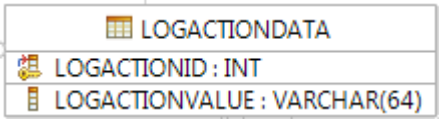
Tähän mennessä järjestelmän piiriin on määritelty viisi dynaamista komponenttia:

- asiakasyritys
- asiakastietokone
- järjestelmän käyttäjä
- tiedostoja sisältävä laite
- tiedosto.

Kun tarkastellaan edellämainittujen viiden komponentin määrityksiä, voidaan todeta asiakasyrityksen pysyvän usein muuttumattomana. Näin perusteita reaaliaikaiselle lokitukselle ei ole.

Muut neljä komponenttia järjestelmässä ovat hyvin usein muuttuvia ja täten rakennetaan lokitus tukemaan näiden objektien tietosisällön muuttumista.

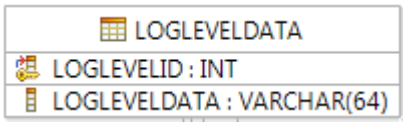
Ensimmäisenä määritellään kolme staattista taulua LOGACTIONDATA, LOGLEVELDATA ja LOGMESSAGEDATA.



Kuva 17. Tapahtunutta operaatiota kuvaava taulu

LOGACTIONDATA on kokoelma järjestelmän piirissä tapahtuneista operaatioista. Tällaisia operaatioita ovat esimerkiksi tiedoston tallennus, tiedoston poistaminen, massamuistilaitteen liittäminen järjestelmään, uuden suojatun laitteen luominen ja asiakastietokoneen liittäminen järjestelmään.

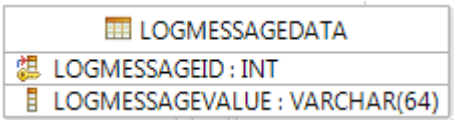
LOGLEVELDATA kuvaa tapahtuneen operaation tyyppiä.



kuva 18. Tapahtuneen operaation tyyppiä kuvaava taulu

Järjestelmän piirissä tapahtuva operaatio voi olla yksi kolmesta. Huomautus tapahtuneesta, varoitus tai virhe. Huomautus on järjestelmässä suoritettu operaatio, joka päättyi oletettuun lopputulokseen. Tämä voi olla esimerkiksi massamuistilaiteen liittäminen järjestelmään. Varoitus on järjestelmässä suoritettu operaatio, jonka lopputulos ei ollut oletettu. Virhe on operaatio, jota ei pystytä suorittamaan loppuun.

LOGLEVELMESSAGE on tapahtuneelle operaatiolle määritelty vastausteksti. Tätä tietoa käytetään rakennettaessa ihmiselle luettavaa lokitietoa näytettäväksi käyttöliittymässä.

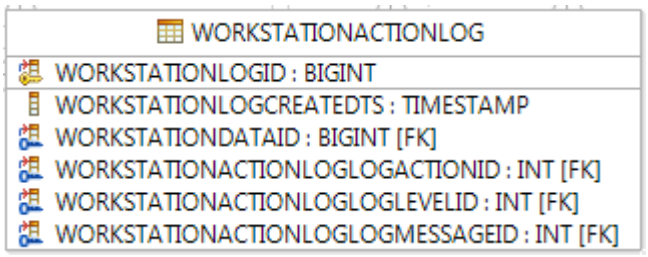


kuva 19. Operaatiolle määritellyn vastauksen sisältävä taulu

Käyttämällä edellä mainittuja kolmea taulua voidaan määritellä kaikki järjestelmässä tapahtuvat operaatiot osoittamalla kyseisistä tauluista tapahtunutta vastaavat tietueet. Tämän toimintamallin ideana on lokiin kirjoitettavan tiedon koon pakottaminen minimiin sekä estää tiedon redundanssia.

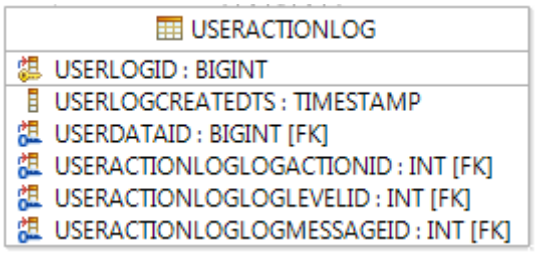
Käsiteltäessä massamuistilaitteita ja tiedostoja todetaan useiden operaatioiden olevan luonnostaan useasti toistuvia. Esimerkiksi massamuistilaitteen yhdistäminen asiakastietoneeseen tai tiedoston avaaminen muokattavaksi. Näistä operaatiosta luotava lokitettava informaatio sisältää paljon lähes identtistä tietoa. Jos kaikki tämä informaatio kirjoitetaan selväkielisenä tekstinä tietokantaan ajaudutaan nopeasti tilanteeseen, jossa tietokanta on kohtuuttoman suuri ja sisältää paljon duplikaatti-informaatiota. Kirjoitettaessa lokia käyttämällä edellä määriteltyjä taulurakenteita, kirjoitetaan tietokantaan kokonaisluku, jonka avulla viitataan haluttuun operaatioon. Näin minimalisoidaan tallennettavan tiedon koko ja vältetään saman merkkijonon toistuva tallentaminen.

Käyttämällä edellä määriteltyä lokitustekniikkaa toteutetaan malli järjestelmän lokituksesta. Järjestelmän lokituksesta aiheutuvan kuorman jakamisen tehostamiseksi eritellään lokitettavan tiedon tallentaminen neljään tauluun, joista jokainen edustaa yhtä järjestelmän dynaamista komponenttia.



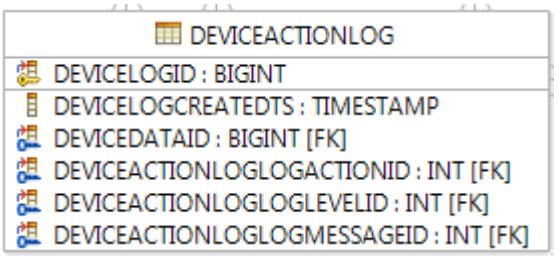
kuva 20. Asiakastietokoneen lokitaulu

Kaikki neljä lokitaulua ovat lähes identtisiä sekä toiminnaltaan että rakenteeltaan. Tällä tavalla tauluja on helppo operoida, ja niille pystytään tarvittaessa luomaan ihmiselle luettava esitysmuoto suoraan aikaisemmin esitetyistä staattisista tauluista, joihin lokitaulut viittaavat.



kuva 21. Käyttäjätiedon lokitaulu

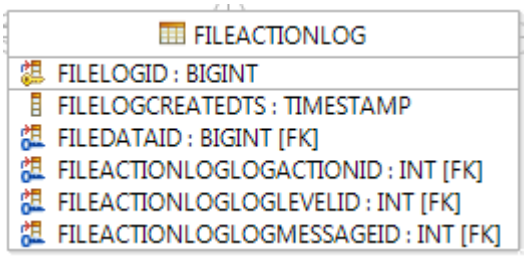
Jokaisen lokitaulun rakenne on seuraava: taulun ensimmäinen sarake on pääavaimena toimiva järjestelmän tuottama kokonaisluku, joka haetaan tietokannan sekvenssistä. Sekvenssi on tietokannassa sijaitseva erikoistyyppinen taulu, joka sisältää vain yhden rivin.



