VARASTONHALLINTAJÄRJESTELMÄN KEHITTÄMINEN

CASE: Polkupyöräkorjaamo Rikman

Mikko Friman

Opinnäytetyö
Toukokuu 2012

Tietojenkäsittely
Luonnontieteiden ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t) FRIMAN, Mikko	Julkaisun laji Opinnäytetyö	Päivämäärä 07.05.2012
	Sivumäärä 72	Julkaisun kieli Suomi
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)
Työn nimi VARASTONHALLINTAJÄRJESTELMÄN KEHITTÄMINEN CASE: Polkupyöräkorjaamo Rikman		
Koulutusohjelma Tietojenkäsittelyn koulutusohjelma		
Työn ohjaaja(t) IMMONEN, Jarkko		
Toimeksiantaja(t) Polkypyöräkorjaamo Rikman		

Tiivistelmä

Kehitysprojektin tavoitteena oli suunnitella ja toteuttaa Polkupyöräkorjaamo Rikmanille selaimella käytettävä varastonhallintajärjestelmä, jolla tehostettaisiin ja osittain korvattaisiin tähän asti käytettyjä manuaalisia varastonhallintamenetelmiä.

Kehitys toteutettiin perinteisen vesiputousmallin mukaisella vaihejaottelulla, johon haettiin teoriaosuudessa tietoperustaa lähdemateriaaleista. Kehitysprosessissa käytetyt vaiheet olivat määrittely, suunnittelu, toteutus, testaus ja käyttöönotto. Määrittelyvaiheessa selvitettiin toimeksiantajan- sekä ohjelmistotason vaatimukset. Suunnitteluvaiheessa vaatimuksista johdettiin toteutussuunnitelmat kuvaten järjestelmän- ja käyttöympäristön arkkitehtuurit, käytettävät työkalut, tietokantaratkaisu ja tarvittavat käyttöliittymät sekä testaus. Suunnitelmien pohjalta toteutettiin ja testattiin suunniteltu tietokantasovellus. Lopuksi kehitetty varastonhallintajärjestelmä asennettiin toimeksiantajalle.

Kehitysprojektin tuloksena syntyivät toimeksiantajan vaatimukset täyttävä varastonhallintaa tukeva tietokantasovellus sekä kehityksen aikana luotu dokumentaatio, jotka ovat listattuina liitteissä. Dokumentaatio pidettiin melko suppeana projektin vähäisten resurssien vuoksi, mutta kuitenkin lopputuotteen toteutuksen vaatimassa laajuudessa.

Kehitettyyn varastonhallintajärjestelmään jäi myös mahdollisuuksia jatkokehittämiselle. Potentiaalisia kohteita olisivat ainakin varastonhallintaa tukevien laitteiden, kuten viivakoodinlukijan hyödyntäminen sekä tilaustoiminnon integroiminen toimittajien omiin tilausjärjestelmiin.

Avainsanat (asiasanat) varastonhallintajärjestelmä, tietokantasovellus, ohjelmistokehitys	
Muut tiedot Liitteet, 17 sivua.	

Author(s)
FRIMAN, Mikko

Type of publication
Bachelor´s Thesis

Date
07052012

Pages
72

Language
Finnish

Confidential

( ) Until

Permission for web publication
(X)

Title
DEVELOPMENT OF WAREHOUSE MANAGEMENT SYSTEM
CASE: Polkupyöräkorjaamo Rikman

Degree Programme
Business Information Systems

Tutor(s)
IMMONEN, Jarkko

Assigned by
Polkupyöräkorjaamo Rikman

Abstract

The goal of this development project was to design and implement a browser based warehouse management system for Polkupyöräkorjaamo Rikman, which would enhance and partially replace some of the manual warehousing methods used before.

The development was carried out using a traditional waterfall model, which was supported by source material in the theory phase. The phases used in the development process were definition, design, implementation, testing and initialization. The definition phase consisted of the research of the client and software requirements. In the design phase the implementation plans were led from these requirements illustrating the system and environmental architectures, the tools used, the database and the user interfaces required and the testing. The implementation was then carried out based on these designs. Finally, the developed warehouse management system was initialized to the client's environment.

The outcome of this development project was a database application to support warehousing and the documentation created during development, which are listed in the appendixes. The application meets the client's requirements very well. The documentation was created in a somewhat constricted manner because of the tight resources of this project, however, in a proportion needed by the implementation.

The warehouse management system could have potential for further development. For example, the utilization of some assisting equipment, such as a bar code reader and the integration of the ordering feature to suppliers' own systems.

Keywords
warehouse management system, database application, software development

Miscellaneous
Appendixes, 17 pages.

SISÄLTÖ

KUVIOT

TAULUKOT

1 JOHDANTO

Varastonhallinta on yrityksen toiminnan kannalta tärkeä ydinprosessi, joka hyvin toteutettuna on yritykselle myös suuri kilpailuetu. Perinteisesti varastotoimintoja, kuten tuotesaldojen ylläpitoa, varaston tuotteiden sijaintitietoja sekä tuotetilauksia on hallinnoitu manuaalisesti paperimenetelmin. Nykyään, tietotekniikan kehityksen myötä ollaan näiden toimintojen automatisoimiseksi ja tehostamiseksi kehitetty sähköisiä varastonhallintajärjestelmiä. Näillä järjestelmillä pyritään vähentämään manuaalisen työn tarvetta ja keskittämään varastoprosessien hallinta yhdeksi kokonaisuudeksi.

Jyväskyläläisellä Polkupyöräkorjaamo Rikmanilla varastonhallintatoimenpiteet on tähän asti suoritettu pääosin perinteisin keinoin, hyödyntäen ”kynä- ja paperimenetelmin” tehtäviä listoja sekä manuaalista työtä. Esimerkiksi tuotetiedot ja -määrät on tarvittaessa tarkastettu suoraan hyllystä, jolloin niiden loppumista on ollut hankala ennakoida. Yrittäjän kanssa oli puhetta näiden toimintojen tehostamisesta tietotekniikan avulla, jotta esimerkiksi tuotteiden hintatietojen ja varastomäärien tarkastelu helpottuisi. Kattavia kaupallisia varastonhallintajärjestelmiä on markkinoilla useita, joiden hankintaan pienyrittäjällä ei kuitenkaan tässä vaiheessa ollut kiinnostusta. Näin ollen tästä avautui hyvä mahdollisuus opinnäytetyön tekemiseen.

Tässä opinnäytetyössä kehitetään tietokantasovellus kohdeyrityksen varastonhallinnan tueksi. Alusta asti toteutettavan sovelluksen kehitys noudattaa pääasiallisesti perinteisen ohjelmistokehityksen vaiheita, joita ovat määrittely, suunnittelu, toteutus, testaus sekä käyttöönotto. Määrittelyvaiheessa selvitetään toimeksiantajan vaatimukset sekä käyttöympäristön asettamat rajoitteet järjestelmälle. Suunnitteluvaiheessa näiden vaatimusten pohjalta luodaan toteutussuunnitelmat ja valitaan käytettävät työkalut, joiden perusteella toteutusvaiheessa järjestelmä realisoidaan. Tes-

tausvaiheessa etsitään toteutetusta järjestelmästä virheitä, minkä jälkeen käyttöönottovaiheessa järjestelmä tullaan asentamaan toimeksiantajan käyttöympäristöön.

Vaikka nykyisin ohjelmistotuotannossa käytetään paljon ketteriä kehitysmenetelmiä, niin tietokantasovelluksen toteutus perinteisen kehitysprosessin mukaan, käymällä läpi kaikki prosessin vaiheet, antaa tekijälle paljon kokemusta ohjelmistokehityksen vaiheissa käytettävistä tekniikoista ja menetelmistä. Tietokantasovelluksen toteuttamisesta tekijällä on hieman aiempaa kokemusta, joskaan ei aivan vastaavan tyyppisestä järjestelmästä.

2 TUTKIMUSASETELMA

Luvussa kaksi esitellään työn tavoitteet ja rajaukset sekä tutkimusmenetelmät- ja -kysymykset.

2.1 Tavoitteet ja rajaukset

Toimeksiantaja

Toimeksiantajana opinnäytetyössä on Polkupyöräkorjaamo Rikman. Yritys on perustettu vuonna 1993. Jyväskylässä sijaitsevan pienyrityksen toimialaan kuuluvat polkupyörien korjaus, huoltotoimenpiteet sekä varaosamyynti. Toimeksiantaja työllistää yrittäjän lisäksi yhden henkilön.

Tausta ja tavoite

Opinnäytetyön idea sai alkunsa ilmi tulleesta ongelmatilanteesta, jolle lähdettiin hakemaan ratkaisua. Yrityksellä ei ole käytössään tällä hetkellä minkäänlaista varastonhallintaa helpottavaa järjestelmää ja kuten perinteisesti monessa pienyrityksessä, jotkut rutiinitehtävät, kuten varastoinventaariot, suoritetaan kynä ja paperi -periaatteella. Tehokkaaksi ratkaisuksi katsottiin tietotekniikan hyödyntäminen yrityksen varastonhallinnassa.

Opinnäytetyön tavoitteena on tehostaa ja osittain korvata kohdeyrityksen varastonhallintaan liittyviä perinteisiä käytäntöjä. Työn tuloksena saatavan tietokantasovelluksen tarkoituksena on automatisoida varastoon kohdistuvien prosessien käsittelyä ja näin tehostaa varastonhallintaa. Yrityksen tuotesaldot muuttuvat luonnollisesti jokaisen myyntitapahtuman sekä tuoteoston yhteydessä. Niitä on näin ollen vaikeaa ja aikaa vievää päivittää käsin tehtävien inventaarioiden avulla. Tämä on myös huo-

mattavan virhealtista, jolloin tuotteiden loppuminen ja uusien tuotetilausten ennakoiminen hankaloituu.

Täysin uuden varastonhallintajärjestelmän kehittäminen ei ollut aluksi ainoa vaihtoehto. Myös samankaltaisia toimintoja sisältäviä kaupallisia sovelluksia on jo useita markkinoilla, mutta pienyrityksillä on harvoin resursseja suuriin it-investointeihin, ja tässä tapauksessa hankinnan hyödyn ja kulujen suhde saattaisi olla tappiollinen yritykselle. Myös jokunen avoimen lähdekoodin ilmainen sovellus on olemassa. Nämä ovat myös hyvin kattavia paketteja. Useimmiten varastonhallintaan keskittyneet sovellukset ovat tosin suurempien toiminnanohjausjärjestelmien osia. Tässä ongelmaksi nousee tuotteiden tarpeettomien toimintojen määrä kohdeyrityksen kannalta.

Osa avoimen lähdekoodin sovelluksista on GNU GPL-lisenssin alaisia, joka sallii niiden lähdekoodin kopioinnin ja muokkauksen kenelle tahansa. Näin ollen pohdittiin myös mahdollisuutta muokata jo valmista sovellusta yrityksen tarpeisiin sopivaksi. Täysin uuteen järjestelmään päädyttiin kuitenkin, sillä lopputuotteen tuli olla hyvin selkeä, helppokäyttöinen sekä toiminnoiltaan täysin räätälöity kyseiseen tarpeeseen. Myös työn tekijän mielestä tämä oli parempi vaihtoehto ammatillisten taitojen kehityksen näkökulmasta. Tästä johtuen päädyttiin aivan uuden sovelluksen kehittämiseen jo olemassa olevan järjestelmän räätälöinnin sijaan.

Alueen rajaus

Opinnäytetyö rajataan määrittelyn, suunnittelun, sovelluskehityksen sekä tietokannan ja testauksen osalta niin laajaksi, kuin kuvatun tietokantasovelluksen toteutus vähintään pakollisten vaatimusten osalta tarvitsee. Dokumentointi pidetään melko suppeana resurssien vähäisyyden vuoksi, mutta kuitenkin siinä laajuudessa, kuin eri toteutuksen vaiheet pakollisesti vaativat. Opinnäytetyössä ei käsitellä yrityksen liiketoiminnallista puolta, kuin siinä määrin, mitä yrityksen kuvaus vaatii.

2.2 Tutkimusmenetelmät

Tämä opinnäytetyö on tietokantasovelluksen toiminnallinen kehittämisprojekti. Projektissa pyritään tehostamaan kohdeyrityksen varastonhallintaan liittyviä nykyisiä toimintamalleja.

Kohdeyritykselle kehitettävä tietokantasovellus pyritään hahmottamaan keskustelemalla ja kuuntelemalla toimeksiantajaa ja luomalla näin selkeä kuva toteutettavasta työstä ja toimeksiantajan tarpeista. Mahdollisimman hyvä ymmärrys ongelmatilanteesta sekä tuotteelle asetetuista vaatimuksista on tärkeää, jotta lopputuotteesta saadaan toimeksiantajan toiveiden mukainen.

Projekti toteutetaan noudattaen perinteisen ohjelmistokehityksen vaiheita (määrittely, suunnittelu, toteutus, testaus ja käyttöönotto) siinä laajuudessa, kun katsotaan projektille hyödylliseksi. Lopputuotteeseen tehdään projektin aikataulun puitteissa vain toimeksiantajan tarpeet täyttäviä toimintoja.

Määrittelyvaiheessa selvitetään asiakkaan vaatimukset järjestelmälle sekä hahmotellaan ohjelmistotason vaateet. Suunnitteluvaiheessa muunnetaan määritellyt vaatimukset toteutustekniseen muotoon, jonka pohjalta toteutusvaiheessa ohjelmoidaan suunniteltu tietojärjestelmä. Järjestelmän testauksella pyritään löytämään virheitä järjestelmän toteutuksesta, määrittelyistä tai suunnitelmista. Käyttöönottovaiheessa toteutettu järjestelmä siirretään lopulliseen käyttöympäristöönsä. (Haikala & Märijärvi 2006, 38–41.)

2.3 Tutkimuskysymykset

Tässä opinnäytetyössä haetaan vastausta alla esitettyyn pääkysymykseen, johon saadaan vastaus sitä täydentävien alakysymysten tuloksista.

Pääkysymys:

1. Miten kehitetään tietokantasovellus kohdeyrityksen tarpeisiin?

Tähän tutkimuskysymykseen vastataan seuraavilla alakysymyksillä.

Alakysymykset:

1. Millaisia toiminnallisuuksia tarvitaan?

   Tähän kysymykseen vastataan projektin vaatimusmäärittelyssä (luku 4.2) sekä käyttötapauksien avulla (luku 4.3).

2. Millainen tietokannan tulee olla?

   Tähän kysymykseen vastataan tietokannan suunnittelu- (luku 5.4) ja toteutusvaiheissa (luku 6.1).

3. Millaisia käyttöliittymiä tarvitaan?

   Tähän kysymykseen vastataan käyttöliittymän suunnittelu- (luku 5.6) ja toteutusvaiheessa (luku 6.3).

4. Miten testaus suoritetaan?

   Tähän kysymykseen vastataan testauksen suunnittelu- (luku 5.5) ja toteutusvaiheissa (luku 6.4)

3 VARASTONHALLINTAJÄRJESTELMÄN KEHITTÄMINEN

Tässä luvussa määritellään aluksi mikä on varastonhallintajärjestelmä. Tämän lisäksi kuvataan työosuudessa toteutettavan tietokantasovelluksen kehitysvaiheet.

3.1 Varastonhallintajärjestelmät

Varastonhallintajärjestelmä (Warehouse management system - WMS) on tietojärjestelmä, joka toimii apuna yrityksen arjessa tarjoten tarkkaa ja reaaliaikaista tietoa varaston prosesseista. Nykyaikaisiin varastonhallintajärjestelmiin voidaan keskittää kaikki yrityksen varastotoiminnot, jolloin ne tuovat kiistattomia hyötyjä varastonhallintaan perinteisiin paperi-pohjaisiin menetelmiin verrattuna. Tietoa varaston tuotteista voidaan hakea nopeammin, ja materiaalitiedot päivittyvät automaattisesti jokaisen varasto-operaation seurauksena. Varastonhallintajärjestelmät voidaan integroida osaksi yrityksen muita tietojärjestelmiä, jolloin materiaalitietoa voidaan hyödyntää nopeasti koko yrityksen laajuisesti. Nykyaikaiset WMS-järjestelmät voivat hyödyntää myös uusinta teknologiaa materiaalinhallinnassa, kuten radiotaajuutta (RFID)- sekä äänitunnistusta. (Richards 2011, 137–138.)

Richardsin (2011) mukaan WMS-järjestelmän yritykselle tuomia hyötyjä ovat mm:

- Varaston näkyvyys ja jäljitettävyys
- Varaston tarkkuus
- Automaattinen täydennys
- Tarkka raportointi
- Parantunut reagointikyky
- Asiakaspalvelun parantuminen
- Paperityön vähentyminen

(Richards 2011, 138–139).