Kuva 22. Laitetiedon lokitaulu

Toinen sarake on tapahtuneen lokitedon aikaleima. Tarkennettuna tämä on tapahtumahetkellä tietokantaa ylläpitävän palvelimen päivämäärä ja kellonaika. Kolmas sarake määrittelee lokitettavan toimenpiteen toteuttaneen objektin. Tämä kenttä osoittaa lokitaulukon tyyppiä vastaavan dynaamisen komponentin informaatiota ylläpitävän taulun pääavaimeen.

Seuraavat kolme saraketta määrittelevät tapahtuneen lokitettavan operaation, tapahtuneen operaation tyypin sekä tapahtumasta johtuvan paluuviestin.



kuva 23. Tiedostotiedon lokitaulu

2.4 Ohjelmistotuotantomenetelmä

Projektin alati muuttuvista vaatimusmäärittelyistä johtuen perinteiset sovelluskehitysmenetelmät eivät sovellu käytettäviksi. Vaikka mitään erityistä ohjelmistotuotantomentelmää ei ole valittu, on ollut luonnollista toteuttaa ohjelmistokehitystä pienissä iteratiivisissa askeleissa, ja täten ajautua lähelle ketterien menetelmien määritelmää [20]. Tällä toimintamallilla on ollut helppo mukautua vaatimusmäärittelyiden muuttumiseen, ja ohjelmiston kehitys on näkynyt selkeästi pienien kokonaisuuksien toteutumisella. Tämä toimintamalli on myös mahdollistanut palautteen saamisen nopeasti, jolloin on ollut mahdollista reagoida esitettyihin ongelmiin sekä parannusehdotuksiin.

Tämä toimintaperiaatteen noudattaminen on todettu käytännölliseksi, koska järjestelmää on kehitetty yhdessä vanhan järjestelmän ylläpidon ohella. Tämä on mahdollistanut vanhan järjestelmän tutkimisen projektin eri vaiheissa sekä molemmista järjestelmäversioista saadun palautteen yhdistämisen yhdeksi kokonaisvaltaiseksi määrittelyksi. Tätä määrittelyä käytetään uuden järjestelmän suunnitelun perustana.

3 KEHITYSYMPÄRISTÖ

Järjestelmän kehitysympäristönä käytetään Ubuntu Linux 9.10 –versiota [21], koodinimeltään ”Karmic Koala”. Palvelin sijaitsee Nebula-palveluntarjoajalta vuokratulla virtuaalipalvelimella. Järjestelmän varmuuskopioinnin hoitaa palveluntarjoaja. Samalle palvelimelle on myös asennettu Apache-www –palvelin [22].

3.1 Kehitystyössä käytetyt työkalut

Järjestelmän kehitykseen käytetyt työkalut voidaan ryhmitellä käyttötarkoituksensa mukaisesti kahteen ryhmään:

- ohjelmistokehitykseen käytettävät työkalut
- tietokantasuunnitteluun käytettävät työkalut.

Järjestelmän versionhallinta on toteutettu käyttäen Subversion(SVN) [23] -ohjelmistoa. Envault Corporation käyttää Subversionia sovelluskehityksensä versionhallintaan, joten on käytännöllistä lisätä projektin tiedostot olemassa olevaan versionhallintaan. Näin hyödynnetään jo olemassa olevia autentikointimetodeja.

Subversionin rinnalla käytetään TortoiseSVN [24] -ohjelmaa helpottamaan työskentelyä versionhallinnan kanssa. TortoiseSVN on graafinen asiakastyökalu Subversion -versionhallintajärjestelmälle.

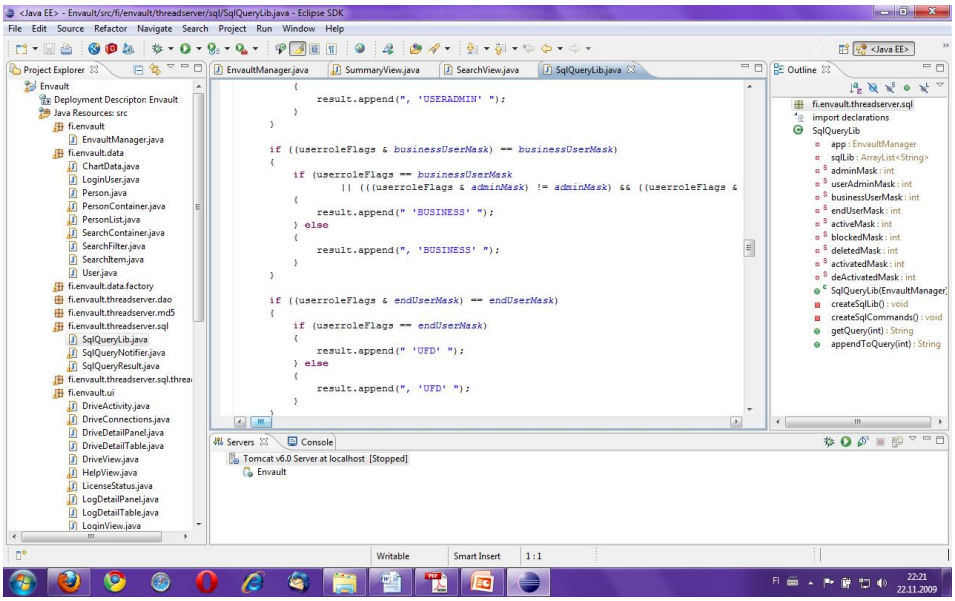
3.2 Ohjelmistokehitykseen käytettävät työkalut

Ohjemistokehityksen työkalut käsittävät ohjelmistot ja ohjelmistokomponentit joiden avustuksella luodaan itse ohjelmistokokonaisuus. Tähän kokonaisuuteen kuuluvat Ohjelmistokehitysympäristö Eclipse [25], Vaadin – sovelluskehys [4], JDBC-tietokanta-ajuri [26] sekä Open Flash Charts 2 -komponentti graafisten kuvaajien luomiseen.

3.2.1 Eclipse IDE

Eclipse on suuren suosion saavuttanut avoimen lähdekoodin ohjelmistokehitysalusta. Vuonna 1996 IBM osti kanadalaisen Object Technologies Internationalin kehittämään Visual Age ohjelmistotyökalujaan [32]. Tämä kehityshanke johti myöhemmin Eclipsen syntymiseen. Vuonna 2004 Eclipse Foundation perustettiin ja Eclipsen kehitys siirtyi IBM:n hallinnasta itsenäiselle säätiölle.

Eclipse on rakenteeltaan modulaarinen, joten ohjelmistokehittäjien on helppoa muokata siitä mieluisensa. Alunperin Eclipse suunniteltiin Java-sovellukehitykseen, mutta se on kasvanut ajan kanssa hyvin kattavaksi, monia eri ohjemointikieliä tukevaksi kokonaisuudeksi.



Kuva 24. Eclipse-sovelluskehitysympäristö

Eräs Eclipsen suosiota voimakkaasti edistänyt piirre on mahdollisuus kirjoittaa liitännäisiä, joilla mahdollistetaan sovelluskehitysympäristöön luotavat uudet toiminnot.