Nykyisin varastonhallintajärjestelmiä on lukematon määrä, ja ne voivat olla joko kokonaan erillisiä ohjelmistokokonaisuuksia tai suurempien toiminnanohjausjärjestelmien (Enterprise resource planning - ERP) osia. Varastonhallintajärjestelmiä voidaan myös vuokrata järjestelmäpalveluna (SaaS). Nämä ohjelmistot tarjotaan usein internet-selaimella käytettävinä pilvipalveluina (Cloud computing), jossa palveluntarjoaja ylläpitää ja tekee päivityksiä järjestelmään tarpeen mukaan suoraan verkon välityksellä. (Richards 2011, 137, 147–148.)

Kuten edellä mainittiin, on markkinoilla paljon erilaisia varastonhallintajärjestelmiä, mutta yleensä ne tarjoavat ainakin samat perustoiminnot, eli materiaaleihin liittyvät tiedot, sijainnit, tuotemäärät sekä tietoja uusien tilauksien tekoon ja varastointiin. Järjestelmät ovat usein monimutkaisia ja sisältävät paljon toiminnallisuuksia, joten ne vaativat käyttäjiltään huomattavaa asetusten mukauttamista omaan varastointiin sopiviksi sekä jatkuvaa varastotietojen hallintaa. (Piasecki n.d.)

Vaikka tässä opinnäytetyössä kehitettävässä tietokantasovelluksessa on samoja tavoitteita kuin edellä kuvatuissa kaupallisissa WMS-järjestelmissä, tulee se sisältämään projektin aikataulun puitteissa vain toimeksiantajan kannalta tärkeimpiä ominaisuuksia.

3.2 Ohjelmistokehityksen vaiheet

Ohjelmistotuotteen kehitys jaetaan perinteisen mallin mukaan useaan eri loogiseen vaiheeseen, jossa jokainen vaihe toteuttaa oman roolinsa kehitysprosessista. Yksi tällainen vaihejakomalli on paljon käytetty vesiputousmalli, jota tullaan käyttämään myös tässä kehitysprojektissa. Nykyään suosittuja ketteriä kehitysmenetelmiä (agile methods), kuten Scrumia, on harkittu myös käytettäväksi, mutta mm. vesiputousmallin suunnitelmakeskeisyys sopii työn tekijän mielestä paremmin tähän kehitysprojektiin.

Perinteisestä vesiputousmallista on erilaisia variaatioita, joiden toteutus poikkeaa toisistaan. Useimmiten siihen kuitenkin kuuluu vähintään määrittely-, suunnittelu- ja toteutusvaiheet sekä laadunvarmistustoimenpiteitä näiden sisällä. (Sommerville 2007, 66–67, 520.)

Laadunvarmistustoimenpiteillä pyritään tuotetun materiaalin sisältämien virheiden löytämiseen jo mahdollisimman aikaisessa vaiheessa projektia. Konkreettisia toimenpiteita laadun ylläpitämiseen tuotekehityksessä ovat todentaminen ja kelpoistaminen (verification & validation) sekä katselmukset ja ohjelmiston testaaminen, josta lisää myöhemmin. Todentaminen kertoo, vastaako tuote suunnitelmia, kelpoistaminen sen sijaan arvioi tuotetta sen käyttötarkoituksen näkökulmasta. Nämä *tarkastukset*, joissa listataan tarkastelun perusteella löydetyt viat ovat tärkeä osa lopputuotteeseen jäävien virheiden minimoinnissa. Projektin vaiheiden päätteeksi pidetään usein myös vaiheen puitteissa luotujen tuotosten teknisiä *katselmuksia*. Katselmuksissa myös asiakasosapuoli voi olla mukana tarkastelemassa projektin etenemistä. Katselmukset sekä tarkastukset jaksottavat projektin kulkua ja erottelevat projektin eri vaiheet toisistaan loogisiksi osiksi. (Sommerville 2007, 516–523.)

Esitutkimus

Projekti alkaa esitutkimuksella, joka aloittaa useimmiten määrittelyvaiheen. Esitutkimus on kehitettävän tuotteen elinkaaren pohja, jossa selvitetään tilaajan vaatimukset, jotka järjestelmän on toteutettava. Näitä selvitystoimenpiteitä ovat mm. tilauksesta vastaavien haastattelut ja havainnointi loppukäyttäjäympäristössä.
Tilaajan tarve järjestelmälle on aina motiivina asiakaslähtöisissä kehitysprojekteissa. Kootut *asiakasvaatimukset* eivät kuitenkaan suoraan kerro, millainen järjestelmä tulee olemaan, vaan ne selvittävät, mitä hyötyä tilaaja järjestelmällä haluaa saavuttaa. Esitutkimus on tavallaan koko projektin tärkein vaihe, ja vaatimusten perinpohjainen ymmärrys on ensiarvoisen tärkeää lopputuloksen kannalta.
(Sommerville 2007, 144–146.)

Määrittely

Määrittelyvaiheessa tilaajan vaatimukset muutetaan enemmän järjestelmän toteutusta kuvaaviksi. Määrittelyvaihe vastaa projektissa kysymykseen ”Mitä järjestelmä tekee?”. Määrittelyllä selvitetään, onko asiakkaan vaatimukset täyttävän järjestelmän toteuttaminen ylipäänsä järkevää ja kuinka se toteutetaan. Tässä vaiheessa asiakasvaatimukset muunnetaan *ohjelmistovaatimuksiksi, jotka* sisältävät ohjelmiston toimintoihin sekä käyttöympäristöön liittyviä vaatimuksia ja rajoituksia.
(Haikala & Märijärvi 2006, 78–81.)

Asiakasvaatimusten ja ohjelmistovaatimusten yhteyttä kuvataan järjestelmän käyttötapauksilla (use cases), jotka simuloivat järjestelmän käyttöä eri käyttäjäryhmien kannalta, oikeassa, lopullisessa käyttöympäristössä. Käyttötapauksilla kuvataan järjestelmän tärkeimpiä toimintoja arkikielisesti, ilman toteutusteknisiä seikkoja, jotta kaikki projektin osapuolet voivat ymmärtää ne.
(Sommerville 2007, 154–156.)

Ohjelmistovaatimusanalyysin pohjalta luotu dokumentti on nk. *toiminnallinen määrittely*. Toiminnalliseen määrittelyyn sisältyvät järjestelmän toiminnallisten sekä ei-toiminnallisten vaatimusten kuvaukset. Toiminnalliset vaatimukset kuvaavat järjestelmän toimintaa, käyttöliittymiä sekä kommunikointia toisten järjestelmien kanssa. Ei-toiminnalliset vaatimukset sen sijaan ovat useimmiten teknisiä. Ne kuvaavat ehtoja, joiden on täytyttävä toiminnallisia vaatimuksia toteutettaessa. Näitä ovat mm. vasteajat, käytettävyys ja suoritusteho. Lisäksi toiminnalliseen määrittelyyn saattaa kuulua myös projektille asetettuja rajoituksia, joita ovat esimerkiksi tarvittavan laitteiston rajoitukset toteutukselle. (Sommerville 2007, 118–125.)

Vaikka asiakasvaatimukset sekä ohjelmistovaatimukset kartoitetaan ja analysoidaan useimmiten jo projektin esitutkimuksessa tai määrittelyvaiheessa, ne kaipaavat useimmiten päivityksiä myös projektin myöhemmissä vaiheissa. Syitä muutoksiin ovat Haikalan ja Märijärven (2006, 99) mukaan mm. seuraavat:

- Asiakasvaatimuksia ei ymmärretä oikein heti alussa.
- Asiakasvaatimuksia jää usein huomaamatta.
- Tilaajan ohjelmisto-, toimintatapa- ja laitteistomuutokset kesken projektin
- Projektin aikataulupaineet

Suunnittelu

Suunnitteluvaiheessa siirrytään ohjelmiston teknisempään toteutukseen. Suunnitteluvaihe kertoo, *miten* määrittelyvaiheessa kuvattu ohjelmisto toteutetaan. Sen ydin on pyrkimys jakaa määritelty ohjelmistokokonaisuus pienempiin osiin, joiden yhteyttä kuvataan aluksi yleisellä tasolla, ja josta edetään jokaisen erillisen moduulin toiminnan kuvaukseen. Kokonaisuutta pilkotaan niin kauan, että osat ovat tarpeeksi pieniä toteutuksen näkökulmasta. Moduulilla tarkoitetaan tässä yhteydessä yhtä järjestelmän tehtävää varten luotua metodia tai luokkaa. (Vliet 2008, 326–329.)

Suunnittelun ensimmäinen vaihe eli arkkitehtuurisuunnittelu kuvaa järjestelmän moduulirakennetta ja pyrkii näin selkeyttämään ohjelmistokokonaisuutta. Järjestelmään pyritään löytämään yksinkertainen rakenne, jossa moduulit toimivat toisistaan riippumattomasti. Järjestelmän arkkitehtuuri ei ota vielä kantaa moduulien sisäiseen toimintaan, vaan pyrkii selventämään monimutkaista tietojärjestelmän rakennetta kuvaamalla kunkin osasen rajapintoja ja suhdetta toisiinsa. Arkkitehtuurisuunnittelu on projektin onnistumisen kannalta yksi tärkeimmistä vaiheista, ja siinä tehdyt epäloogisuudet tai virheet saattavat nousta kustannuksiltaan myöhemmissä vaiheissa kalliiksi. Monessa projektissa järjestelmän arkkitehtuurin suunnittelu katsotaan kuitenkin toissijaiseksi vaiheeksi suunniteltaessa pelkästään yksittäisiä järjestelmän osia, johtuen mm. aikataulupaineista tai halusta näyttää asiakkaalle ”tuloksia” mahdollisimman nopeasti. (Vliet 2008, 326–327.)

Moduulisuunnittelu keskittyy yksittäisen järjestelmän moduulin sisäisen rakenteen kuvaamiseen. Usein yksi järjestelmän moduuli on yksi tiedosto. Yleensä jokainen suunniteltava tiedosto sisältää loogisesti yhteenkuuluvia asioita, toteuttaen näin

osansa kokonaisuudesta. Jokaisen moduulin sisäinen toiminta pyritään eristämään muusta järjestelmästä mahdollisimman hyvin, jotta niihin tehtävät muutokset eivät vaikuttaisi muuhun järjestelmään. Yksittäisen moduulin suunnittelu kuuluu usein osaksi toteutusvaihetta, jonka ohjelmistosuunnittelija suorittaa itse.
(Haikala & Märijärvi 2006, 308−313,319.)

Onnistuneella modulaarisella eli loogisesti ositetulla suunnitellulla saavutettavia hyötyjä:

- Selkeämpi kuva kokonaisuudesta
- Moduulia voidaan muokata tai se voidaan korvata helposti ilman tarvetta muuttaa montaa järjestelmän osaa.
- Yhden moduulin toteutus ja testaus voidaan suorittaa irrallaan muista.
- Moduuli on uudelleenkäytettävämpi.

(Haikala & Märijärvi 2006, 313.)

Käyttöliittymäsuunnittelu

Järjestelmän käyttöliittymä on sovelluksen käyttäjälle näkyvä osa, jolla käyttäjä kommunikoi sovelluksen kanssa. Käyttöliittymää suunniteltaessa päätetään, millaisia näkymiä ja sisältöä järjestelmä tulee tarjoamaan, ja mitä tapahtuu käyttäjän painaessa kutakin suunniteltua käyttöliittymän painiketta. Käyttäjä voi esimerkiksi muokata tietokannan sisältöä helposti suoraan tietokantasovelluksen käyttöliittymän kautta, ilman tarvetta puuttua suoraan järjestelmän lähdekoodiin tai tietokantaan. Käyttöliittymäsuunnittelulla on suuri vastuu järjestelmän käytöstä saatavasta kokemuksesta. Laiminlyödyllä suunnittelulla käyttöliittymä saattaa estää käyttäjää ymmärtämästä kaikkia toiminnallisuuksia ja olla loppukäyttäjälleen näin ollen liian sekava. Pahimmassa tapauksessa sekava käyttöliittymä saattaa jopa hidastaa tehtävän suorittamista verrattuna entiseen tapaan.
(Sommerville 2007, 363−364.)

Hyvässä käyttöliittymän suunnittelussa tulisi huomioida, että se kehitetään taidoiltaan eritasoisille loppukäyttäjille eikä kehitystiimin jäsenille. Käyttöliittymässä tulisi

huomioida näkymien yhtenäisyys, jotta oppiessaan yhden toiminnon suorittamislogiikan käyttäjä osaisi periaatteessa käyttää jo muitakin toimintoja. Käyttäjän tulisi joutua yllättymään käyttöliittymän kanssa mahdollisimman vähän, sillä käyttäjän oppimisen kannalta on häiritsevää, jos yksi järjestelmän osa tekee samankaltaisesta toimintopainikkeesta jotain aivan toista kuin toinen osa. Järjestelmän tulisi myös huomioida käyttäjän tekemät virheet. Virheiden eliminoinnissa apuna voidaan käyttää esimerkiksi toimintojen vahvistuskysymyksiä tai kumoamis-toimintoja.
(Sommerville 2007, 363–366.)

Tietokantasuunnittelu

Tietokannan suunnittelussa pyritään tilaajaa haastattelemalla selvittämään kaikki oleellinen tieto kohdeyrityksen toiminnasta, jota tietokanta tulee pitämään sisällään, sekä rajoitteet, joita yrityksen liiketoiminta asettaa kyseisille tiedoille. Tietokantaan luodaan näiden pohjalta taulurakenne, jossa jokainen taulu edustaa jotain käsiteltävää tietoaihetta. Kerätyt tiedot muodostavat tietokannan kentät, jotka jaotellaan loogisesti niitä vastaaviin tauluihin. Tauluista valitaan yksi kenttä pääavaimeksi. *Pääavain* yksilöi taulun tietueet eli ilmentymät. Lisäksi kenttiä täytyy määritellä tarkemmin sen mukaan, millaista tietoa ne tulevat sisältämään.
(Hernandez 2003, 81–88.)

Tietojen toiston ja turhan tiedon välttämiseksi tietokannan taulujen välille on luotava loogiset suhteet. Tietokannan hierarkisuutta ja taulujen välisiä suhteita kuvataan eritasoisilla tauluilla, missä alemmat *(child)- taulut* viittaavat ylempiin (*parent*)- tauluihin yhdistävillä kentillä eli *viiteavaimilla*. Nämä *viiteyhteydet* muodostetaan taulut loogisesti yhdistävillä kentillä. Viiteyhteyksiä voi olla erilaisia riippuen siitä, kuinka moneen toisen taulun tietueeseen voidaan viitata. Näitä *kardinalisuuksia* voi olla yhden-suhde-yhteen, yhden-suhde-moneen ja monen-suhde-moneen. Näistä ensimmäinen tyyppi ilmenee, kun taulujen ilmentymät voivat viitata vain yhteen toisen taulun ilmentymään. Yhden-suhde-moneen-tapauksessa yksi ilmentymä voi viitata toisen taulun moneen ilmentymään, ja viimeisessä tyypissä monta tietuetta voi viita-

ta moneen toisen taulun ilmentymään. Tässä tapauksessa viite-eheys voidaan varmistaa välitaululla, joka yhdistää taulut niiden avainkentillä eheäksi kokonaisuudeksi. (Hernandez 2003, 6–7,62–69.)

Edellä kuvatulla relaatiomallisella suunnittelulla on mahdollista tehdä tietokannasta:

- ehyttä tietoa säilyttävä
- tietoa toistamaton
- liiketoiminnan tarpeet ymmärtävä
- itsenäisesti toimiva osa järjestelmää
- helposti ja nopeasti tietoa hakeva

(Hernandez 2003, 17).

Suunnitteluvaiheessa toiminnallinen määrittely muuntuu tekniseksi määrittelyksi, jossa kuvataan kehitettävän ohjelmiston toteutusteknisiä seikkoja. Tekniseen määrittelyyn kuuluvat mm. järjestelmän laitteisto- ja ohjelmistoympäristöjen kuvaukset, tietokanta- ja ohjelmistoarkkitehtuurit sekä moduulikuvaukset. (Haikala & Märijärvi 2006, 83–84.)

Toteutus ja testaus

Toteutusvaiheessa realisoidaan suunnitteluvaiheessa kuvattu järjestelmä määritetyillä välineillä. Totetus on yhdistelmä suunnittelua, ohjelmointia ja testausta sekä todennusta. Lähtökohtana toteutusvaiheessa ovat yksittäiset järjestelmän moduulit, jotka suunnitellaan, kehitetään ja testataan. Moduulit yhdistetään järjestelmäksi suunniteltujen rajapintojen kautta. Paino toteutusvaiheessa on luotettavuudella, toimivuudella ja yksinkertaisuutta korostavalla toteutustavalla. (Vliet 2008, 13.)

Projektin testausta suunniteltaessa lähtökohtana on useasti tuotteesta tai projektista muodostuvat riskitekijät, joita ovat mm.:

- Virheellinen lopputuote
- Huonosti määritetyt asiakasvaatimukset
- Ohjelmistovirheen aiheuttamien vahinkojen mittakaava

- Tilaajan palaute epäonnistuneesta tuotteesta
- Asiakasvaatimusten toteutumattomuus
- Kehitystiimin taidot
- Testauksen kunnioitus kehitystiimissä

(Thompson n.d., 127–129).

Riskit jaetaan prioriteetin mukaisiin ryhmiin, ja testauksen suunnittelussa lähtökohtana ovat osa-alueet järjestelmä- tai organisaatiotasolla, joiden riskit omaavat suurimman toteutumistodennäköisyyden. Riskiarviolla vaikutetaan testauksen aikatauluttamiseen, priorisointiin, tekniikoihin sekä määrään projektin elinkaaren eri vaiheissa. Riskilähtöinen lähestymistapa vaatii jatkuvaa riskien arviointia ja ratkaisemista koko projektin ajan.
(Thompson n.d., 127–129.)

Testaaja voi olla myös projektin ulkopuolinen henkilö. Tällainen ”itsenäinen” testaaja voi olla tehokas apu testausprosessissa. Tälla tavalla ”ulkoistamalla” testausaktiviteettejä, ohjelmakoodista tai dokumentista voidaan löytää paljon enemmän virheitä kuin pelkästään oman tuotoksen katselmoinnilla. Itsenäinen puolueeton testaaja näkee tuotoksen myös realistisemmin kuin tuotoksen suunnittelija/tekijä. Testaajan itsenäisyyteen vaikuttaa etäisyys tuotekehitykseen. Toisaalta, mitä etäisempi eli itsenäisempi testaaja on projektista, sitä negatiivisemmin saattaa tietämättömyys projektin konkretiasta vaikuttaa tuloksiin. (Thompson n.d., 129–131.)

Testauksen suunnittelussa mietitään lähestymisnäkökulmia testauksen toteutukseen. Kaksi eniten käytettyä näkökulmaa testaukseen ovat järjestelmän rakennetta testaava valkolaatikkotestaus (White-Box-testing) sekä mustalaatikkotestaus (Black-Box-testing). Ensin mainitussa strategiassa valitut testitapaukset pyritään tarkastelemaan, kuinka ohjelmisto toimii. Testaus tapahtuu teknisestä näkökulmasta tarkastelemalla ohjelmiston sisäistä toimintaa. Valkolaatikkotestaus vaatii teknistä tietämystä ohjelmiston kehityksestä, joten itsenäinen testaaja ei useinkaan osallistu tähän. Mustalaa-

tikkotestaus taas testaa järjestelmän ominaisuuksia ottamatta kantaa sen sisäiseen toteutukseen. Tällä pyritään löytämään virheitä järjestelmän suunniteltujen ominaisuuksien toiminnasta. Mustalaatikkotestauksen puitteissa voidaan suorittaa toiminnallisia testejä, joilla selvitetään, *mitä* järjestelmän tulisi tehdä, sekä ei-toiminnallisia testejä, jotka tutkivat, *miten* järjestelmä tekee sen, mitä sen pitäisi. Mustalaatikkotestaus ei vaadi niinkään teknistä tietoa kehitettävästä ohjelmistosta, vaan ymmärrystä järjestelmän toimi-alueesta (mm. suorityskyvyn testaaminen) sekä ongelmista, jota ratkaisemaan järjestelmää kehitetään. (Black 2009, 2–3.)

Yleinen testaustapa tietojärjestelmäprojektissa on nk. V-malli. Siinä testaus jaetaan projektin eri vaiheisiin. Yleisesti käytettyjä vaiheita ovat ainakin *yksikkötestaus* (module testing), *integrointitestaus* (integration testing) ja *järjestelmätestaus* (system testing). Testit suunnitellaan niitä vastaavissa suunnittelutasoissa: yksikkötestit moduulisuunnitteluvaiheessa, integrointitestit arkkitehtuurisuunniteluvaiheessa ja järjestelmätestaus määrittelyvaiheessa. (Haikala & Märijärvi 2006, 288–290.)

Yksikkötestauksessa tutkitaan yksittäistä järjestelmän moduulia, mm. luokkia ja aliohjelmia. Tarkoituksena on verrata moduulin todellista toimintaa sen suunniteltuun toimintaan. Yksikkötestauksen lähtökohta on käänteinen verrattuna yksikön eli moduulin toteuttamiseen, sillä pyrkimyksenä ei ole välttämättä moduulin toimiminen oikein vaan virheiden löytäminen sen toiminnasta. Yksikkötestien perusteella debuggaus eli vikojen paikallistaminen koodista on huomattavasti helpompaa testattaessa järjestelmää pienissä osissa, kuin se olisi koko järjestelmän mittakaavassa. (Glenford, Sandler & Badgett 2012, Kappale 5 – Module (Unit) Testing.)

Integrointitestauksen tarkoitus on testata moduulin rajapintojen interaktiota suhteessa toisiin moduuleihin. Testaaminen suoritetaan samaan aikaan järjestelmän moduulien integraation kanssa. Näin järjestelmän kokoaminen osista onnistuu varmemmin, ja mahdolliset järjestelmän toimintaa haittaavat bugit löydetään jo heti järjestelmää rakennettaessa. Integrointitestaus on tärkeä vaihe etenkin suuremmissa kehitysprojekteissa, joissa järjestelmän moduulit käyttävät toistensa palveluita toi-

minnoissaan, joskin se usein jää toissijaiseksi testausvaiheeksi projektin resursseja suunniteltaessa. (Black 2009, 6–7.)

Järjestelmätestauksella testataan perinteisessä vesiputousmallisessa ohjelmistokehityksessä koko kootun järjestelmän toimintaa. Järjestelmätestaus on usein myös tuotteen ”julkaisutestaus” projektin toimeksiantajalle, jossa järjestelmän toimintaa verrataan toiminnalliseen määrittelyyn eli testataan tuotteelle asetettujen vaatimusten toteutuminen. Järjestelmätestaus suoritetaan pääosin käyttäen mustalaatikkotestauksen periaatteita tarkastelematta lähdekoodia. Pääpaino on järjestelmän käyttöliittymän syötteiden ja niistä saatavien seurauksien tarkastelussa. Järjestelmätestauksessa tarkastellaan, toimiiko järjestelmä suunnitellulla tavalla. Tässä vaiheessa voidaan suorittaa esimerkiksi jo aiemmin suunniteltuja käyttötapauksia ja testata näiden onnistuminen. (Sommerville 2007, 540–545.)

4 SOVELLUKSEN MÄÄRITTELY

Tämä luku kuvaa kehitysprojektin ensimmäiset vaiheet alkaen esitutkimuksen alkuselvityksistä ja johtaen tuotteen vaatimusmäärittelyihin ja ohjelmistotason määrittelyihin. Kehityksen eri vaiheissa käytetään apuna teoriaosuudessa esiin nostettuja seikkoja. Projektin vaihejakomalliksi valittiin perinteinen vesiputousmalli, jota räätälöitiin projektin tarpeisiin sopivaksi. Nykyään myös paljon käytettäviä ketteriä kehitysmenetelmiä, kuten Scrum, harkittiin käytettäväksi, mutta tässä projektissa päädyttiin kuitenkin vesiputousmalliin, sen selkeän vaihejaottelun ja suunnitelmakeskeisyyden vuoksi. Lisäksi toimeksiantajalla oli alusta alkaen melko selvät vaatimukset, joista oli hyvä johtaa suunnitelmat.

4.1 Esitutkimus

Projektin alussa haastateltiin toimeksiantajaa ja selvitettiin, mihin tarkoitukseen järjestelmä tullaan tekemään sekä millaista hyötyä siitä pyritään saamaan. Haastatteluista selvisi, että nykyinen toimintamalli varastonhallinnassa on pääosin manuaalinen. Tuoteinventaariot suoritetaan kynä-paperi-menetelmin, ja varastosaldojen tarkkaileminen sekä tuotetietojen etsiminen tapahtuu käytännössä suoraan hyllystä katsomalla.

Tehostamistarpeeksi katsottiin etenkin manuaalisten paperimenetelmien osittainen korvaaminen automatisoimalla varaston sisällön ylläpitoa sekä keskittämällä kaikki tuote- ja hintatiedot yhteen tietojärjestelmään, josta niiden nopea tarkastelu sekä lasku- ja tilauslistojen kokoaminen olisi tehokasta. Tuoteinventaariolistojen teko katsottiin myös osa-alueeksi, jota järjestelmän tulisi tehostaa.

Tutkimuksen aluksi pohdittiin, millainen tietokantajärjestelmä lähtökohdiltaan soveltuisi parhaiten kohdeyrityksen tarpeisiin. Vaihtoehtoina olivat joko valmiin avoimen

lähdekoodin (open source) järjestelmän muokkaaminen, aivan uuden tietojärjestelmän kehittäminen tai kaupallinen valmis ohjelmisto. Kaupallinen ohjelmisto ei tullut kuitenkaan kyseeseen suuren investointitarpeen vuoksi. Avoimen lähdekoodin GPL-(General public license) lisenssin alaisia varastonhallintajärjestelmiä löytyi myös markkinoilta, ja niiden lähdekoodin käyttö, muokkaus ja uudelleenjakelu on useimmiten täysin sallittua ja maksutonta (GPL-lisenssistä voi lukea lisää osoitteesta http://www.opensource.org/licenses/gpl-license.php). Näin ollen tällaisen järjestelmän hyödyntämistä kohdeyrityksen tarpeisiin harkittiin. Esimerkiksi suomalainen *Pupesoft* on PHP:llä ohjelmoitu ja selaimella käytettävä, pääasiassa PK-yrityksille suunnattu toiminnanohjausjärjestelmä, joka sisältää muiden toiminnallisuuksiensa lisäksi myös varastonhallintaominaisuuksia. *MyWMS*- ja *OpenWMS*- järjestelmät ovat taas puhtaasti varastonhallintajärjestelmiä.

Avoimen lähdekoodin sovellusten hyötynä nähtiin monipuoliset toiminnallisuudet sekä laaja kehittäjäjoukko niiden takana, joka varmistaa kehityksen ja päivitysten jatkuvan myös tulevaisuudessa. Myös tietoturvan ja testaamisen kannalta avoimen lähdekoodin sovellukset koettiin turvallisiksi monen eri kehittäjän ollessa niiden toteutuksessa mukana.

Tämän projektin tarpeisiin nähden näiden sovellusten todettiin kuitenkin sisältävän paljon tarpeettomia ominaisuuksia, joiden räätälöintiin arvioitiin kuluvan projektin aikatauluun nähden liian paljon aikaa. Lisäksi jo lähdekoodiin ja sen logiikkaan tutustuminen olisi ollut aikaa vievää. Monet järjestelmät olivat myös kehitetty menetelmin ja työkaluin, joihin työn tekijä ei ollut aikaisemmin paljonkaan perehtynyt, joten lopulta täysin uuden järjestelmän kehittäminen tuntui parhaalta vaihtoehdolta.

4.2 Vaatimusmäärittely

Vaatimusmäärittelyvaiheessa lähtökohtana oli selvittää toimeksiantajan vaatimukset sekä käyttöympäristön asettamat rajoitteet järjestelmälle. Selvitykset tehtiin

haastattelemalla toimeksiantajaa. Vaatimuksia keräämällä pyrittiin vastaamaan kysymykseen, mitä järjestelmä tulee tekemään sekä miten käyttöympäristö tulee rajoittamaan sen toteuttamista. Haastatteluiden pohjalta kerättiin lista alustavista vaateista, joista koottiin myöhemmin alustava kokoelma toiminnallisista- sekä -ei-toiminnallisista vaatimuksista (liite 1). Jokainen vaatimus yksilöitiin ID-tunnuksella, jotta vaatimukset olisivat hyvin erotettavissa toisistaan myöhemmän viittauksen helpottamiseksi. Vaatimuksille asetettiin myös luokka, pakollinen tai valinnainen, riippuen vaatimusten tärkeydestä toimeksiantajalle. Pakolliset vaatimukset olivat priorisoitiltään toimeksiantajalle tärkeimpiä, ja ne toteutettiin ensisijaisesti. Valinnaiset vaatimukset sovittiin toteutettaviksi, jos projektin suhteellisen tiukka aikataulu sen sallisi. Esimerkki toiminnallisista vaatimuksista on kuviossa 1.

ID	Kuvaus	Luokka
FR_1	Käyttäjä voi lisätä tuotteen järjestelmään	Pakollinen
FR_2	Käyttäjä voi poistaa tuotteen järjestelmästä	Pakollinen
FR_3	Käyttäjä voi muokata järjestelmässä olevan tuotteen tietoja	Pakollinen

KUVIO 1. Esimerkki toiminnallisista vaatimuksista

Tärkeimmät järjestelmän toiminnalle asetetut pakolliset vaatimukset ovat tietokannan hallitsemiseen liittyvät toiminnot tuotteiden, toimittajien sekä palveluiden osalta ja laskun luonti- toiminto sekä tuoteinventaariolistojen tulostus ja tuotteiden saldohälytykset. Lopputuotteen sovittiin olevan valmis vähintään pakollisten vaatimusten täytyttyä ja toimeksiantajan tällöin hyväksyttyä tuotteen.

Vaatimusmäärittelyssä pyrittiin saamaan mahdollisimman tarkka yhteisymmärrys toimeksiantajan kanssa, jotta järjestelmän kehitys saataisiin lähtemään käyntiin heti alusta alkaen oikeiden tietojen pohjalta. Oletuksena oli, että varsinkin vesiputousmallin mukaisessa vaihejaottelussa vaatimusten muutokset kesken myöhempien projektin vaiheiden saattavat aiheuttaa paljolti lisätyötä jouduttaessa palaamaan aiempiin vaiheisiin täydentämään suunnitelmia, dokumentaatiota ja toteutusta.

Projektin toimeksiantajalla oli alusta alkaen selkeä kuva ongelmatilanteesta sekä lopputuotteessa tarvittavista toiminnoista, joten vaatimusmäärittely eteni varsin jouhevasti, joskin muutamaan otteeseen päädyttiin päivittämään projektin vaatimuksia.

4.3 Käyttötapaukset

Toiminnallisten vaatimusten pohjalta luotiin järjestelmän tärkeimpiä toimintoja kuvaavat käyttötapaukset (liite 2), joilla pyrittiin konkretisoimaan järjestelmän käyttöä ja esittelemään sitä toimeksiantajalle. Käyttötapauksissa kuvattiin toiminnon suorittajat, esiehdot, toiminnon kuvaus, poikkeukset sekä toiminnon onnistuneen suorituksen lopputulos. Suorittaja tämän järjestelmän kohdalla on aina käyttäjä, sillä useampia eritasoisia käyttäjätilejä, kuten ylläpitäjä tai peruskäyttäjä, ei katsottu tässä järjestelmässä tarpeellisiksi. Esiehdot kertovat, mitä toimintoja täytyy olla voimassa, jotta käyttötapauksen mukaisen toiminnon suorittaminen onnistuu. Kuvaus-kohta kertoo yleiskielisesti toiminnon suorittamisen vaiheet sekä kaikki mahdolliset poikkeavat tilanteet, joihin käyttäjä voi törmätä virheellisen käytön seurauksena. Poikkeukset-kohta selittää edellä mainitut poikkeavat tilanteet tarkemmin ja kuvaa, kuinka järjestelmä vastaa käyttäjälle näissä tilanteissa. Jälkiehdot kuvaavat lopputilanteen onnistuneen toiminnon suorittamisen jälkeen sekä järjestelmän palautteen käyttäjälle onnistuneesta suorituksesta. Taulukossa 1 on esimerkki järjestelmän käyttötapauksesta.

TAULUKKO 1. Käyttötapausesimerkki tuotteen lisäyksestä järjestelmään

Suorittajat:	Käyttäjä
Esiehdot:	Järjestelmä on käynnissä, tietokanta toiminnassa, käyttäjä on painanut "lisää tuote"-painiketta.
Kuvaus:	Käyttäjä syöttää tiedot kenttiin [Poikkeus 1: kentässä kiellettyjä merkkejä] ja painaa ”tallenna”-painiketta.
Poikkeukset:	1. Kentässä kiellettyjä merkkejä: Käyttäjä on syöttänyt johonkin

	kenttään kiellettyjä merkkejä. Käyttäjää pyydetään korjaamaan virheellisen kentän syöte ja yrittämään uudelleen.
Jälkiehdot:	Käyttäjä on lisännyt tuotteen järjestelmään, josta järjestelmä informoi käyttäjää.