Tässä projektissa käytetään Vaadin-sovelluskehyksen Eclipse-liitännäistä luotaessa Java-ohjelmakoodista Javascript-komponentteja. Tarkemmin sanottuna projektissa käytetään Vaadin sovelluskehyksen Eclipse-liitännäistä luomaan asiakaspään javascript-komponentit luotujen www-palvelimella sijaitsevien java-komponenttien vastineeksi. Luotaessa nämä komponentit pakataan yhdeksi WAR (Web Application Achive) –tiedostoksi [27].

Web application archive on pakattu tiedostorakenne. Se on suunniteltu java-pohjaisten www-sovellusten levittämiseen.

3.2.2 Vaadin-sovelluskehys

Vaadin on suomalaisen IT Mill -ohjelmistotalon kehittämä sovelluskehys, joka tarjoaa rajapinnan rikkaiden web-sovellusten luontiin käyttäen pelkästään java-ohjelmointikieltä. Käytännön tasolla tämä muokkaa www-sovelluskehityksen huomattavasti lähemmäksi perinteistä työpöytäsovellusta. IT Mill aloitti sovelluskehyksensä kehitystyön vuonna 2000, ja sen lähdekoodi avattiin vuonna 2007 nimellä IT Mill toolkit. Nykyisen nimen Vaadin sai vuonna 2009. Vaadin-sovelluskehyksen perusidea on luoda java-ohjelmointikielellä toteutetusta sisällöstä palvelimen java-teknologiaan perustuva servlet-luokan toteutus sekä asiakkaan selaimessa toteutettava javascript-komponenttikokonaisuus.

Vaadin sovelluskehys toimii javan Servlet -ohjelmointirajapinnan päällä. Tällä sovelluskehyksellä toteutettu www-sovellus toimii java servlet -sessiona[13], joka käyttää *Terminal Adapter* -komponenttia hallitakseen kommunikaation asiakkaan käyttämän internetselaimen kanssa. Terminal Adapter on abstrahointikerros, jonka tehtävänä on käyttöliittymän komponenttien piirtäminen www-sivuksi sekä välittää asiakkaan selaimelta saapuva informaatio sovellukselle. Terminal Adapter käyttää käyttöliittymän piirtämiseen UIDL (User Interface Definition Language) –kieltä [28], jonka kommunikaatio tapahtuu JSON (Javascript Object Notation) [29] -formaatilla.

JSON-formaatti on kevyt ja helposti käsiteltävä tiedonsiirtoon suunniteltu formaatti, joka soveltuu erinomaisesti käytettäväksi javascript-ohjelmointi-kielen kanssa. JSON-formaatin määritelmä juontaa juurensa Ecma-262 versio 3 standardista [30].

Ohessa esitettävässä JSON-komponentissa määritellään kuvitteellisen pudostusvalikon sisältö.

```
{
 "valikko": "tiedosto,
 "komennot": [
 {
        "otsikko": "Uusi",
        "toiminto":"uusiDokumentti"
 },
 {
        "otsikko ": "Avaa",
        "toiminto ": "avaaDokumentti"
 },
 {
        "otsikko ": "Sulje",
        "toiminto ": "suljeDokumentti"
 }
 ]
}
```

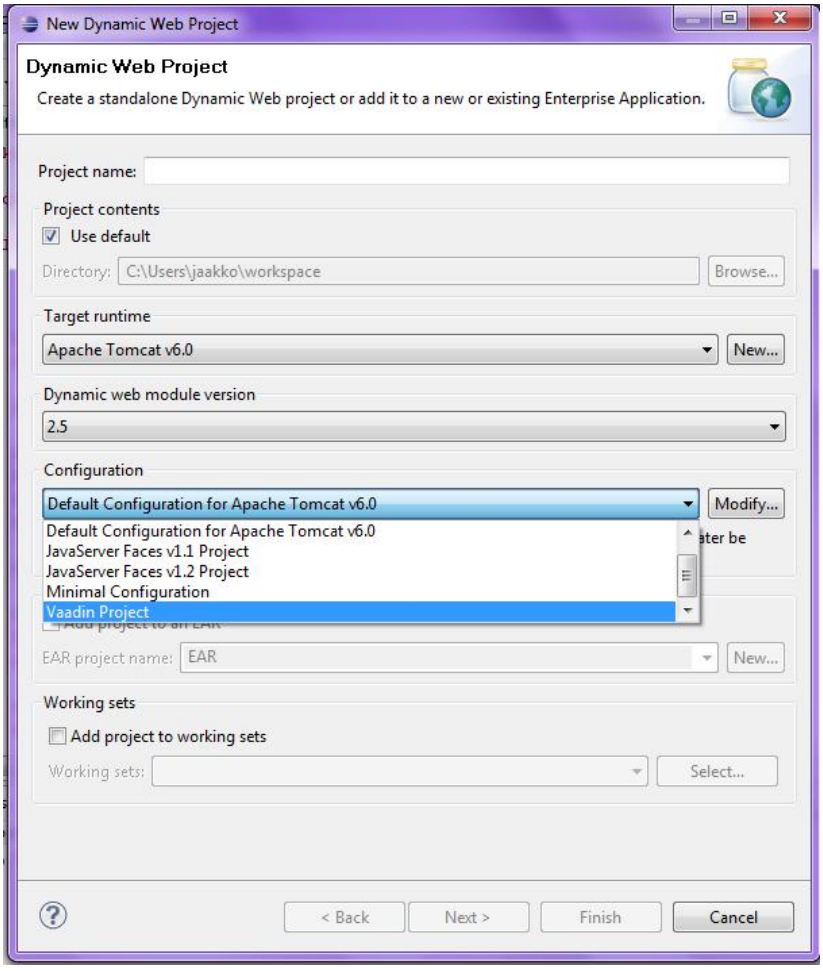
Asiakaspäässä vaadin tukeutuu Google Web Toolkit (GWT)[8] -sovelluskehykseen. GWT on hakukoneestaan tunnetun yhtiön Googlen valmistama kokoelma työkaluja, jotka ovat tarkoitettu www-käyttöliittymien toteuttamiseen.

Vaadin sovelluskehyksen käyttöönotto vaatii sovelluskehyksen java-kirjaston lataamisen koneelle jolla aiotaan työskennellä. Tämä on .jar-tiedosto, joka on ladattavissa sovelluskehyksen virallisilta www-sivuilta. Sivusto tarjoaa mahdollisuuden ladata joko pelkkä .jar-tiedosto tai esittelypaketti, joka sisältää sovellukehyksen kirjastotiedoston sekä esimerkkejä sen käytöstä.

Vaikka Vaadin sovelluskehyksen käyttö ei varsinaisesti vaadi sille luodun Eclipse-liitännäisen käyttöä, sen käyttö helpottaa Vaadin-projektin hallintaa ja tässä projektissa sitä käytetään oletuksena. Uusi Vaadin-projekti Luodaan Eclipse-sovelluskehitysympäristössä valitsemalla ”File” alavetovalikosta

”New” ja seuraavaksi ”Other”. Näin aukevasta ikkunasta valitaan kansio ”web” ja tuplaklikataan ”Dynamic Web Project”.

Seuraavaksi aukeavasta ikkunasta valitaan ”Configuration” kohdasta ”Vaadin Project”, annetaan projektille nimi ja valitaan ”Finish”.