Käyttötapausten pohjalta toimeksiantaja kertoi omia näkemyksiään, jotka täsmentivät järjestelmän määrittelyitä ja josta oli paljon hyötyä myös myöhemmissä vaiheissa, mm. käyttöliittymän suunnittelussa sekä järjestelmätestauksessa.

5 SOVELLUKSEN SUUNNITTELU

Tässä luvussa kuvataan, miten määrittelyjen pohjalta suunniteltiin ja kuvattiin toteutettavan sovelluksen, käyttöympäristön ja tietokannan arkkitehtuurit, käyttöliittymät, käytettävät työkalut sekä testaus.

5.1 Arkkitehtuurisuunnittelu

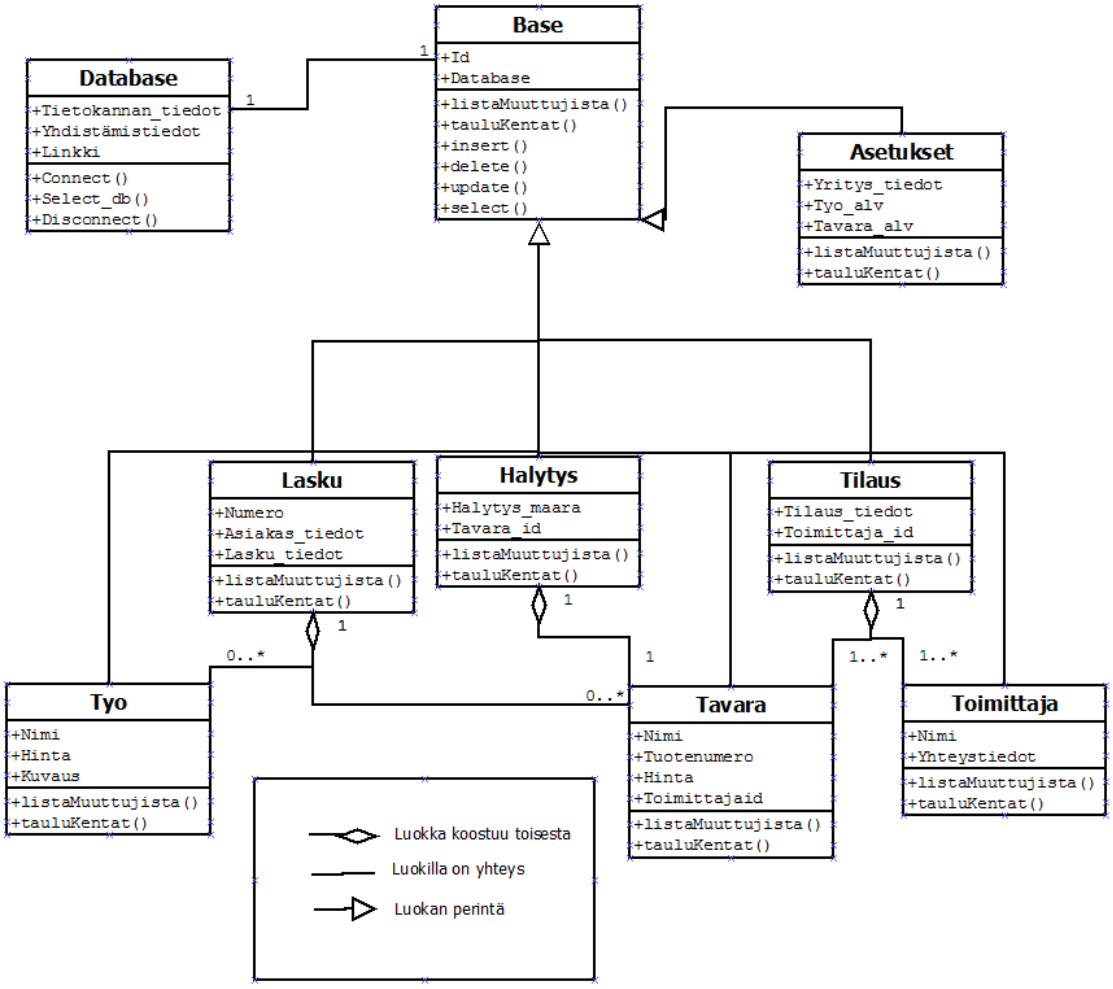
Järjestelmää alettiin suunnitella määriteltyjen tietojen pohjalta hahmottelemalla, millaisia tietoja kokonaisrakenne tulee käsittelemään, ja missä suhteissa nämä tiedot tulevat toimimaan toisiinsa nähden. Koska käytettävän PHP-ohjelmointikielen versio tukee olio-ohjelmoinnin periaatteita, päädyttiin järjestelmässä hyödyntämään luokkarakennetta. Järjestelmän arkkitehtuuria pyrittiin hahmottamaan UML (Unified Modeling Language) -mallinnuskielellä luomalla arkkitehtuurista luokkakaavio (kuvio 2), jossa kuvataan luokat (ts. moduulit) ja niiden väliset hierarkiat sekä osallistuvuusrajoitteet. Tässä vaiheessa ei mietitty luokkien sisältöä sen kummemmin, vaan pyrittiin selvittelemään, millaisia luokkia tarvitaan ja miten ne sijoittuvat keskenään järjestelmään. Luokkakuvaukseen sijoitettiin vain joitakin ideatasolla olevia muuttujia ja metodeja. Luokat johdettiin toiminnallisten vaatimusten määrittelyjen pohjalta.

Hierarkisuus luokkien välillä kuvattiin eritasoisilla luokilla. Järjestelmän arkkitehtuurin alemman tason luokat:

- Työ
- Tavara
- Hälytys
- Toimittaja
- Tilaus
- Lasku

- Asetukset

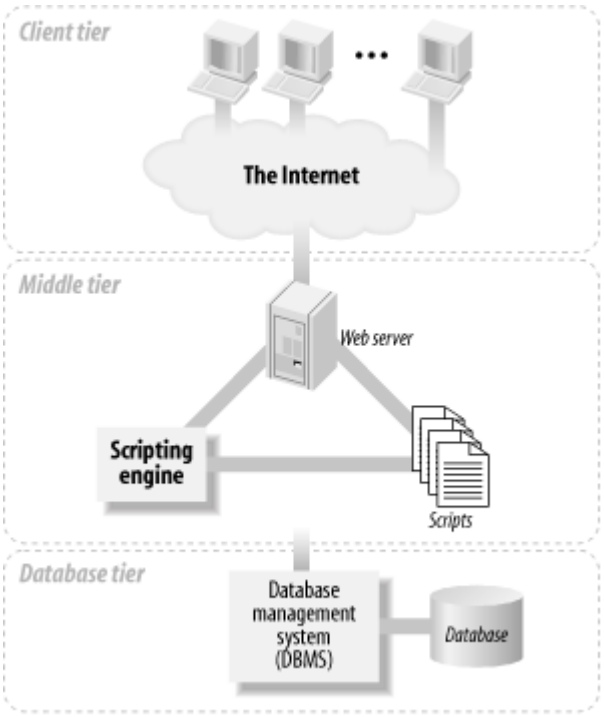
”Base”-luokka suunniteltiin toteuttavaksi luokaksi, joka periytettiin alemman tason luokille, jotka tulevat käyttämään sen palveluita. Tietokannan ja sovelluksen välille suunniteltiin tietokanta-toimintoja sisältävä luokka - Database, joka tarjoaa muille luokille yhteyden tietokantaan. Asetukset-luokka on muista erillinen luokka, josta johdetaan muihin luokkiin sen sisältämiä attribuutteja eli ominaisuuksia. Näitä ovat esimerkiksi arvonlisäverot, jotka vaikuttavat töiden ja tavaroiden hintoihin sekä laskulle koostettavat yritystiedot. Luokkien väliset rajoitteet kuvattiin lukumäärittein. Esimerkiksi yksi Tilaus voi koostua yhdestä tai useammasta Tavarasta ja Toimittajasta.



KUVIO 2. Luokkakaavio

5.2 Toimintaympäristö

Toimintaympäristön arkkitehtuuri noudattaa tässä projektissa kolmikerros-mallia (three-tier-model) (kuvio 3), jossa käyttäjän internet-selain toimii asiakaskerroksena, jonka kautta sovellusta käytetään. Asiakaskerros keskustelee http-protokollan välityksellä välikerroksen eli web-palvelimen kanssa. Palvelimella toimii ilmainen Apache-palvelinohjelmisto, jossa on PHP-laajennus. Web-palvelimella sijaitsee myös sovelluksen ohjelmakoodi. Välikerros toimii palvelimena asiakaskerrokselle ja asiakkaana pohjakerrokselle eli tietokantapalvelimelle, jolla tässä tapauksessa on asennettuna MySQL-tietokantaohjelmisto. Kaikki kerrokset sijoitettiin tässä projektissa yhdelle paikalliselle Windows-pohjaiselle tietokoneelle.



KUVIO 3. Esimerkki kolmikerros-mallista (Web database applications with php & mysql, 2003.)

Web-palvelimesta ainoa käyttäjälle näkyvä osa on järjestelmän käyttöliittymä. Käyttöliittymän kautta järjestelmän käyttäjä suorittaa toimintoja, eli useimmiten pyytää

palvelimelta jotain uutta näkymää tai suorittaa kyselyjä tietokantaan. Palvelimella sijaitseva PHP-tulkki kääntää php-kielisen ohjelmakoodin selaimen ymmärtämälle kielelle ja palauttaa käyttöliittymään html (hypertext markup language) -sivun. Tietokantaan kohdistuvissa kyselyissä palvelimen tietokantaohjelmisto suorittaa sql (structured query language) -kielisen haun tietokannasta, ja palvelin palauttaa html-sivun uusilla tiedoilla järjestelmän käyttöliittymään.

5.3 Työkalujen valinta

Suunnittelussa ja kehitystyössä käytettävien työkalujen valintaan vaikuttivat niiden saatavuus, kehitettävän järjestelmän sekä loppukäyttäjäympäristön asettamat vaatimukset sekä työn tekijän aikaisempi käyttökokemus. Kaikki käytetyt työkalut olivat ilmaisia sekä avoimen lähdekoodin lisenssien alaisia. Projekti toteutettiin pääasiassa työn tekijän kahdella tietokoneella, joista toinen on kannettava ja toinen pöytäkone. Molemmissa koneissa on Windows 7 -käyttöjärjestelmät.

Suunnittelun apuna käytettiin Dia-kaaviopiirrosohjelmaa (versio 0.97.1), joka on hyvin helppokäyttöinen ja sisältää tarvittavat ominaisuudet sekä kaaviotyypit tämän projektin tarpeisiin. Pääasiallisena kehitystyökaluna projektissa käytettiin Netbeans -ohjelmointiympäristön versiota 7.1, joka oli projektin alkaessa uusin versio. Netbeans valittiin kehitystyökaluksi, koska se on hyvin tuettu PHP-ohjelmointikielen kanssa. Lisäksi Netbeans oli työn tekijälle jo entuudestaan tuttu mm. JAMK:n ohjelmointikursseilta. Netbeansissa on paljon ohjelmointia tehostavia ominaisuuksia, kuten ohjelmakoodin värillinen korostus, automaattinen koodin muotoilu sekä koodilohkojen tagien automaattinen täydennys. Myös projektin kooditiedostojen hallinta ja navigointi niiden välillä on kätevää Netbeansin kautta. (Netbeansin PHP-kehitystä tukevista ominaisuuksista voi lukea lisää osoitteesta http://netbeans.org/features/php/). Tämän lisäksi kehityksessä käytettiin satunnaisesti Notepad++ -tekstieditoria (versio 5.6.8) sen helppokäyttöisyyden sekä keveyden vuoksi.

Kehitysympäristössä oli asennettuna uusin WampServer-palvelinohjelmisto (versio 2.2), jossa on asennettuna valmiiksi PHP (versio 5.3.10), MySQL-tietokantaohjelmisto (versio 5.5.20) sekä Apache-palvelinohjelma (versio 2.2.21). WAMP-ohjelmistoa hyödynnettiin kehitystyössä sekä testauksessa. MySQL-tietokantaohjelmisto valittiin, koska tiedettiin jo aiemmasta käyttökokemuksesta sen hyvästä yhteensopivuudesta PHP:n kanssa. Tietokantasuunnittelussa käytettiin WAMP-paketin mukana tullutta PHPMyAdmin-ohjelmaa (versio 3.2.0.1), joka tarjoaa graafisen käyttöliittymän tietokannan hallintaan. Tämä poisti tarpeen käyttää MySQL:n omaa konsolia tietokannan muutoksiin sekä testaamiseen.

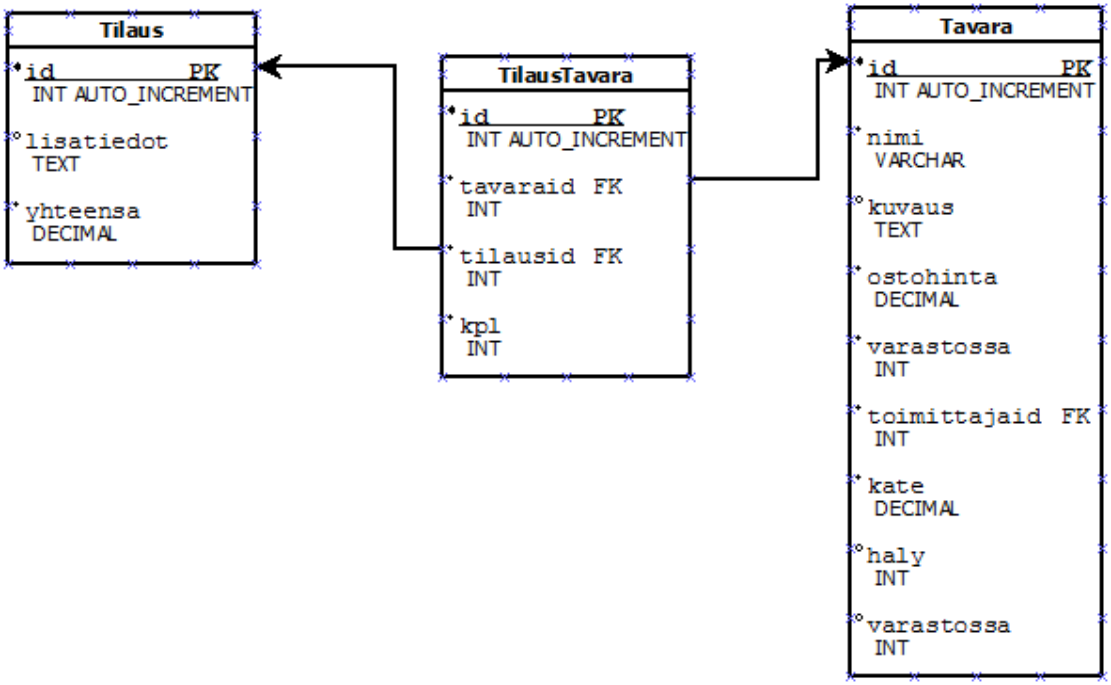
5.4 Tietokannan suunnittelu

Tässä projektissa tietokanta suunniteltiin kolmessa osassa. Aluksi vaatimusmäärittelyjen, käyttötapausmallien sekä täydentävien toimeksiantajan haastatteluiden pohjalta kerättiin lista tiedoista, joita tietokanta tulisi sisältämään. Näistä tiedoista koottiin alustava taulurakenne sekä vaadittavat kentät tietokannalle. Heti alusta asti pyrkimyksenä oli suunnitella tietokantaan mahdollisimman muuttumaton taulurakenne, jotta suurempia muutoksia tietokantaan ei jouduttaisi tekemään enää myöhemmissä vaiheissa. Oletuksena oli, että tietokannan kenttiin kohdistuvien muutosten teko ei tulisi aiheuttamaan kovinkaan suuria toimenpiteitä.

Toisessa vaiheessa luotiin kerättyjen tietojen pohjalta ER (Entity-Relationship) -käsitekaavio (liite 3), jolla saatiin hahmoteltua tarkemmin tietokannan rakennetta ja taulujen suhteita. Jokaiselle taululle lisättiin yksilöivä pääavain (primary key – PK) -kenttä, jotta taulun muodostamat tietueet erotettaisiin toisistaan. Taulujen väliset kardinalisuudet kuvattiin ER-kaavioon määrittämällä sopivat suhdetyypit kunkin toiseensa liittyvän taulun välillä. Esimerkiksi tilaus *koostuu* tavaroista ja toimittajista. Kardinalisuudet kuvaavat, kuinka moneen kohteeseen kukin taulun tietue voi olla suhteessa. Esimerkiksi Lasku- ja Tavara-taulujen välinen yhteys on yhden-suhde-moneen, jota kuvataan ER-kaaviossa 1 – M-merkinnällä. Toisin sanoen lasku koostuu

yhdestä tai monesta tavarasta. Tietokantaan suunniteltiin myös Asetukset-taulu, jonka kenttien arvot vaikuttavat muiden taulujen kenttien arvoihin.

Lopuksi ER-kaavio transformoitiin eli muunnettiin relaatiotietokannaksi ja suunniteltiin taulujen kentille tietotyyppimääritykset niiden sisältämän tiedon mukaan (liite 4). Tässä vaiheessa huomioitiin erityisesti taulujen väliset viiteyhteydet, jotta tietokantaan olisi helppoa ja tehokasta suorittaa hakuja järjestelmästä. Tauluihin, jotka viittaisivat muihin tauluihin, lisättiin viiteavain (Foreign Key - FK) -kenttä. Viiteavaimilla viitataan taulujen pääavaimiin. Esimerkiksi tavaralle tulee voida määrittää toimittaja, joten Tavara-tauluun lisättiin viiteavain-kentäksi toimittajaid eli Toimittaja-taulun tietueet yksilöivä kenttä. Monisuhteisten taulujen väliin luotiin välitaulut, jotka yhdistävät kahta päätaulua. Esimerkiksi yksi id-kentällä yksilöity tilaus voi sisältää monta eri tavaraa, joten ongelma ratkaistiin luomalla taulujen väliin TilausTavara-taulu, jossa yhdistettiin viiteavaimilla kukin tilaus ja siihen liittyvät tavarat. Kuviossa 4 on esimerkki monisuhteisesta relaatiosta.



KUVIO 4. Esimerkki tietokannan relaatioista

Jokainen tietokannan kenttä tuli määritellä myös tarkemmin sen sisältämän tietotyypin mukaan. Pääavaimet sekä muut samanlaista numeerista tietoa sisältävät kentät

määritettiin kokonaisluku-kentiksi (Integer - INT). Taulujen pääavaimet saivat myös auto_increment-määrityksen, jonka mukaan jokaiselle uudelle tietueelle lisätään automaattisesti yksilöllinen kokonaisluku-tunniste. Tekstitietoa sisältäville kentille määritettiin tietotyyppi sen mukaan, kuinka paljon ne tulisivat sisältämään tekstiä. Tietokannan tekstikentät ovat joko VARCHAR- tai TEXT-tyyppisiä kenttiä. TEXT-tyyppiä käytettiin pääasiassa kentille, jotka sisältäisivät paljon tekstiä, ja VARCHAR-tyyppiä lyhemmille tekstikentille. VARCHAR-kenttien merkkimäärät määritettiin tarpeellisen kokoisiksi, tarkoituksena vähentää ylisuuria merkkijonoja. Hintoja tai muita desimaalilukuja tarvitsevat kentät määritettiin DECIMAL-tyypillä, johon voitiin määrittää vielä tarvittava desimaalilukujen määrä. Lasku-taulun päivämäärä- ja eräpäivä-kentät saivat DATE-tyypin, joka tallentaa merkkijonon tietokantaan päivämäärä-muodossa. Kenttien sisältöä tarkennettiin myös määrittämällä taulujen pakolliset kentät, joihin ei voida syöttää tyhjää arvoa tietuetta lisättäessä. Esimerkiksi Lasku-taulun asiakas-tiedot voivat olla tyhjinä käyttäjän näin halutessa, mutta laskun id-kenttä ei voi.

5.5 Testauksen suunnittelu

Järjestelmän testausta suunniteltaessa pohdittiin aluksi projektin sekä kehitettävän tuotteen riskitekijöitä. Suurimpina riskeinä nähtiin aikataulun paikkansapitävyys, pakollisten vaatimusten toteutuminen lopputuotteessa sekä tärkeimpiin toiminnallisuuksiin jäävien virheiden aiheuttamat ongelmat. Riskeistä pääteltiin, että testausstrategioina tulisi käyttää hieman järjestelmän lähdekoodia hyödyntävää lasilaatikkotestausta sekä järjestelmän käyttäytymistä tarkkailevaa mustalaatikkotestausta ja näiden yhdistelmiä. Lisäksi jo projektin alussa päätettiin, että projektissa tuotetulle dokumentaatiolle sekä suunnitelmille olisi järkevää suorittaa staattista testausta eli katselmuksia ja tarkastuksia, joilla pyritään analysoiden sekä tarkastellen minimoimaan virheiden määrä mahdollisimman aikaisessa vaiheessa.

Testauksesta tässä projektissa vastasi pääosin työn tekijä itse, mutta järjestelmän käyttäytymisen testauksessa käytettiin apuna myös muutamaa ulkopuolista testaajaa. Tarkastuksissa sekä katselmoinneissa pyrittiin hyödyntämään mahdollisimman paljon projektin ulkopuolista näkökulmaa. Projektissa katsottiin tärkeäksi testata sovellusta sekä katselmoida dokumentaatiota mahdollisimman paljon toimeksiantajan kanssa. Näin haluttuihin muutoksiin voitaisiin reagoida heti eikä vasta järjestelmän valmistumisen jälkeen.

Järjestelmän lähdekoodia ja sovelluksen toimintalogiikkaa hyödyntävää lasilaatikkotestausta suunniteltiin käytettäväksi pääasiallisesti sovelluksen toteutuksen yhteydessä testaamalla toteutettujen elementtien toimintaa itsenäisesti ja yhdessä. Mustalaatikkotestaustapaa tulisi käyttää myös koko toteutusvaiheen ajan järjestelmän oikean toiminnan vertaamisessa sen toiminnallisiin vaatimuksiin. Erilaista syöteaineistoa tulisi testata eri toiminnallisuuksien tekstikenttiin ja analysoida niiden pohjalta järjestelmän osien toimintaa. On käytännössä mahdotonta testata jokaista syötettä tai syötekombinaatiota, joten testauksessa suunniteltiin käytettävän kentän syötteiden raja-arvoja. Raja-arvoilla tarkoitetaan tässä sallittujen syötteiden muutamaa pienintä ja suurinta arvoa. Oletuksena oli, että jos toiminto antaa odotetun laisen vastauksen syötteen raja-arvoilla, niin se tekee sen myös muilla näiden välisillä arvoilla. Järjestelmän toimintaa tuli testata myös oletetusti ongelmia aiheuttavilla syötteillä, kuten 0 tai kielletyt merkit.

Projektin lopussa, ennen tuotteen lopullista käyttöönottoa, suoritettaisiin vielä järjestelmätestaus, jossa toimeksiantaja testaisi koko sovelluksen toimintaa käyttäen järjestelmän omia käyttöliittymiä. Tämä suoritettaisiin mm. kokeilemalla kuvattujen järjestelmän käyttötapauksien suorittamista (liite 2) sekä todentamalla asiakkaan vaatimusten toteutuminen järjestelmässä.

5.6 Käyttöliittymäsuunnittelu

Käyttöliittymien suunnittelussa pyrittiin ottamaan huomioon visuaaliset seikat, navigointi järjestelmässä sekä toimintojen käytettävyys. Järjestelmän ulkonäön suunnittelussa työn tekijällä oli melko vapaat kädet, ja toimeksiantajan ainoat vaatimukset käyttöliittymälle olivat yksinkertaisuus sekä helppo opittavuus. Näkymien tuli myös olla yhdenmukaisia. Järjestelmän käyttöliittymät suunniteltiin pääosin toiminnallisten vaatimusten sekä käyttötapausten pohjalta. Näihin nojaten piirrettiin alustavia kuvia eri näkymistä, joita esiteltiin toimeksiantajalle. Toimeksiantajan hyväksyttyä alustavat mallit kuvattiin käyttöliittymän näkymiä UML-kielisesti käyttöliittymädiagrammilla, jossa on esitelty kaikki eri näkymät ja liikkuminen näiden välillä (liite 5).

Näkymäkuvista edettiin ensimmäisen prototyypin toteuttamiseen, jossa työn tekijä ohjelmoi osan järjestelmän käyttöliittymistä html-merkkauskielellä hyödyntäen myös css (cascading style sheets)-tyylikieltä. Tässä vaiheessa ei toteutettu vielä varsinaisia toiminnallisuuksia järjestelmään vaan esiteltiin ainoastaan eri näkymien piirteitä, kuten yleistä visuaalista ilmettä, toimintapainikkeiden sijainteja ja tekstien sijoittumista.

Käyttöliittymän yläosa suunniteltiin kiinteäksi elementiksi, joka säilyisi kaikissa näkymissä muuttumattomana. Tällä pyrittiin helpottamaan navigointia sekä edistämään järjestelmän yhtenäisyyttä. Yläosaan sijoitettiin joitain tärkeäksi katsottuja painikkeita, kuten hakukenttä ja monivalintapainikkeet sekä asetukset, ostoskori ja inventaariolistan teko. Käyttöliittymän keskielementti suunniteltiin esittämään uusia näkymiä käyttäjän liikkuessa järjestelmässä ja alaosa taas tarjoamaan jokaisessa näkymässä mahdollisuuden palata takaisin edelliselle sivulle.

Ensimmäinen näkymä, etusivu, näyttää listauksen varaston tavaroista ja niiden tärkeimmistä tiedoista. Jokaisen kohteen kohdalle sijoitettiin valintapainikkeet, josta sitä voidaan tarkastella lähemmin, muokata, poistaa tai lisätä ”ostoskoriin” lasku- tai

tilauslistaukseen. Myös uuden tavaran lisääminen onnistuu tästä valikosta. Tiedot esitetään taulukkomuotoisena, jossa jokainen rivi vastaa yhtä kohdetta.

Hakupainiketta painamalla avautuu uusi haku-näkymä, joka esittää haetut tiedot taulukoina listaten tulokset hakukriteerien mukaisiin kategorioihin, muuttaen ainoastaan kohteen valintapainikkeet sen tarvitsemien toimintojen mukaan. Kohteen tiedoista valittiin tähän näkymään näytettäviksi vain tärkeimmäksi katsotut, kuten nimet, hinnat ja tavaroiden varastosaldot.

Kohteiden muokkaus-, lisäys- sekä asetukset-näkymä ovat käytännössä samanlaisia. Ne sisältävät tekstikenttiä, pudotusvalikoita ja toimintopainikkeita tietojen tallennukseen sekä joihinkin toiminnallisuuksiin, kuten tavaran myyntihinnan laskemiseen. Kohteen tarkastelunäkymä esittää kaikki tiedot kohteesta taulukkomuotoisena. Tästä näkymästä on myös mahdollista edetä suoraan kohteen muokkausnäkymään.

Ostoskorinäkymässä käyttäjä voi tarkastella ostoskoriin lisättyjä tavaroita ja palveluita sekä muokata kohteiden määriä tai poistaa niitä listasta. Taulukkomuotoinen lista kertoo myös kohteiden yksikköhinnat sekä koko listauksen yhteishinnan. Tavara- ja palvelulistat erotettiin toisistaan selkeällä rajauksella, jolla pyrittiin selkeyttämään näkymää mahdollisesti suurienkin listojen erotettavuuden vuoksi. Ostoskorinäkymään lisättiin painikkeet laskun sekä uuden tilauksen luomiselle.

Laskunäkymässä yhdistettiin ostoskorinäkymän tavara- ja palvelulistat sekä laskun muiden tietojen täydennys. Muille tiedoille lisättiin näkymään tekstikentät asiakkaan kaikille tiedoille sekä laskun tarpeellisille yksityiskohdille, kuten eräpäivälle ja viitenumerolle. Tilausnäkymä suunniteltiin muuten samanlaiseksi laskunäkymän kanssa, mutta siinä tilausta varten listatut tuotteet ryhmiteltiin tavaran toimittajan mukaan, ja käyttäjä voi syöttää tilaukselle vain lisätietoja.

Järjestelmän käyttöliittymään varattiin tila ”palaute”-kohdalle jokaisen näkymän yläosaan, johon kaiken järjestelmän käyttäjälle antaman palautteen ja ohjeistuksien

tulisi tulostua. Käyttäjää pyritään informoimaan jokaisen toiminnon suorittamisen jälkeen joko onnistuneesta suorituksesta, kuten tietojen onnistuneesta tallennuksesta, tai vaikkapa jonkin pakollisen kentän tyhjäksi jättämisestä. Käyttäjän ohjeistaminen ja palautteen antaminen jokaisen toiminnon suorittamisen jälkeen katsottiin tärkeäksi järjestelmän opittavuuden ja helppokäyttöisyyden vuoksi. Lisäksi joitain pysyviä muutoksia koskevia toimintapainikkeita, kuten tietoja poistettaessa, järjestelmän tulisi tulostaa varmistuskehote, jolla voitaisiin peruuttaa toiminto. Tällä pyrittiin estämään tilanteet, jossa käyttäjä sattuisi poistamaan virhepainalluksen seurauksena tietoja. Käyttäjän tuli myös saada jonkinlainen ilmoitus varastosaldojen tilasta, joten käyttöliittymään suunniteltiin ilmoitusikkuna, joka tulostuisi tuotesaldojen vähetessä tietyn rajan alle. Ilmoitus oli loogista sijoittaa uuden laskun tekemisen yhteyteen.

Järjestelmästä tuli myös voida suorittaa monivalintahakuja 7:llä eri kriteerillä. Näitä olivat tuotenumero, tavaran ja toimittajan nimi, työn otsikko, laskun- ja tilauksen päivämäärä sekä varastosaldohälytykset tavaran nimen perusteella. Käyttöliittymään katsottiin hyödylliseksi myös mahdollistaa kaikkien kategorian mukaisten kohteiden listaaminen.

Toimeksiantajalla oli myös vaatimuksena mahdollisuus tulostaa tuoteinventaariolistoja, laskuja sekä tilauksia. Näihin kaikkiin tulisi tehdä omat näkymät, jotka tulisi optimoida tulostuksen asettamat vaatimukset silmällä pitäen. Kaikki tarpeeton tieto tuli karsia pois, kuten käyttöliittymän ylä- ja alaosa sekä toimintopainikkeet. Tulosteista tehtäisiin myös mustavalkoisia, millä voitaisiin nopeuttaa tulostusta ja säästää tulostimen värejä. Laskuille tehtäisiin erikseen normaalin laskupohjan mukainen näkymä, jonka käyttäjä voisi tulostaa asiakkaalle.

6 SOVELLUKSEN TOTEUTUS

Tämä luku kuvaa kehitettävän varastonhallintasovelluksen toteutusvaiheet tietokannan, sovelluksen ja testauksen osalta.

6.1 Tietokannan toteutus

Tietokanta toteutettiin suunnitteluvaiheessa kuvatun relaatiokaavion pohjalta (liite 4) rakentamalla sql-kielinen luontilausetiedosto (liite 6), josta tietokanta voitaisiin asentaa helposti. Aluksi tehtiin luontilauseet tietokannalle sekä sen käyttäjälle. Tietokannan luomisen yhteydessä määritettiin tietokantapuolen merkistökoodaukseksi utf-8, jotta tietokanta tunnistaisi myös selaimelta saamansa skandinaaviset merkit. Jotta tietokantaa voitaisiin käsitellä, luotiin siihen käyttäjä. Käyttäjälle määritettiin tunnukset, joilla tietokantaan päästäisiin käsiksi. Tunnuksille määritettiin lisäksi oikeudet lisätä tai poistaa käyttäjiä, jos tarve vaatisi tulevaisuudessa. Tämän jälkeen kirjoitettiin kaikille suunnitelluille tauluille ja niiden kentille luontilauseet, joissa ovat myös sql-kielisesti ilmaistuna kenttien spesifikaatiot. Viite-eheyden säilyttämiseksi taulujen välille määritettiin ON UPDATE CASCADE- sekä ON DELETE CASCADE-komennot, jotka muokattaessa tai poistettaessa päätaulun tietueita varmistavat, että myös kaikkiin tähän päätauluun viittaaviin tietueisiin tehdään samat muutokset. Käyttäjä ei pysty suoraan käyttöliittymän kautta muokkaamaan päätaulujen id-kenttiä, mutta ON UPDATE CASCADE-komento päätettiin kuitenkin lisätä, jos syystä tai toisesta päätaulujen id-kentät muuttuisivatkin tai esimerkiksi jatkokehityksen puitteissa niitä täytyisi muokata. Luontitiedostoon lisättiin myös asetukset-taululle oletustietoja.

Tietokanta asennettiin aluksi työn tekijän tietokoneelle testausta varten ajamalla luontilauseketiedosto PHPMyAdmin-ohjelman sql-komentoikkunan kautta. Tietokannan kenttien toimivuuden testaamiseksi kantaan syötettiin mahdollisimman autent-

tista tietoa. Viite-eheyden toimivuutta testattiin poistamalla ja muokkaamalla päätauluihin lisättyjä tietueita.