Kuva 25. Vaadin-projektin luonti

Vaadin sovelluskehys luo projektin rungon jossa sijaitsee yksi java-luokka, joka on nimetty muotoon *<Annettu Projektin nimi>Application.java* ja kirjoittaa valmiiksi yleisimmät, projektin suorittamisen kannalta tarvittavien ohjelmistokomponenttin määrittelyt.

Tämä generoitu luokka on Vaadin-projektin pääluokka, joka edustaa ohjelman suorittamisen alkupistettä.

*TestprojectApplication.java

```
package com.example.testproject;

import com.vaadin.Application;
import com.vaadin.ui.*;

public class TestprojectApplication extends Application
{
    @Override
    public void init()
    {
        Window mainWindow = new Window("Testproject Application");
        Label label = new Label("Hello Vaadin user");
        mainWindow.addComponent(label);
        setMainWindow(mainWindow);
    }
}
```

Kuva 27. Vaadin sovelluskehyksen luoma tiedosto.

Kuvassa 27 esitetään minimalistinen sovellus, joka tulostaa käyttäjän internet selaimeen tekstirivin ”Hello Vaadin User”.

Tämän projektin käyttöliittymäksi luodun Vaadin-sovelluksen kannalta keskeisiä komponentteja ovat

- com.vaadin.Application
- com.vaadin.ui
- com.vaadin.data
- com.vaadin.event.

com.vaadin.Application

Application-luokka on jokaisen Vaadin-sovelluskehyksellä toteutetun www-sovelluksen perusta, jossa keskeisimmiksi toiminnoiksi nousevat Init()- sekä SetMainWindow()-metodit.

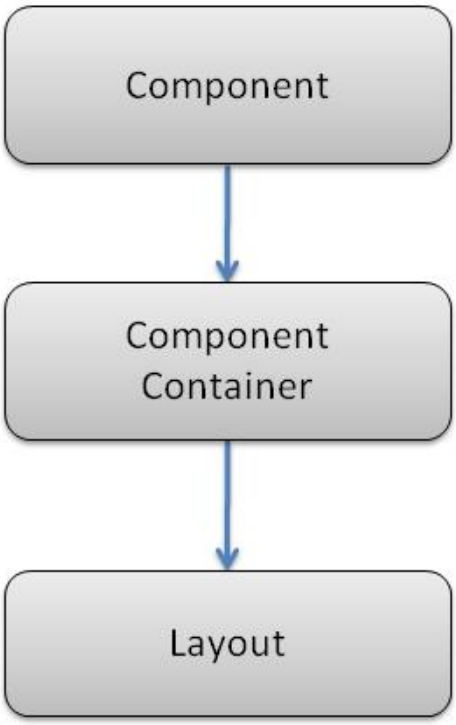
Application on abstrakti luokka [31], joten jokaisen Vaadin-sovelluksen on peritettävä pääluokkansa Application-luokasta ja toteutettava Init()-metodi. Tämä kyseinen suoritetaan aina kun on tarvetta käynnistää uusi instanssi vaadin-sovelluksesta. Toinen pakotettu toimenpide on SetMainWindow()-metodi, jolla asetetaan sovelluksen pääikkuna. Käynnistyessään Vaadin-sovellus tarvitsee ainakin yhden ikkunan, pääikkunan. Jos sovellus koostuu vain yhdestä ikkunasta, on tämä automaattisesti sovelluksen pääikkuna.

com.vaadin.ui

Tämä paketti on Vaadin sovelluskehyksen kokoelma käyttöliittymän kokoamiseen tarkoitetuista komponenteista. Jokaisen käyttölliittymäkomponentin, jonka on tarkoitus toimia Vaadin-sovelluskehyksellä toteutetussa www-sovelluksessa on toteutettava *Component*-rajapinta, jonka kautta järjestelmä määrittelee käyttöliittymän perustoimintoja ohjaavan metodiikan, joihin kuuluu esimerkiksi komponentin liittäminen sovellukseen ja komponentin asettaminen näkyväksi tai näkymättömäksi.

Seuraava komponentti rajapintahierarkiassa on *ComponentContainer,* joka täydentää Component-rajapintaan toiminnallisuuden sisältää muita komponentteja.

Hierarkian viimeinen toteutus Layout täydentää osaltaan rajapintahierarkiaa komponentin sijainnin konseptilla ja mahdollistaa komponentteja sisältävälle komponentille näiden sijoittelun.

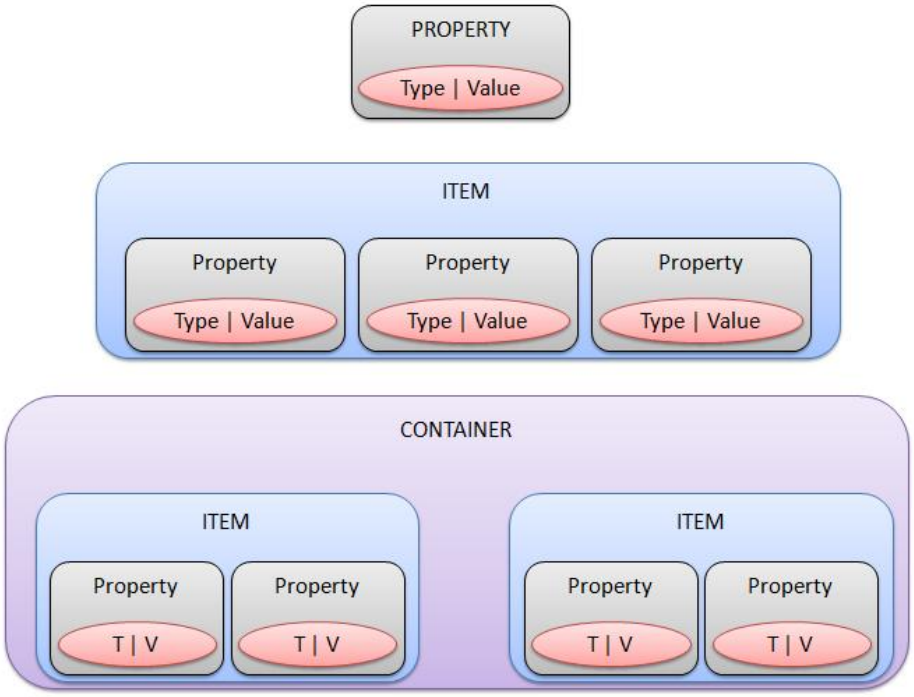


kuva 28. com.vaadin.ui paketin päärajapinnat

Component-rajapinnan oletustoteutus on AbstractComponent-luokka, joka on puolestaan perustoteutus suurimmalle osaa Vaadin-sovelluskehyksen käyttöliittymäkomponenteista.

com.vaadin.data

Tämän paketin sisältö on kokoelma tiedon hallintaan ja validointiin tarkoitettuja komponentteja. Com.vaadin.data määrittelee käsiteltävän tiedon kolmiportaisena loogisena kokonaisuutena.



Kuva 29. com.vaadin.datan informaatiomalli

Property kuvaa loogisesti pienintä mahdollista tiedon tallennusyksikköä. Tämä on järjestelmän näkemys matalan tason tietotyypeistä, esimerkiksi kokonaisluvuista.

Tosin on huomattava, että property voi olla minkä tahansa java-ohjelmointikielen Object-luokasta periytetyn komponentin ilmentymä, joten se ei kuvaa universaalisti määriteltyjä ohjelmointikielien tunnistamia alkeistietotyyppejä, vaan korkealta abstraktiotasolta katsottaessa määriteltävän tietorakenteen hierarkiassa matalimmalla sijaitsevaa informaatiota ylläpitävää yksikköä. Sen määritelmään kuitenkin kuuluu kapasiteetti pitää sisällään yksi, ja vain yksi arvo, joten se on usein jokin universaali alkeistietotyyppi.

Property-tietorakenteella on ominaisuuksinaan sen sisältämän informaation tyyppi, varsinainen informaatio, sekä mahdollisuus lukita property-tyyppinen tieto vain lukuoperaatiota hyväksyväksi.

Tietorakennemääritelmän toinen taso *item* kuvaa kokoelmaa property-tyyppisestä informaatiosta. Tätä tietorakennetta voidaan ajatella esimerkiksi Java-teknologian JavaBean [32] -rakenteena. Myös tietokannasta noudetun informaation käsittelyssä yksi rivi on yksi uniikki item-tietorakenne. Item-rakenteen keskeinen tehtävä informaation käsittelyssä on ryhmitellä kokoelma property-tyyppisiä tietorakenteita yhden uniikin tunnisteen taakse, jonka avulla voidaan hyödyntää juuri tiettyä informaatiokokoelmaa.

Viimeinen taso *Container* kuvaa järjestelmän tietorakenteen määritelmässä kokoelmaa organisoidusta informaatiosta. Tämän rakennetta voidaan havainnollistaa Java-ohjelmointikielen kontekstista esimerkiksi geneerisellä ArrayList-listarakenteella, koska Container asettaa sisältämilleen Item-tyyppisille komponenteille hyvin samankaltaiset rajoitukset. Tämän rakenteen sisältämämien item-komponenttien tulee kaikkien sisältää sama määrä property tyyppisiä rakenteita ja näiden tulee olla samaa tyyppiä sekä kaikki item-tyyppiset rakenteet tulee sisältää uniikki tunniste. Huomattavaa tosin on, että vaikka Container-rakenteen sisältämät item-rakenteet ovat uniikkeja, eivät ne ole välttämättä indeksoituja tai järjestettyjä.

com.vaadin.event

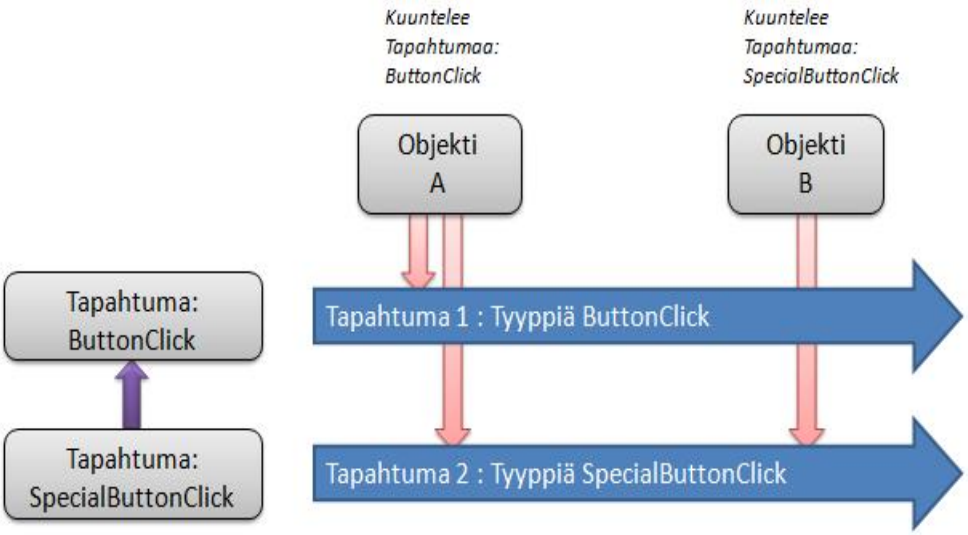
Event-paketin sisältö on keskittynyt sovelluskehyksen komponenttien välisten toiminnallisuuksien hallitsemiseen. Paketin keskeisimmät komponentit ovat periytettävä tapahtumankäsittelymalli sekä EventRouter-mekanismi.

Käytännön toteutuksen tasolla Vaadin-sovelluskehyksen tapahtumienkäsittelymalli on lähellä Java-teknologian vastaavaa toimintamallia. Tuotettaessa ohjelmakoodia lisätään tapahtumakuuntelijoita luotuihin komponentteihin joiden halutaan toteuttavan toiminnallisuuksia. Näiden tapahtumakuuntelijoiden vastakappaleiksi tuotetaan metodeita, jotka reagoivat tapahtuneisiin toimenpiteisiin. Seuraavassa koodiesimerkissä luodaan yksinkertainen Vaadin-sovellus, joka sisältää yhden painikkeen ja siihen liitetyn metodin.

```
package com.example.testproject;

import com.vaadin.Application;
import com.vaadin.ui.*;

public class ButtonApplication extends Application implements ClickListener
{
  @Override
  public void init()
  {
    Window mainWindow = new Window("Test");
    Button button = new Button("click!");
    button.AddListener((Button.ClickListener)this)
    mainWindow.addComponent(button);

    setMainWindow(mainWindow);
  }
  public void buttonClick(button.ClickEvent event)
  {
    Button source = event.getButton();
    if(source == button)
    {
      getMainWindow().showNotification(“Button Clicked!”);
    }
  }
}
```

Eräs Vaadin-sovelluskehyksen tapahtumienkäsittelymallin kehittyneemmistä ominaisuuksista on tapahtumien periyttäminen.

Tämä toimintamalli mahdollistaa tapatumien hallinnan hierarkisesti sekä valikoiden. Käytännössä tapahtumien periyttäminen tarkoittaa toimintamallia jossa objektille asetettu tapahtumakuuntelija vastaanottaa sille määritellyt tapahtumat sekä näistä tapahtumista periytetyt tapahtumat.



kuva 30. Periytyvä tapahtumienkäsittely

Tätä toimintamallia hyödyntämällä sovelluskehittäjä pystyy vaivattomasti hallitsemaan tapahtumia loogisesti ryhminä sekä eriyttämään tietyille halutuille tapahtumille yksittäisiä toiminnallisuuksia.

3.2.3 JDBC-Ajuri ja tietokantayhteydet

JDBC-ajureilla tarkoitetaan java-ohjelmointikielle tarkoitettuja tietokanta-ajureita, joiden avulla mahdollistetaan tietokantayhteyksien muodostaminen.

Kuten edellä on mainittu, projektin tietokantana käytetään PostgreSQL-tietokantaa, joten myös tarvittu jdbc-ajuri on kyseisen tietokantavalmistajan tarjoama ajuri.