6.2 Sovelluksen ohjelmointi

Suunnitellun pohjalta alettiin toteuttaa sovellusta Netbeans-kehitystyökalulla. Kehityksen aluksi suunniteltiin jokaisen luokan tarvitsemat attribuutit sekä hahmoteltiin tarvittavia metodeja ja luokkien rakennetta. Aluksi ohjelmointia tehtiin toiminnan kannalta kriittisimmille luokille pääosin pseudokoodisesti eli kirjoittaen ohjelmakoodia ilman kirjaimellista syntaksia. Kun tärkeimmät luokat saatiin hahmoteltua, korjattiin niihin tarvittava syntaksi, jotta niiden toimivuutta voitiin testata.

Kuten luokkakaaviossa (kuvio 2) suunniteltiin, ylemmän tason Base-luokka tuli toteuttavaksi luokaksi sisältäen mm. tietokannan hallintaan tarvittavat select-, insert-, update- ja delete-metodit. Base-luokka periytettiin kaikille alemman tason lapsiluokille, jotta ne voisivat käyttää sen operaatioita (luokat on kuvattu luvussa 5.1). Tietokantayhteyden hallinnoimiseksi muodostettiin Database-tietokantaluokka, johon sisällytettiin tietokantayhteyteen tarvittavia operaatioita, kuten tietokannan valinta, yhdistäminen ja katkaisu. Database-luokasta muodostetun tietokantaolion kautta saatiin muille sovelluksen operaatioille yhteys tietokantaan.

Tietokannan hallintaan tarvittavia toimintoja silmällä pitäen kirjoitettiin jokaiselle lapsiluokalle tarvittavat muuttujat ja kohdetaulun kentät kokoavat metodit (listaMuuttujista() & tauluKentat()). Tietokantaoperaatiot toteutettiin antamalla näiden listojen arvot toteuttavan luokan metodien sql-kielisiin lauseisiin. Tällä rakenteella pyrittiin välttämään tarpeetonta ohjelmakoodin toistoa. Kuviossa 5 on esimerkki tästä.

```
public function insert()
        //Lisää tietueen kantaan
 {
             //Lisätään listaMuuttujista()-metodin arvot
             //tauluKentat()-metodin palauttamiin kenttiin
             $this->query = ("INSERT INTO " . $this->taulu .
             '(' .$this->tauluKentat().')' . " VALUES
             (" .$this->listaMuuttujista().")")
                 or die("Tallennus ei onnistunut, koeta uudestaan!");
             //Suoritetaan
             $tulos=mysql_query($this->query);
             //Tarkistetaan onnistuiko
             if ($tulos) {
             return true;
             }
             else {
             return false;
             }

 }
```

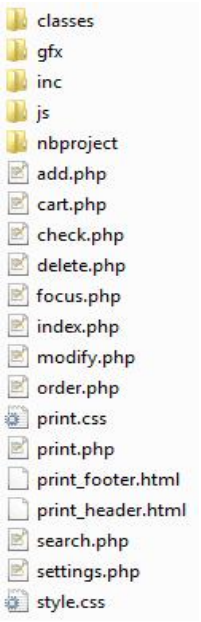
KUVIO 5. Esimerkki kehitetyn tietokantasovelluksen metodeista

Järjestelmästä tuli voida suorittaa tietokantahakuja 7:llä eri kriteerillä. Tämä monivalintahaku toteutettiin kirjoittamalla jokaiselle hakukriteerille omat metodinsa. Kuviossa 6 on esimerkki haeTavarat-metodista, joka kutsuu toista, tietokannasta hakevaa query-metodia, joka suorittaa tietokantahaun käyttöliittymästä saadun syötteen perusteella. haeTavarat-metodia voidaan kutsua hakutulokset tulostavasta käyttöliittymätiedostosta esimerkiksi tällä tavoin, $db->haeTavarat($hakusana), jossa db-muuttuja sisältää tietokantayhteyden ja hakusana-muuttuja käyttöliittymän hakukentästä saamansa syötteen.

```
    public function haeTavarat($nimi)
{
    //hakee hakusanalla tietokannasta tavarat

    //kutsutaan tietokannasta hakevaa
    //query()- metodia parametrein(taulu,kannan
    //kenttä josta haetaan = hakukriteeri, ja hakusana)
    $this->query("tavara","nimi" ,$nimi);
    //asetetaan tavarat-taulukko
    $tavarat = array();
    //käydään tietokantahaun
    //palauttamat tiedot silmukassa läpi
       while ($rivi = mysql_fetch_array($this->tulos)) {
    //uusi Tavara-olio tavara-muuttujaan
    $tavara = new Tavara($this->database,"tavara");
    //asetetaan Tavara-luokan muuttujiin haun palauttamat
    //arvot set-metodeilla
    $tavara->setNimi($rivi["nimi"]);
    $tavara->setKuvaus($rivi["kuvaus"]);
    $tavara->setID($rivi["id"]);
    $tavara->setKate($rivi["kate"]);
    $tavara->setOstohinta($rivi["ostohinta"]);
    $tavara->setVarastossa($rivi["varastossa"]);
    //ladotaan tavarat-taulukkoon silmukassa
    //läpikäydyt tiedot ja palautetaan id:t aina
    //taulukon avaimiksi get-metodilla
    $tavarat[$tavara->getID()] = $tavara;
    }
    return $tavarat;
}
```

KUVIO 6. Esimerkki kehitetyn tietokantasovelluksen metodeista

Käyttöliittymän tarvitsevat sivut ohjelmoitiin pääasiassa vaatimusmäärittelyssä asetetun priorisaation perusteella, eli pakollisten vaatimusten tarvitsemat sivut toteutettiin ensin ja valinnaiset näiden jälkeen. Kuviossa 7 on esitelty sovelluksen hakemistorakenne. Classes-kansio sisältää järjestelmän luokat, jotka tarjoavat palveluita päätason käyttöliittymän tiedostoille. Gfx-kansio sisältää järjestelmän käyttöliittymässä käytetyt kuvat. Kaikki kuvat on konvertoitu mahdollisimman pienikokoisiksi, jotta niiden lataus olisi nopeaa. Inc-kansio sisältää käyttöliittymän kiinteiden ylä- ja alaelementtien tiedostot, jotka sisällytetään jokaiseen käyttöliittymään. Js-kansiossa on javascript-tiedosto, joka sisältää kaikki järjestelmässä käytetyt javascript-funktiot. Nbproject-kansio on Netbeans-kehitystyökalun automaattisesti luoma projektikansio, joka poistetaan lopullisesta asennuksesta.



KUVIO 7. Sovelluksen hakemistorakenne

6.3 Käyttöliittymän toteutus

Järjestelmän käyttöliittymien tarvitsevat toiminnallisuudet toteutettiin suunnitteluvaiheessa (luku 5.6) hahmoteltujen html-kielisten näkymien pohjalta lisäämällä niihin tarvittavaa grafiikkaa ja yhdistämällä toiminnallisuudet suunniteltuihin painikkeisiin. Toimintopainikkeisiin lisättiin toimintoja symboloivat kuvat ulkoasun kohentamiseksi ja jotta ne olisivat visuaalisesti helppo yhdistää kohdetoimintoihin. Kuvat ladattiin netin ilmaisista kuvakirjastoista. Käyttöliittymän tarvitsemat <table>-rakenteiset taulukot toteutettiin Table-apuluokalla, jonka käytöllä tehostettiin tarvittavien taulukoiden luomista.

Laskujen ja tilausten luomiseksi järjestelmään toteutettiin ostoskori-toiminto, johon voidaan listata järjestelmän etusivulta tai hakunäkymästä valittuja tuotteita ja palveluita. Listattujen kohteiden tuli säilyä ostoskorissa käyttäjän liikkuessa myös muilla järjestelmän sivuilla. Tämä ongelma ratkaistiin PHP:n sessiomuuttujan avulla ($_SESSION[]), johon tallennettiin käyttöliittymästä lisättyjen kohteiden id-kenttien arvot, valitut tuote- ja palvelumäärät sekä listauksen yhteishinta. Id-arvojen perus-

teella ostoskori onnistuu tunnistamaan listatut kohteet. Ostoskori-näkymä on esitelty kuviossa 8.

KUVIO 8. Ostoskori-näkymä (testidatalla)

Käyttöliittymän kenttien syötetietojen tarkastus katsottiin tärkeäksi tekijäksi järjestelmän toimimisen sekä tietoturvan kannalta. Ratkaisuksi päädyttiin tarkastelemaan

kenttiin syötettyjä merkkejä PHP:n preg_match()-funktiolla, johon määriteltiin kaikki sallittavat syötteet. Tämän listan ulkopuoliset merkit antavat ilmoituksen käyttäjälle virheellisestä syötteestä. Sallittavia merkkejä tuli harkita tarkasti, jotta sallittujen merkkien listan ulkopuolelle ei jätettäisi mitään käytössä oikeasti tarvittavaa merkkiä. Preg_match-funktiota käytetään esimerkiksi näin, preg_match('/^[0-9\.]{1,}$/', $_POST["ostohinta"]). Funktio tarkastaa, sisältääkö ostohinta-kentän syöte muita merkkejä kuin sallitut numerot, pisteet ja välilyönnit. Kaikki tietokantaan kulkevat syötteet tarkastetaan myös MySQL-kirjaston mysql_real_escape_string()-funktiolla, joka lisää erikoismerkkejä sisältäviin syötteisiin kenoviivat haitallisten syötteiden estämiseksi. Käyttöliittymän joitain toteutettuja näkymiä esitellään liitteessä 7.

6.4 Testaus

Testausta suoritettiin koko toteutusvaiheen ajan jokaisen metodin tai toiminnallisuuden toteutuksen jälkeen tarkastelemalla ohjelmakoodin logiikkaa ja antamalla metodeille eri parametrejä lähdekoodissa. Järjestelmän käyttöliittymän kautta seurattiin, toteuttivatko testikohteet odotettuja asioita. Virheiden jäljittämisen avuksi määritettiin kehitysvaiheessa PHP:n asetustiedostosta virheiden tulostus päälle (display_errors). Lisäksi kaikkiin tietokantaan liittyviin operaatioihin lisättiin epäonnistuneiden suoritusten palauttamaksi tulosteeksi mysql_error()-funktion virheilmoitus.

Käyttöliittymän toimintaa testattiin myös syöttämällä erilaisia arvoja järjestelmän tekstikenttiin. Lähtökohtaisesti pyrkimyksenä oli saada näkymissä aikaan jotakin suunnittelematonta. Kun tiedossa oli, mitä tekstikenttiin normaalisti syötetään, aloitettiin testaus syöttämällä kenttiin normaalista poikkeavaa tietoa. Esimerkiksi tavaran ostohinta-kenttä sai sisältää vain numeroita 0–9 ja pisteitä, joten syöteaineistona käytettiin mm. -merkkisiä lukuja, kirjaimia ja erikoismerkkejä. Lisäksi kenttiin syötettiin sql-kielisiä tietokantahakuja, joilla pyrittiin testaamaan järjestelmää haitallisten sql-injektioiden varalta, joissa pääosin järjestelmän käyttöliittymää hyödyntämällä suoritetaan erilaisia, usein haitallisia tietokantahakuja.

7 KÄYTTÖÖNOTTO

Tässä luvussa kuvataan varastonhallintasovelluksen järjestelmätestaus, käyttöympäristön tarvitsemat ohjelmistot sekä lopuksi järjestelmän asennus lopulliseen ympäristöönsä.

7.1 Järjestelmätestaus

Ennen tietokantasovelluksen varsinaista käyttöönottoa järjestettiin erillinen palaveri toimeksiantajan kanssa, jossa testattiin järjestelmän toimintaa kokonaisuutena. Tässä vaiheessa toimeksiantaja pyrki suorittamaan joitain järjestelmän tärkeimpiä toiminnallisuuksia pääosin itsenäisesti työn tekijän avustaessa tarvittaessa. Tämän hyväksymis- tai ”julkaisutestauksen” puitteissa suoritettiin järjestelmän käyttöliittymän kautta määrittelyvaiheessa kuvattuja käyttötapauksia (liite 2), kuten kohteiden (esim. tavarat ja palvelut) lisäystä järjestelmään ja laskun muodostamista tuote ja palvelulistoista. Lisäksi käytiin läpi toiminnallisten vaatimusten listaa (liite 1), jossa toimeksiantaja arvioi, toteutuvatko järjestelmässä listaan kootut toiminnot.

Tämän testauksen tarkoituksena oli osoittaa vähintään järjestelmän pakolliset vaatimukset täytetyiksi, jolloin kehitettävän työn oltiin sovittu olevan valmis. Testauksen päätteeksi sovittiin järjestelmän olevan valmis käyttöönottoasennukseen.

7.2 Tarvittavat ohjelmistot

Varastonhallintajärjestelmää tullaan käyttämään työn tilaajan kannettavalla tietokoneella, jossa on asennettuna Windows 7 -käyttöjärjestelmä. Loppukäyttäjän koneella tulisi olla päivitettyinä internet- selainten uusimmat versiot, jotta sovelluksen käyttöliittymä toimisi suunnitellun näköisenä. Sovellus kehitettiin ja

testattiin pääosin Mozilla Firefox (versio 12.0) -selaimella, jolla sitä kehoitettiin myös käyttämään. Järjestelmää testattiin lisäksi muutamalla muulla selaimella, kuten Internet Explorerilla (versio 9) ja Operalla (versio 11.60), joten näitäkin selaimia on mahdollista käyttää, ilman suuria muutoksia käyttöliittymissä. Käyttäjän selaimessa täytyy lisäksi olla tuki javascript-komentosarjoille, joita osa järjestelmän toiminnallisuuksista käyttää. Vaatimusmäärittelyn ei-toiminnallisessa osiossa (liite 1) on kuvattu palvelimelta vaaditut ohjelmistot.

7.3 Asennus käyttöympäristöön

Järjestelmän asennus yrittäjän tietokoneeseen oli lopuksi helppo suorittaa, sillä järjestelmä oli rakennettu samassa ympäristössä ja pääosin samoilla ohjelmistoilla, johon se tultiin asentamaankin. Järjestelmä tuli toimimaan vielä ainakin tässä vaiheessa paikallisesti yrittäjän tietokoneella. Jatkossa, toimeksiantajan tutustuttua kunnolla järjestelmään, suunniteltiin järjestelmä siirrettäväksi ulkoiselle palvelimelle, jossa esimerkiksi tietokannan varmuuskopiointi saataisiin automatisoitua, eikä järjestelmä olisi riippuvainen vain yhdestä koneesta. Tässä vaiheessa tulisi tietysti huomioida myös käyttäjän autentikointi sekä tietoturvaseikat paremmin.

Toimeksiantajan tietokoneeseen asennettiin aluksi tarvittava ohjelmisto uusimmasta WampServer-paketista (versio 2.2), jonka luoman palvelinhakemiston juureen sovelluksen lähdekoodit sisältävä kansio siirrettiin. WAMP-ohjelmiston käyttö on asennuksen kannalta hyvin vaivatonta, sillä sen asennustiedoston suorittamalla asentuivat kerralla kaikki tässä projektissa kehitetyn tietokantasovelluksen käyttämät ohjelmat, kuten Apache-palvelinojelmisto (versio 2.2.21), MySQL-tietokantaohjelmisto (versio 5.5.20) sekä PHP (versio 5.3.10). Asennuksen jälkeen tarkistettiin vielä WAMP-ohjelmiston valikosta, että PHP:n tarvitsemat laajennukset, kuten tietokantaoperaatioihin tarvittava mysqli, olivat käytössä, sekä asetettiin palvelin toimimaan aina käynnistyessään offline-tilassa.

Kun tiedostot saatiin sijoitettua palvelinhakemistoon, luotiin tietokanta asennusta varten kootusta luontilausetiedostosta (liite 6), joka suoritettiin WAMP-ohjelmiston mukana tulleen PHPMyAdmin-ohjelman sql-ikkunassa. Kun tietokanta saatiin myös luotua, voitiin järjestelmän toimintaa kokeilla kokonaisuutena. Järjestelmälle suoritettiin lopullisessa käyttöympäristössään myös testausta varmuuden vuoksi, jotta varmistuttaisiin, että esimerkiksi jotkut selaimen asetukset eivät vaikuttaisi käyttöliittymän toimintaan negatiivisesti. Oletuksena kuitenkin oli, että koska ohjelmisto ja käyttöjärjestelmä olivat vastaavanlaiset kuin työn tekijän tietokoneilla, järjestelmän toiminta vastaisi kehitysympäristössä koettua.

Työn tilaaja ei ottanut varastonhallintajärjestelmää suoraan tuotantokäyttöön, vaan halusi aluksi opetella järjestelmän käyttöä kaikessa rauhassa. Käyttöönoton yhteydessä sovittiin toimeksiantajan kanssa kuukauden mittaisesta ylläpitoajasta, jonka aikana työn tekijä suorittaisi tarvittaessa ylläpitotoimenpiteitä, bugikorjauksia ja opastaisi järjestelmän käytössä.