Java-ohjelmointikielessä yleinen malli tietokantayhteyksien hyödyntämiseen on ConnectionManager-, Connection-, Statement-, sekä ResultSet-luokkien käyttö. Alla esitetyssä esimerkissä luodaan kuvitteellinen tietokantayhteyttä hallitseva luokka.

```
public class Dao
{


private String dbSrv;
private String dbUsr;
private String dbPw;
private String dbName;
private String dbPort;
private Connection connect;
private Statement state;

public Dao(String dbSrv,String dbUsr,String dbPw,String dbName,String dbPort)
{
  this.dbServer = dbServer;
  this.dbUser = dbUser;
  this.dbPass = dbPass;
  this.dbName = dbName;
  this.dbPort = dbPort;
}

public void connect()
{
  try
  {
    Class.forName("org.postgresql.Driver").newInstance();
    connect = DriverManager.getConnection("jdbc:postgresql://"
    + dbServer + ":" + dbPort + "/" + dbName, dbUser, dbPass);
    state = connect.createStatement();
  }
  catch (Exception ex)
  {
    System.out.println("Exception: " + ex);
  }
}

public void exit()
{
  try
  {
    connect.close();
  }
  catch (Exception ex)
  {
    System.out.println("Exception: " + ex);
  }
}
```

Oheinen esimerkki luo tietokantayhteyden, kun luokasta luodaan olio käytettäväksi. Class.forName()-metodi alustaa halutun ajurin käyttöön, jonka

jälkeen luodaan Connection-luokan ilmentymä käytten DriverManager-luokan getConnection-metodia. Tälle metodille annetaan parametrinä merkkijono, jonka perusteella luodaan varsinainen yhteys halutulle tietokantapalvelimelle. Viimeisenä operaationa luokan konstruktorissa luodaan Statement.luokan ilmentymä, joka on tietokantayhteyden suorittava komponentti.

Kyseisen esimerkin mukaisessa järjestelmässä itse tietokantakysely tapahtuisi käyttämällä yllä esitetyn esimerkin *state*-komponenttia seuraavasti.

```
ResultSet rs = state.executeQuery(“select * from users”);

while(rs.next())
{
  rs.getString(“username”);
}
```

Tietokantaoperaation onnistuneen suorituksen jälkeen state-luokan ilmentymä palauttaa ResultSet-luokan ilmentymän, joka sisältää tietokannasta noudetun informaation.

Tietokantayhteyden käyttö Vaadin-sovelluskehyksessä

Suunniteltavassa järjestelmässä toteutaan tiedon noutaminen ja tallentaminen edellä esitetystä Java-ohjelmointikielen yleisen toteutuksen esimerkistä eroavalla tavalla. Hyödynnetään Vaadin-sovelluskehyksen QueryContainer-luokan ilmentymää, joka on kuvan 29 mukaisen Container-rajapinnan toteuttava, erityisesti relaatiotietokannasta haettavan tiedon tallentamiseen suunniteltu toteutus.

QueryContainer esittää indeksoitua ja järjestettyä tietorakennetta, joka luodaan jdbc-ajurin avulla toteutetusta Connection-luokan ilmentymästä. Tämän tietorakenteen sisältö luodaan tietokantaoperaation seurauksena rakennetun ResultSet-luokan ilmentymän sisältämästä informaatiosta. Esimerkkinä QueryContainer-luokan ilmentymän luonti, joka samalla toteuttaa tietokantaoperaation.

```
Class.forName("org.postgresql.Driver").newInstance();

connect = DriverManager.getConnection("jdbc:postgresql://"
    + dbServer + ":" + dbPort + "/" + dbName, dbUser, dbPass);

QueryContainer myQc = new QueryContainer(“select * from users”,connect);
```

Vaadin-sovellukehyksessä Container-rajapinnan toteuttavien luokkien suurin hyötyarvo saadaan niiden sitomisesta järjestelmässä luotuihin informaatiota hyödyntäviin komponentteihin.

Useat ohjelmointikielet tukevat konseptia dynaamista informaatiota esittävästä rakenteesta sekä sille informaatiota tarjoavasta *DataSource*-komponentista, joka sidotaan sitä tarvitsevaan komponenttiin.

Esimerkkinä voidaan käyttää relaatiotietokannasta saatua informaatiota. Tämä informaatio on tallennettu tietokantatauluihin joten hyvin useassa tapauksessa tietokannasta noudettu informaatio on myös loogiselta mallitaan taulu. Tarkastelemalla yksinkertaista kuvitteellista tietokantakyselyä

```
“select * from users”
```

voidaan todeta tämän kyselyn palauttavan kaiken informaation tietokannan taulusta, jolle on annettu nimeksi *users.* On mahdotonta päätellä, kuinka paljon informaatiota tämä tietokantaoperaatio palauttaa, tai erityisesti sen rakennetta, tuntematta tietokantaa, josta kyseinen informaatio on noudettu. Konsepti tiedon sitomisesta niitä tarvitseviin komponentteihin poistaa rajoitteen informaation rakenteen tuntemisesta.

Alla esitetty esimerkki luo tietokantayhteyden, noutaa kuvan 13 mukaisesta userdata-taulusta kaikki käyttäjät QueryContainer-luokan ilmentymän avulla, sitoo ne informaatiota esittävään Table-komponenttiin ja palauttaa kutsuttaessa juuri luodun Table-luokan ilmentymän.

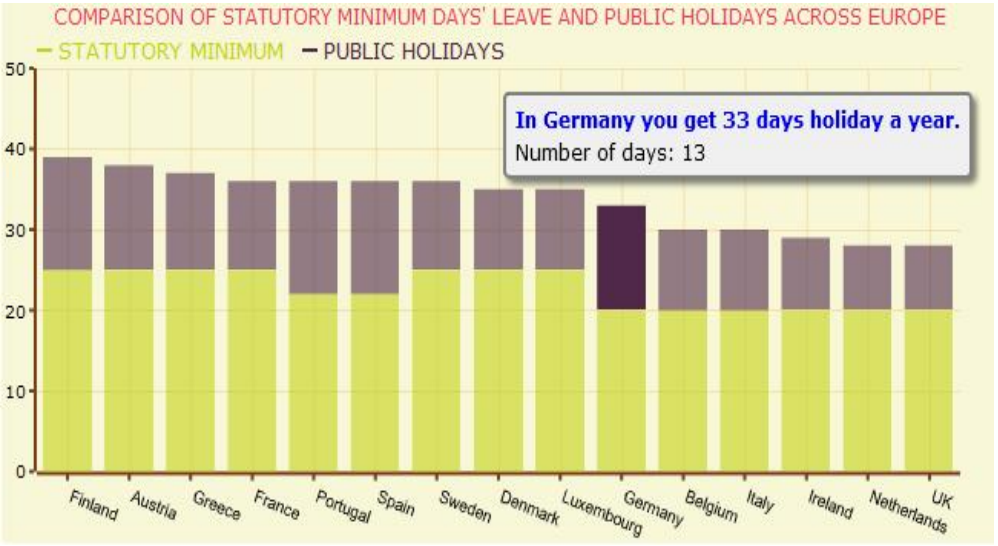
```
public Table connectAndCreateTable()
{
  try
  {
    Class.forName("org.postgresql.Driver").newInstance();

    connect = DriverManager.getConnection("jdbc:postgresql://"
    + dbServer + ":" + dbPort + "/" + dbName, dbUser, dbPass);

    QueryContainer qc = new QueryContainer(“select * from userdata”,connect);
  }
  catch (Exception ex)

  return new Table(“My User Table”, qc);
}
```

Tämä toimintamalli antaa mahdollisuuden eriyttää suunniteltavan järjestelmän käyttöliittymä sen informaatiota hallitsevasta rakenteesta. Useissa järjestelmissä tietokanta itsessään ei turvallisuuden takia salli suorien SQL()-kyselyiden suorittamista, vaan vaatii sille määriteltyjen funktioiden tai etukäteen määriteltyjen tietokantanäkymien käyttämisen informaation noutamiseen.

Toinen merkittävä hyöty tiedon sitomisessa on sen uudelleenkäytettävyydessä. Informaatio, joka on mahdollista sitoa sitä tarvitseviin komponentteihin dynaamisesti on myös mahdollista erottaa niistä tai korvata se toisella informaatiolla. Koska Vaadin-sovelluskehityksen avulla toteutettu sovellus perustuu Java servlet-teknologiaan, pätee siihen myös Java servlet-instanssin elinkaaren määritys ja täten kerran luotu objekti on olemassa, kunnes se tuhotaan. Hyödyntämällä tätä ominaisuutta on mahdollista käyttää olemassa olevan servlet-instanssin jo olemassa olevia komponentteja.

3.2.4 Open Flash Chart 2

Open Flash Chart 2[OFC2] on avoimen lähdekoodin työkalu erilaisten graafisten esitysten luomiseen.



kuva 32. Open Flash Chart 2

OFC2 on Flex-ohjelmointikielellä luotu flash komponentti, joka tuottaa näytettävän graafiset esitykset sille annetun JSON-tiedoston perusteella.

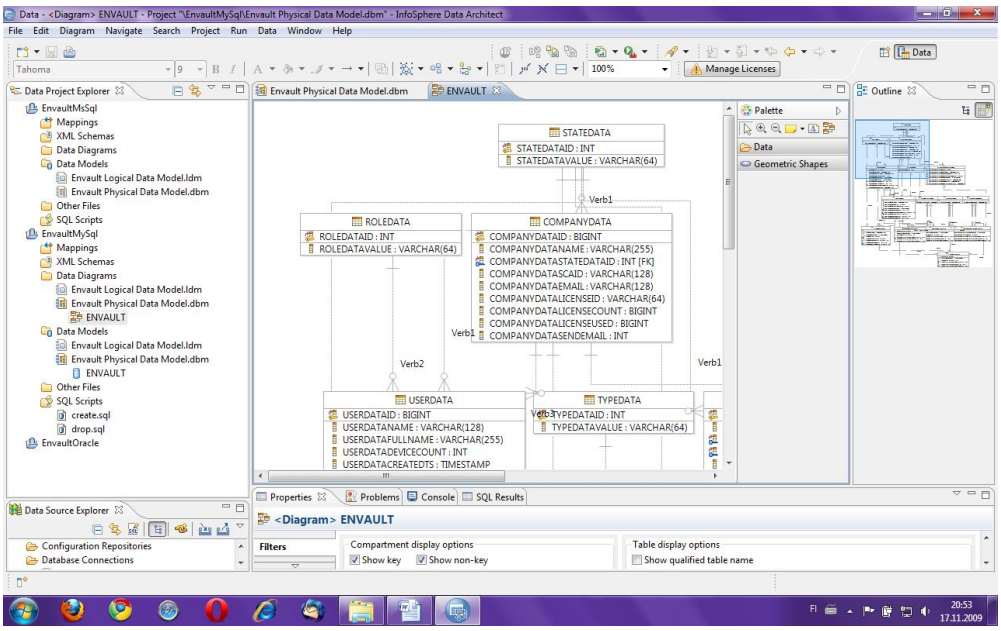
3.3 Tietokantasuunnitteluun käytettävät työkalut

Tietokantasuunnittelussa käytettävien ohjelmistojen tarkoituksena on suunnitella tietokannan rakenne ja luoda tätä rakennetta kuvaava mallitiedosto, jota yleisesti kutsutaan englanninkielisellä termillä *schema* [19]. Tätä rakennetta kuvaavaa mallitiedostoa käytetään alustamaan tietokantapalvelimelle suunnitelman mukainen rakenne.

Eri valmistajien tarjoamilla tietokantaratkaisuilla mallitiedostojen rakenne voi olla hyvinkin erilainen.Tässä projektissa käsitteellä mallitiedosto ja schema tarkoitetaan sql-kielellä toteutettua tiedostoa, joka sisältää sql-lausekkeita tietokannan komponenttien luontia varten.

3.3.1 IBM Data Architect

IBM:n tarjoama Data Architect on graafinen tiedon mallintamisohjelmisto joka tarjoaa laajan valikoiman työkaluja eri tietokantavalmistajille tarkoittetujen tietokantamallien luomiseen.



kuva 33. IBM Data Architect

Data Architectin avulla on mahdollista luoda tietokannan fyysinen malli aikaisemmin luodusta loogisesta mallista, olemassa olevasta mallitiedostosta tai toiminnassa olevasta tietokannasta.

3.3.2 SQLFairy

SQLFairy on komentorivipohjainen, Perl-ohjelmointikielellä toteutettu avoimen lähdekoodin ohjelmisto, jonka avulla voidaan luoda valmistajakohtaisesta tietokantaa kuvaavasta mallitiedostosta toisen tietokantaratkaisun hyväksymä mallitiedosto. Alla esitetty komento kääntää Oracle-tietokannalle suunnitellun mallitiedoston PostgreSQL:n vaatimaan muotoon.

```
sqlt -f Oracle -t PostgreSQL oracle.sql > pgsql.sql
```

Projektissa käytetään SqlFairy-ohjelmiston tarjoamia toiminnallisuuksia täydentääksemme IBM Data Architect:n tarjoamia toiminnallisuuksia, kuten esimerkiksi yllä esitettyä mallitiedoston käännöstä.

4 TULEVAT TAVOITTEET

Projektin tulevaisuuden tavoitteet koostuvat ensisijaisesti tuen luomisesta eri tietokantavalmistajien tarjoamille tietokantajärjestelmille sekä käyttöliittymän ulkoasun kohentamisesta.

Nykyinen järjestelmä on suunniteltu toimimaan Linux-käyttöjärjestelmää käyttävällä palvelimella ja täten projektin looginen etenemissuunta on Microsoft Windows Server-käyttöjärjestelmää sekä Microsoft SQL Server-tietokantapalvelinta hyödyntävä konseptitason suunnitelma.

Käyttöliittymän osalta projektin seuraava osio käsittää asiakkaalle mahdollisuuden muokata käyttöliittymän ulkoasua henkilökohtaisemmaksi käyttämällä ennalta määriteltyjä käyttöliittymäteemoja tai luoda kokonaan oman teemansa.

Huomion arvoisena pidetään myös mahdollisuuksia, jotka avautuvat massamuistilaitteiden yleistymisestä kuluttajaeletroniikan osana. Tulevaisuuden suunnitelmiin kuuluu mobiililaitteiden massamuistisisällön suojaaminen Envault Corporationin tarjoamalla teknologialla sekä mobiililaitteiden tarjoamien erityisominaisuuksien, kuten paikannuksen tarjoavien palveluiden hyödyntäminen.