8 TYÖN ARVIOINTI

Tässä luvussa arvioidaan kehitysprojektista saatuja tuloksia ja haetaan vastauksia tutkimuskysymyksiin. Lopuksi pohditaan yleisesti kehitysprojektin toteutusta ja tehtyjä ratkaisuja.

8.1 Työn tulosten arviointi

Tämän opinnäytetyön tuloksia ovat raportti kehitysprojektin vaiheista sekä toimeksiantajan varastonhallintamenetelmien tehostuminen ja ajan säästö tuoteinventaariolistojen ja tuotetilausten kokoamisessa, tuotteiden tietojen, saldojen ja hintojen hallinnassa sekä laskutuksessa ja laskujen arkistoinnissa. Koska kehitetty järjestelmä otettiin projektin lopuksi vasta testikäyttöön, jossa toimeksiantaja tutustuu siihen ja päättää sitten, missä mittakaavassa sitä tullaan käyttämään, saatuja hyötyjä on vielä tässä vaiheessa vaikea arvioida. Lopullisen käyttöönoton myötä ja ajan kuluessa nähdään vasta, miten toiminta tehostuu ja aikaa säästyy.

Kehitetty varastonhallintajärjestelmä sekä tuotettu dokumentaatio tehtiin kohdeyrityksen lähtökohdista ja tarpeista, joten sen soveltaminen muuhun yhteyteen ei todennäköisesti käy aivan suoraan, joskin jatkokehityksen ja räätälöinnin puitteissa se on mahdollista. Raportti on pyritty kirjoittamaan niin, että ohjelmistokehityksestä kiinnostuneet tai vastaavanlaista järjestelmää suunnittelevat voivat siitä toivon mukaan saada hyötyä.

Miten kehitetään tietokantasovellus kohdeyrityksen tarpeisiin?
Kehitystyö jaettiin pääosin perinteisen vesiputousmallin mukaisiin vaiheisiin, joiden toteutuksella pyrittiin vastaamaan tälle opinnäytetyölle asetettuihin alakysymyksiin. Tähän pääkysymykseen saadaan vastaus näistä täsmentävistä alakysymyksistä.

Millaisia toiminnallisuuksia tarvitaan?

Vastaus tähän alakysymykseen selvitettiin projektin vaatimusmäärittelyssä, luvussa 4.2, jossa toimeksiantajaa haastattelemalla koottiin lista toiminnallisista vaatimuksista (liite 1), joita järjestelmään tuli toteuttaa. Toimeksiantajan priorisoidessa vaatimukset pakollisesti toteutettaviin toiminnallisuuksiin sekä aikataulun salliessa toteutettaviin valinnaisiin toiminnallisuuksiin, tärkeimmiksi toiminnallisuuksiksi luettiin seuraavat:

- Tietokannan hallitsemiseen (uuden lisääminen, muokkaus, poisto ja kohteen tarkastelu sekä haku) liittyvät toiminnot tuotteiden, toimittajien sekä palveluiden osalta.
- Laskujen luonti ja tulostus järjestelmässä olevista tuotteista ja palveluista.
- Tuoteinventaariolistojen kokoaminen kaikista järjestelmässä olevista tuotteista sekä niiden tulostaminen.
- Tuotteiden varastosaldoihin asetettavat saldohälytykset, joiden pohjalta käyttäjää kehotetaan täydentämään varastoa tuotteiden vähetessä.

Kysymykseen, millaisia toiminnallisuuksia tarvitaan, vastataan myös tärkeimmistä toiminnallisista vaatimuksista johdetuilla järjestelmän käyttötapauksilla (luku 4.3), joita on listattu liitteessä 2.

Millainen tietokannan tulee olla?

Tähän alakysymykseen haettiin vastausta luvussa 5.4 – Tietokannan suunnittelu. Tietokannaksi valittiin MySQL, sillä työn tekijän aiemmasta kokemuksesta se sopi hyvin yhteen PHP:n kanssa. Suunnittelun perusteella tietokantaan muodostui relaatiomallinen taulurakenne, jota kuvataan tietokannan ER-kaaviossa (liite 3). Tietokannan taulujen viite-yhteydet, suhteet sekä kenttien määritykset huomioon ottaen muodostui tietokantarakenteen kuvaava relaatiotietokanta, joka on kuvattu liitteessä 4.

Lopulliseen tietokantaan tarvittiin 11 taulua:

- asetukset, johon tallennetaan mm. tavaroiden hintatietoihin, saldohälytyksiin sekä laskuihin ja tilauksiin vaikuttavia tietoja. Näitä tietoja ovat mm. tuotteiden ja palveluiden arvonlisäverot, yleinen saldohälytys kaikille tuotteille sekä yrityksen tiedot.
- halytys, johon tallennetaan hälyttävien tuotteiden pääavainten arvot sekä tuotteelle määritetty saldoraja. Tauluun tallennetaan tietoja laskutoiminnossa, kun kohdetuotteen varastosaldo saavuttaa määritetyn hälytysrajan.
- lasku, johon arkistoidaan tallennetut laskut sisältäen asiakastiedot sekä laskun tiedot.
- laskutavara, johon tallennetaan viiteavaimiksi laskuun liittyvien tuotteiden ja kohdelaskun pääavainten arvot sekä tuotteiden kappalemäärä ja yhteishinta.
- laskutyo, johon sisältyy viiteavaimina kohdelaskun ja palveluiden pääavainten arvot sekä palveluiden määrä ja yhteishinta.
- tavara, joka sisältää kaikki järjestelmään lisätyt tuotteet. Tavara-taulun tietueille voidaan määrittää erikseen saldohälytykset, joiden kohdalla hälytetään yleisen asetukset-tauluun tallennetun hälytysmäärän sijaan.
- tilaus, joka sisältää tallennetut tilaukset sekä niiden lisätiedot. Toimii tilauslistojen päätauluna.
- tilaustavara, johon tallennetaan viiteavaimiksi kohdetilausten ja tuotteiden pääavainten arvot sekä tuotemäärä.
- tilaustoimittaja, joka sisältää viiteavaimina kohdetilausten sekä toimittajien pääavainten arvot. Taulu lisättiin, koska yhteen tilauslistaan tuli voida määrittää useita toimittajia.
- toimittaja, johon tallennetaan tuotteiden toimittajien yhteystiedot sekä haluttaessa lisätietoja.
- tyo, joka sisältää kohdeyrityksen tarjoamat palvelut, kuvauksen sekä hinnan.

Millaisia käyttöliittymiä tarvitaan?

Tähän alakysymykseen vastattiin käyttöliittymäsuunnittelun (luku 5.6) perusteella. Tarvittavat käyttöliittymät johdettiin vaatimusmäärittelyistä ja käyttötapauksista,

joiden perusteella käyttöliittymiä hahmoteltiin käyttöliittymädiagrammilla (liite 5). Diagrammissa kuvataan kaikki järjestelmän tarvitsevat näkymät, sekä navigointi niiden välillä. Käyttöliittymien sisältö on hyvin riippuvainen tietokannan sisältämistä tiedoista, joita niissä käsitellään.

Nyrkkisääntöinä järjestelmän käyttöliittymien suunnittelussa olivat yksinkertaisuus, helppo opittavuus ja yhdenmukaiset näkymät. Käyttöliittymiin suunniteltiin kolme eri osaa. Käyttöliittymien yläosa tulisi pysymään jokaisessa näkymässä muuttumattomana. Yläosaan sijoitettiin tärkeimmiksi katsottuja toimintopainikkeita, jotka tulisivat olla saatavilla kaikilta sivuilta. Näitä olivat haku-kenttä ja hakukriteerien monivalintapainikkeet, asetukset, ostoskori sekä inventaariolistan luonti. Käyttöliittymien keskiosaan tulostuvat järjestelmän eri näkymät. Toinen käyttöliittymän kiinteä elementti on alaosa, joka tarjoaa jokaisessa näkymässä mahdollisuuden palata takaisin edelliselle sivulle.

Suunnittelun perusteella järjestelmässä tarvittavia käyttöliittymiä olivat:

- Etusivu, joka näyttää listauksen varaston tavaroista ja niiden tärkeimmistä tiedoista. Jokaisen listatun kohteen kohdalle sijoitettiin valintapainikkeet, josta sitä voidaan tarkastella lähemmin, muokata, poistaa tai lisätä ”ostoskoriin” lasku- ja tilauslistaukseen. Tuotteita voidaan myös lisätä tästä näkymästä.
- Haku, joka esittää hakukriteerien ja hakusanan perusteella haetut tiedot taulukkomuotoisena listana. Tästä näkymästä listatuille kohteille voidaan suorittaa samoja toimintoja kuin etusivulla.
- Ostoskori, jossa käyttäjä voi tarkastella listattuja tuotteita ja palveluita, ja muodostaa listauksen pohjalta uuden laskun tai tilauksen.
- Lasku, jossa käyttäjä voi luoda laskun asiakkaalle listattujen tuotteiden ja palveluiden pohjalta lisäämällä listaukseen asiakkaan sekä laskun tiedot. Lasku voidaan myös tulostaa tästä näkymästä.
- Tilaus, jossa ostoskoriin listatuista tuotteista voidaan luoda tilauslista. Tilattavat tuotteet jaetaan tuotteelle asetetun toimittajan mukaan ryhmiksi. Tilauslista voidaan myös tulostaa tilausnäkymästä.

- Asetukset, jossa voidaan määritellä järjestelmän toimintoihin, kuten laskuun, tilaukseen sekä hälytyksiin vaikuttavia tietoja.
- Haettujen kohteiden kaikkien tietojen tarkastelu-, muokkaus- ja poistonäkymät.

Käyttäjän ohjeistamiseksi jokaisen näkymän yläosaan lisättiin ”palaute”-tila, jonka kautta järjestelmä kommunikoi käyttäjän kanssa tulostamalla palautetta ja ohjeistuksia suoritettujen toimintojen perusteella. Poikkeuksellisesti joissain tärkeissä tapahtumissa käyttöliittymät tulostavat myös ponnahdusikkunan tapahtuman suorituksen vahvistamiseksi tai ilmoittavat käyttäjälle esimerkiksi varastosaldojen vähentymisestä hälyttävälle tasolle.

Miten testaus suoritetaan?

Tähän alakysymykseen haettiin aluksi vastausta testauksen suunnittelun perusteella, luvussa (5.5), jonka pohjalta testausta päädyttiin suorittamaan testausvaiheessa (luku 6.4) hieman lähdekoodin sekä käyttöliittymän tasoilla. Lähdekooditasolla testattiin toteutettujen metodien suoriutumista tehtävistään erilaisilla parametreillä. Tuloksia seurattiin pääosin käyttöliittymän kautta. Järjestelmän käyttäytymistä erilaisiin syötteisiin testattiin käyttöliittymätasolla mm. raja-arvoilla sekä ei-sallituilla syötteillä. Tuotetulle dokumentaatiolle ja suunnitelmille suoritettiin katselmuksia sekä staattista testausta. Projektissa testauksesta vastasi pääosin työn tekijä itse, mutta katselmoinneissa sekä järjestelmän käyttäytymisen testaamisessa hyödynnettiin muutamaa ulkopuolista testaajaa.

Ennen järjestelmän varsinaista käyttöönottoa suoritettiin järjestelmätestaus (luku 7.1), jossa pyrittiin osoittamaan järjestelmän toteuttavan vähintään vaatimusmäärittelyssä asetetut pakolliset vaatimukset. Tässä vaiheessa toimeksiantaja testasi järjestelmää pääosin itse suorittamalla määrittelyvaiheessa kuvattuja käyttötapauksia (liite 2).

8.2 Pohdinta

Tehdyt ratkaisut

Kehitysprojektin vaihejaotteluksi valittu perinteinen vesiputousmalli sopi varsin hyvin tähän projektiin toimeksiantajan tietäessä heti alusta asti varsin tarkasti millaisia ominaisuuksia järjestelmässä tulee olla. Tästä johtuen perusvaatimukset säilyivät lähes muuttumattomina loppuun asti. Tarkkojen vaatimusten pohjalta oli myös helppo aloittaa järjestelmän toteutuksen suunnittelu. Oliopohjaisen toteutusmallin hyödyntäminen osoittautui myös hyväksi valinnaksi täysin proseduraalisen toteutuksen sijaan. Sen suunnittelu vei hieman aikaa projektin aikataulusta, mutta toisaalta tällä ratkaisulla säästettiin aikaa toteutuksessa. Luokkarakenteen hyödyntämisellä vähennettiin tarpeetonta ohjelmakoodin toistoa. Tällä tavoin myös järjestelmän ylläpito ja jatkokehitys helpottuvat kun rakenne on selkeämpi ja muutoksia ei tarvitse välttämättä tehdä moneen paikkaan koodia. PHP:n ja MySQL-tietokannan yhdistelmä oli varsin toimiva, joskin tietokantaoperaatioissa oltaisiin voitu käyttää myös joitain MySQL:n kehittyneempiä ominaisuuksia, kuten tallennettuja proseduureja (stored procedures) ja näkymiä (views).

Toteutuksessa oltaisiin voitu hyödyntää jossain määrin AJAX (Asynchronous JavaScript And XML)-tekniikkaa, jolla joistakin toiminnoista, kuten ostoskorista saataisiin vuorovaikutteisempia sivujen uudelleenlataustarpeen vähentyessä. Yleisesti joitain järjestelmän toimintoja oltaisiin myös voitu monipuolistaa. Hälytystoiminto on varsin rajoittunut ja sitä oltaisiin voitu parantaa esimerkiksi antamalla käyttäjälle mahdollisuus reagoida joillain toiminnoilla saamiinsa hälytyksiin. Järjestelmä voisi esimerkiksi luoda automaattisesti listan kaikista hälyttävistä tuotteista uutta tilauslistaa varten. Järjestelmän testauksesta jouduttiin aikataulullisista syistä jättämään pois projektin alussa suunniteltu testitapausten mukainen yksikkötestaus.

Yleistä pohdintaa työstä

Tämä kehitysprojekti ei anna yleispätevää mallia siitä, kuinka kehitetään vastaavanlainen tietojärjestelmä, vaan kuvaa yhden toteutustavan, jonka mukaan ohjelmisto-

tuote voidaan kehittää. Toteutus ei seuraa kirjaimellisesti perinteisen vaihejaon mukaista vesiputousmallia, vaan sitä sovellettiin tähän projektiin sopivaksi. Esimerkiksi testausta suoritettiin koko toteutusvaiheen ajan, ja käyttöliittymää suunniteltaessa alustavia näkymiä ”protoiltiin” toimeksiantajalle. Tiivis yhteydenpito toimeksiantajan kanssa oli muutenkin varsin tärkeää työn onnistumisen kannalta, sillä tarvittaviin muutoksiin voitiin näin reagoida nopeammin.

Kehitetyssä tietokantasovelluksessa toteutuivat kaikki sille asetetut pakolliset- sekä valinnaiset vaatimukset, joten projektia voidaan tältä osin pitää onnistuneena. Kaupallisiin, tai avoimen lähdekoodin varastonhallintajärjestelmiin verrattaessa, toteutettu järjestelmä on toiminnallisuuksiltaan vielä tässä versiossaan varsin suppea. Yhden miehen resurssit sekä kohtuullisen tiukka aikataulu rajoittivat jossain määrin toteutusta.

Kehitettyyn varastonhallintajärjestelmään jäi potentiaalia jatkokehitykselle. Yksi tällainen kohde voisi olla jo projektin alussa suunniteltu viivakoodinlukijan hyödyntäminen. Tämä sovittiin kuitenkin jätettäväksi pois tästä järjestelmän versiosta ja lisättäväksi mahdollisesti myöhemmin tarvittavan laitteiston hankinnan yhteydessä. Viivakoodinlukijan hyödyntäminen tulisi vaatimaan mm. laitteen lukeman syötteen tarkastelua sovelluksen puolelta sekä viivakoodillisten EAN-tarrojen tulostelua erikseen jokaiseen tuotteeseen. Lisäksi yksi suuri kehitysmahdollisuus olisi tilaustoiminnossa, joka voitaisiin integroida osaksi tuotteiden toimittajien omia järjestelmiä toteuttamalla rajapinta järjestelmän tilauslistatoiminnon sekä toimittajien omien verkkotilausjärjestelmien välille. Tätä voitaisiin käyttää esimerkiksi automatisoimaan uusia tilauksia. Tätä tosin vaikeuttaa se, että suurimmalla osalla toimittajista on erilaiset tilausjärjestelmät, joten se vaatisi usean eri integraatiojärjestelmän toteuttamista. Jatkokehityksen kannalta olisi myös kiinnostavaa tutkia radiotaajuutta hyödyntävän etätunnistustekniikan (RFID) mahdollisuuksia kohdeyrityksen näkökulmasta.