5 YHTEENVETO

Tämä insinöörityö esittää tiedonhallintajärjestelmän suunnitteluprosessin keskeisimmät osa-alueet. Tutkinnan kohteena ovat konseptit, jotka johtivat valittuihin ratkaisuihin, niiden toteuttamiseen käytettävät ohjelmistot ja ohjelmistokomponentit, sekä kooste projektin tulevaisuuden haasteita, joiden pohjalta kehitystyö jatkuu vuonna 2010. Projektin keskeisimpiä komponentteja olivat Java-teknologiaan perustuva Vaadin-sovelluskehys web-sovellusten luontiin, Apache Tomcat Servlet Container Java-Servlet teknologiaan perustuvien web-komponenttien toteuttamiseen, sekä PostgreSQL-tietokantajärjestelmä informaation hallintaan. Tavoitteena oli luoda suunnitelma, jonka pohjalta kehitystyö aloitetaan sekä rajallisen toiminnallisuuden toteuttava testiversio.

Osana suunnitteluprosessia tarkasteltiin yleisesti informatiivisen järjestelmän tietokannan suunnitteluun kuuluvia prosesseja. Projektin suunnitelman mukainen tietokanta on tämän insinöörityön valmistumisen hetkellä testikäytössä ja sen suorituskyvystä on saatu positiivisia tuloksia.

Järjestelmän käyttöliittymän testiversion perusrakenne tuotettiin tasolle, jolla sen ulkoasua voitiin tarkastella ja miettiä jatkokehityksen suuntaa. Järjestelmälle asetettujen toiminnallisuusvaatimuksien osalta käyttöliittymä on kuitenkin tämän työn valmistumisen hetkellä vielä vahvasti kehitysvaiheessa ja täten siitä saatuja tuloksia ei pystytä tarkemmin analysoimaan. Projektin kehitystyö jatkuu tämän insinöörityön dokumentoinnin jälkeen.

VIITELUETTELO

[1] *Envault™ Removable Media Protection* [Verkkodokumentti, viitattu 7.8.09] Saatavissa:http://www.envaultcorp.com/products.php?page=overview.

[2] Wikipedia, *LDAP* [Verkkodokumentti, viitattu 3.8.09] Saatavissa: http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol.

[3] Wikipedia, *Comma Separated Values* [Verkkodokumentti, viitattu 13.8.09] Saatavissa: http://en.wikipedia.og/wiki/.csv.

[4] *Vaadin-sovelluskehys*, dokumentaatio [Verkkodokumentti, viitattu 15.9.09] Saatavissa:http://vaadin.com/download/current/docs/book-of-vaadin.pdf.

[5] *Apache Tomcat 6.0 documentation* [Verkkodokumentti, viitattu 15.9.09] Saatavissa:http://tomcat.apache.org/tomcat-6.0-doc/index.html.

[6] *PotsgreSQL 8.3 Documentation* [Verkkodokumentti, viitattu 15.9.09] Saatavissa:http://www.postgresql.org/docs/8.3/interactive/index.html.

[7] Wikipedia, *Java* [Verkkodokumentti, viitattu 15.9.09] Saatavissa:http://fi.wikipedia.org/wiki/Java.

[8] *Google Web Toolkit Developers's guide* [Verkkodokumentti, viitattu 15.9.09] Saatavissa http://code.google.com/webtoolkit/doc/latest/DevGuide.html.

[9] Wikipedia, *Ajax* [Verkkodokumentti, viitattu 2.10.09] Saatavissa:http://fi.wikipedia.org/wiki/Ajax_(ohjelmointi).

[10] Wikipedia, *Javascript* [Verkkodokumentti, viitattu 3.10.09] Saatavissa:http://fi.wikipedia.org/wiki/JavaScript.

[11] Wikipedia, *Web Application*.[verkkodokumentti, viitattu 3.10.09] Saatavissa:http://en.wikipedia.org/wiki/Web_application.

[12] Wikipedia, *Client-Server* [Verkkodokumentti, viitattu 1.11.09] Saatavissa:http://en.wikipedia.org/wiki/Client-server.

[13] Hämeen-Anttila, Tapio; Ahonen, Tero; Åstrand, Kim, *JavaSerlets Quality*, Docendo, Porvoo 2003.

[14] Java, Abstract Windowing Toolkit [Verkkodokumentti, viitattu 2.11.09] Saatavissa:http://java.sun.com/j2se/1.5.0/docs/guide/awt/.

[15] Open Flash Charts 2 [Verkkodokumentti, viitattu 14.12.09] Saatavissa:http://teethgrinder.co.uk/open-flash-chart-2/.

[16] Wikipedia, *ActionScript* [Verkkodokumentti, viitattu 14.12.09] Saatavissa:http://en.wikipedia.org/wiki/ActionScript.

[17] Wikipedia, *xhtml* [Verkkodokumentti, viitattu 18.12.09] Saatavissa:http://en.wikipedia.org/wiki/Xhtml.

[18] IBM, infosphere data architect [Verkkodokumentti, viitattu 19.12.09] Saatavissa http://www-01.ibm.com/software/data/optim/data-architect/.

[19] Hovi, Ari; Huotari, Jouni; Lähdemäki, Tapio, *Tietokantojen suunnittelu & indeksointi*, Docendo, Porvoo 2005.

[20] Wikipedia, *Agile Software Development* [Verkkodokumentti, viitattu 2.1.10] Saatavissa:http://en.wikipedia.org/wiki/Agile_software_development.

[21] Ubuntu linux [Käyttöjärjestelmä] Saatavissa : http://www.ubuntu.com.

[22] Apache http server [Verkkodokumentti, viitattu: 4.1.10] Saatavissa:http://httpd.apache.org/docs/.

[23] Subversion documentation [Verkkodokumentti, viitattu 4.1.10] Saatavissa:http://subversion.apache.org/docs/.

[24] TortoiseSVN Documentation [Verkkodokumentti, viitattu 3.2.10] Saatavissa:http://tortoisesvn.net/support.

[25] Eclipse IDE [Verkkodokumentti, viitattu 4.2.10] Saatavissa:http://www.eclipse.org/.

[26] Wikipedia, JDBC Driver [Verkkodokumentti, viitattu 4.2.10] Saatavissa:http://en.wikipedia.org/wiki/JDBC_driver.

[27] Wikipedia, WAR (Sun File Format) [Verkkodokumentti, viitattu 4.2.10] Saatavissa:http://en.wikipedia.org/wiki/WAR_(Sun_file_format).

[28] UIDL Documentation [Verkkodokumentti, viitattu 4.2.10] Saatavissa: http://www.uidl.net/doc/.

[29] JSON Documentation [Verkkodokumentti, viitattu 4.2.10] Saatavissa: http://www.json.org/.

[30] ECMA-262 standard, edition 3 [Verkkodokumentti, viitattu 4.2.10] Saatavissa: http://www.mozilla.org/js/language/E262-3.pdf.

[31] Java, Abstract methods and classes[Verkkodokumentti, viitattu 4.2.10] Saatavissa: http://java.sun.com/docs/books/tutorial/java/IandI/abstract.html.

[32] Eclipsen [Verkkodokumentti, viitattu 4.2.2010] Saatavissa: http://wiki.assemblix.net/Eclipse.