LÄHTEET

Black, R. 2009. Managing the Testing Process - Practical Tools and Techniques for Managing Hardware and Software Testing. Viitattu 24.3.2012. Http://books.google.fi.

Glenford, J.M., Sandler, C. & Badgett, T. 2012. The art of software testing. Viitattu 4.4.2012. Http://books.google.fi.

Haikala, I. & Märijärvi, J. 2006. Ohjelmistotuotanto. Helsinki: Talentum.

Hernandez, M.J. 2003. Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design. Viitattu 3.4.2012. Http://books.google.fi.

O'Reilly & Associates, Inc. 2003. Web database applications with php & mysql. Chapter 1. Database applications and the web. Viitattu 16.4.2012. Http://docstore.mik.ua/orelly/webprog/webdb/ch01_01.htm.

Piasecki, D. N.d. Warehouse Management Systems (WMS). Viitattu 31.3.2012. Http://www.inventoryops.com/warehouse_management_systems.htm.

Richards, G. 2011. Warehouse Management – A complete guide to improving efficiency and minimizing costs in the modern warehouse. Kogan Page Ltd. Viitattu 20.4.2012. Http://janet.amkit.fi/.

Sommerville, I. 2007. Software Engineering. 8. painos. Pearson Education Limited.

Thompson, G. N.d. Jyväskylän ammattikorkeakoulun Testaus-opintojakson luentomateriaali. Viitattu 3.4.2012.

Vliet, H.V. 2008. Software Engineering – Principles and Practice. 3. painos. John Wiley & Sons Ltd.

LIITTEET

LIITE 1. Vaatimusmäärittely

Toiminnalliset vaatimukset

ID	Kuvaus	Luokka
FR_1	Käyttäjä voi lisätä tuotteen järjestelmään	Pakollinen
FR_2	Käyttäjä voi poistaa tuotteen järjestelmästä	Pakollinen
FR_3	Käyttäjä voi muokata järjestelmässä olevan tuotteen tietoja	Pakollinen
FR_4	Käyttäjä voi lisätä järjestelmässä olevalle tuotteelle saldohälytyksen	Pakollinen
FR_5	Käyttäjä voi muokata järjestelmässä olevan tuotteen saldohälytystä	Pakollinen
FR_6	Käyttäjä voi lisätä hinnan järjestelmässä olevalle tuotteelle	Pakollinen
FR_7	Käyttäjä voi muokata järjestelmässä olevan tuotteen hintaa	Pakollinen
FR_8	Käyttäjä voi lisätä toimittajan järjestelmään	Pakollinen
FR_9	Käyttäjä voi poistaa toimittajan järjestelmästä	Pakollinen
FR_10	Käyttäjä voi muokata järjestelmässä olevan toimittajan tietoja	Pakollinen
FR_11	Käyttäjä voi lisätä palvelun järjestelmään	Pakollinen
FR_12	Käyttäjä voi poistaa palvelun järjestelmästä	Pakollinen
FR_13	Käyttäjä voi muokata järjestelmässä olevan palvelun tietoja	Pakollinen
FR_14	Käyttäjä voi lisätä hinnan järjestelmässä olevalle palvelulle	Pakollinen
FR_15	Käyttäjä voi muokata järjestelmässä olevan palvelun hintaa	Pakollinen
FR_16	Käyttäjä voi muokata järjestelmän asetuksia	Pakollinen
FR_17	Käyttäjä voi muokata yrityksen tietoja	Pakollinen
FR_18	Käyttäjä voi lisätä laskun palveluista ja tuotteista	Pakollinen
FR_19	Käyttäjä voi poistaa järjestelmässä olevan laskun	Pakollinen
FR_20	Käyttäjä voi lisätä tilauksen	Valinnainen
FR_21	Käyttäjä voi poistaa järjestelmässä olevan tilauksen	Valinnainen
FR_22	Käyttäjä voi tarkastella järjestelmässä olevan tuotteen tietoja	Pakollinen
FR_23	Käyttäjä voi tarkastella järjestelmässä olevan toimittajan tietoja	Pakollinen
FR_24	Käyttäjä voi tarkastella järjestelmässä olevan palvelun tietoja	Pakollinen
FR_25	Käyttäjä voi tarkastella järjestelmässä olevan tilauksen tietoja	Valinnainen
FR_26	Käyttäjä voi hakea järjestelmässä olevaa tuotetta tuotenumeron tai nimen perusteella	Pakollinen
FR_27	Käyttäjä voi hakea järjestelmässä olevaa palvelua nimen perusteella	Pakollinen
FR_28	Käyttäjä voi hakea järjestelmässä olevaa toimittajaa nimen perusteella	Pakollinen
FR_29	Käyttäjä voi hakea hälytyslistan hälyttävistä tuotteista	Pakollinen
FR_30	Käyttäjä voi hakea järjestelmässä olevan tilauksen	Valinnainen
FR_31	Käyttäjä voi hakea järjestelmässä olevan laskun	Pakollinen
FR_32	Käyttäjä voi koota tuote- ja palvelulistan järjestelmässä olevista tuotteista ja palveluista	Pakollinen
FR_33	Käyttäjä voi tulostaa raportin järjestelmässä olevista tuotteista	Pakollinen
FR_34	Käyttäjä voi tulostaa järjestelmässä olevan laskun	Pakollinen
FR_35	Käyttäjä voi tulostaa järjestelmässä olevan tilauksen	Valinnainen
FR_36	Järjestelmä laskee järjestelmässä olevan tuotteen verollisen hinnan ja alv:n osuuden	Pakollinen
FR_37	Järjestelmä laskee järjestelmässä olevan palvelun verollisen hinnan ja alv:n osuuden	Pakollinen
FR_38	Järjestelmä laskee käyttäjän kokoaman tuote- ja palvelulistan verollisen- ja verottoman hinnan sekä alv:n osuuden	Pakollinen
FR_39	Järjestelmä hälyttää loppuvasta tuotteesta	Pakollinen

Ei-toiminnalliset vaatimukset

ID	Kuvaus	Luokka
NFR_1	Palvelimella tulee olla tietokantana MySQL versio 5.5.20	Pakollinen
NFR_2	Palvelimen tulee tukea PHP versiota 5.3.10	Pakollinen
NFR_3	Palvelinohjelmistona on Apache 2.2.21	Pakollinen
NFR_4	Järjestelmä tulee toimimaan paikallisesti	Pakollinen

LIITE 2. Käyttötapaukset

Tuotteen lisäys järjestelmään

Suorittajat:	Käyttäjä
Esiehdot:	Järjestelmä on käynnissä, tietokanta toiminnassa, käyttäjä on painanut "lisää tuote"-painiketta.
Kuvaus:	Käyttäjä syöttää tiedot kenttiin [Poikkeus 1: Kentässä kiellettyjä merkkejä] ja painaa ”tallenna”-painiketta
Poikkeukset:	1. Kentässä kiellettyjä merkkejä: Käyttäjä on syöttänyt johonkin kenttään kiellettyjä merkkejä. Käyttäjää pyydetään korjaamaan virheellisen kentän syöte ja yrittämään uudelleen.
Jälkiehdot:	Käyttäjä on lisännyt tuotteen järjestelmään, josta järjestelmä informoi käyttäjää.

Palvelun lisäys järjestelmään

Suorittajat:	Käyttäjä
Esiehdot:	Järjestelmä on käynnissä, tietokanta toiminnassa, käyttäjä on valinnut hakukriteeriksi ”työn” ja painanut "Hae"-painiketta sekä painanut avautuvasta näkymästä "lisää työ"-painiketta.
Kuvaus:	Käyttäjä syöttää tiedot kenttiin [Poikkeus 1: Kentässä kiellettyjä merkkejä] ja painaa ”tallenna”-painiketta.
Poikkeukset:	1. Kentässä kiellettyjä merkkejä: Käyttäjä on syöttänyt johonkin kenttään kiellettyjä merkkejä. Käyttäjää pyydetään korjaamaan virheellisen kentän syöte ja yrittämään uudelleen.
Jälkiehdot:	Käyttäjä on lisännyt palvelun järjestelmään, josta järjestelmä informoi käyttäjää.

Tietojen haku järjestelmästä

Suorittajat:	Käyttäjä
Esiehdot:	Järjestelmä on käynnissä, tietokanta toiminnassa.
Kuvaus:	Käyttäjä kirjoittaa haku-kenttään hakusanan [Poikkeus 1: Kentässä kiellettyjä merkkejä] ja valitsee hakukriteerit painamalla haluttua kriteeriä [Poikkeus 2: Kriteerejä ei valittuna] ja painaa "Hae"-painiketta [Poikkeus 3: Ei hakutuloksia].
Poikkeukset:	1. Kentässä kiellettyjä merkkejä: Käyttäjä on syöttänyt haku-kenttään kiellettyjä merkkejä. Käyttäjää pyydetään korjaamaan virheellinen syöte ja yrittämään uudelleen. 2. Kriteerejä ei valittuna: Hakukriteerejä ei ole valittu painettaessa "Hae"-painiketta. Käyttäjää informoidaan valitsemaan kriteerit ja yrittämään uudelleen. 3. Ei hakutuloksia: Käyttäjän hakusyötteellä ei löytynyt tuloksia. Käyttäjää informoidaan yrittämään uudelleen.
Jälkiehdot:	Käyttäjä on löytänyt halutut tiedot ja ne ovat tulostuneet käyttöliittymään.

Laskun lisääminen tuote/palvelu listasta

Suorittajat:	Käyttäjä
Esiehdot:	Järjestelmä on käynnissä, tietokanta toiminnassa, haluttu tuote/palvelu haettu.
Kuvaus:	Käyttäjä lisää halutun tuotteen/palvelun listaan painamalla "ostoskori"-painiketta ja valitsee avautuvasta näkymästä halutun tuotemäärän [Poikkeus 1: Määrä 0] ja painaa "lasku"-painiketta [Poikkeus 2: Tuotteita ei valittuna]. Käyttäjä syöttää asiakkaan tiedot avautuvan näkymän kenttiin [Poikkeus 3: kentässä kiellettyjä merkkejä] ja painaa "Hyväksy"-painiketta [Poikkeus 4: laskun numero jo käytössä].
Poikkeukset:	1. Määrä 0: Käyttäjä on valinnut määräksi 0:n. Käyttäjälle ilmoitetaan, että tuotteita/palveluita ei valittuna.

	2. Tuotteita ei valittuna: Järjestelmä informoi käyttäjää, että tuotteita/palveluita ei ole valittuna laskutettavaksi. 3. Kentässä kiellettyjä merkkejä: Käyttäjä on syöttänyt johonkin kenttään kiellettyjä merkkejä. Käyttäjää pyydetään korjaamaan virheellisen kentän syöte ja yrittämään uudelleen. 4. Laskun numero jo käytössä: Järjestelmässä on jo lasku samalla numerolla. Käyttäjää informoidaan tästä.
Jälkiehdot:	Käyttäjä on lisännyt laskun, josta järjestelmä informoi käyttäjää.

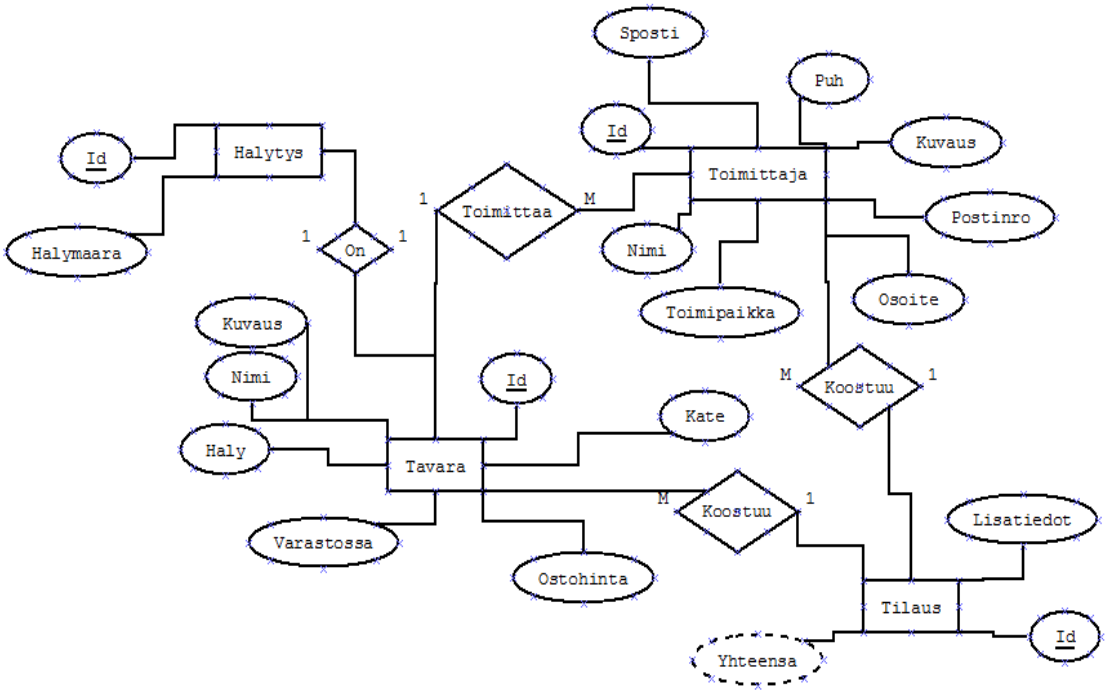
Tuoteinventaariolistan tulostaminen

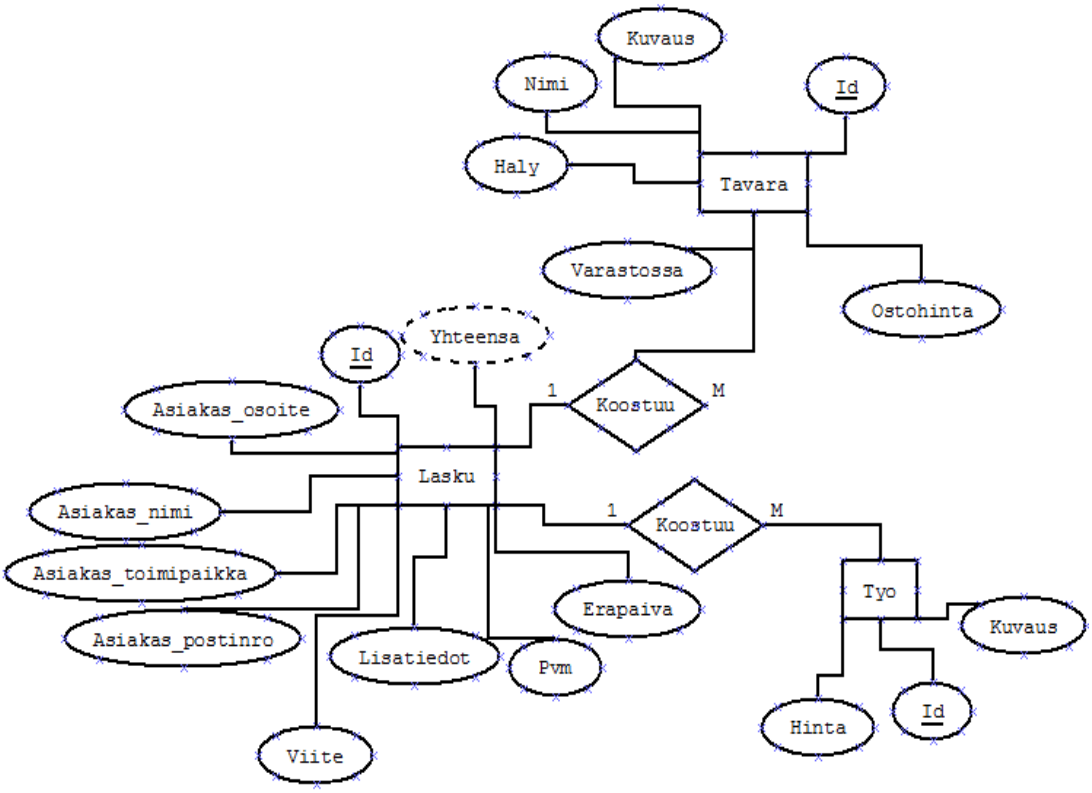
Suorittajat:	Käyttäjä
Esiehdot:	Järjestelmä on käynnissä, tietokanta on toiminnassa
Kuvaus:	Käyttäjä painaa "inventaariolista"-painiketta. [Poikkeus 1: Tuotteita ei lisättynä] Käyttäjä tulostaa listan selaimen tulostus toiminnolla.
Poikkeukset:	1. Tuotteita ei lisättynä: Tuotteita ei ole lisättynä järjestelmään. Käyttäjää informoidaan tästä.
Jälkiehdot:	Käyttäjä on tulostanut tuoteinventaariolistan.

Saldohälytyksen lisääminen tuotteelle

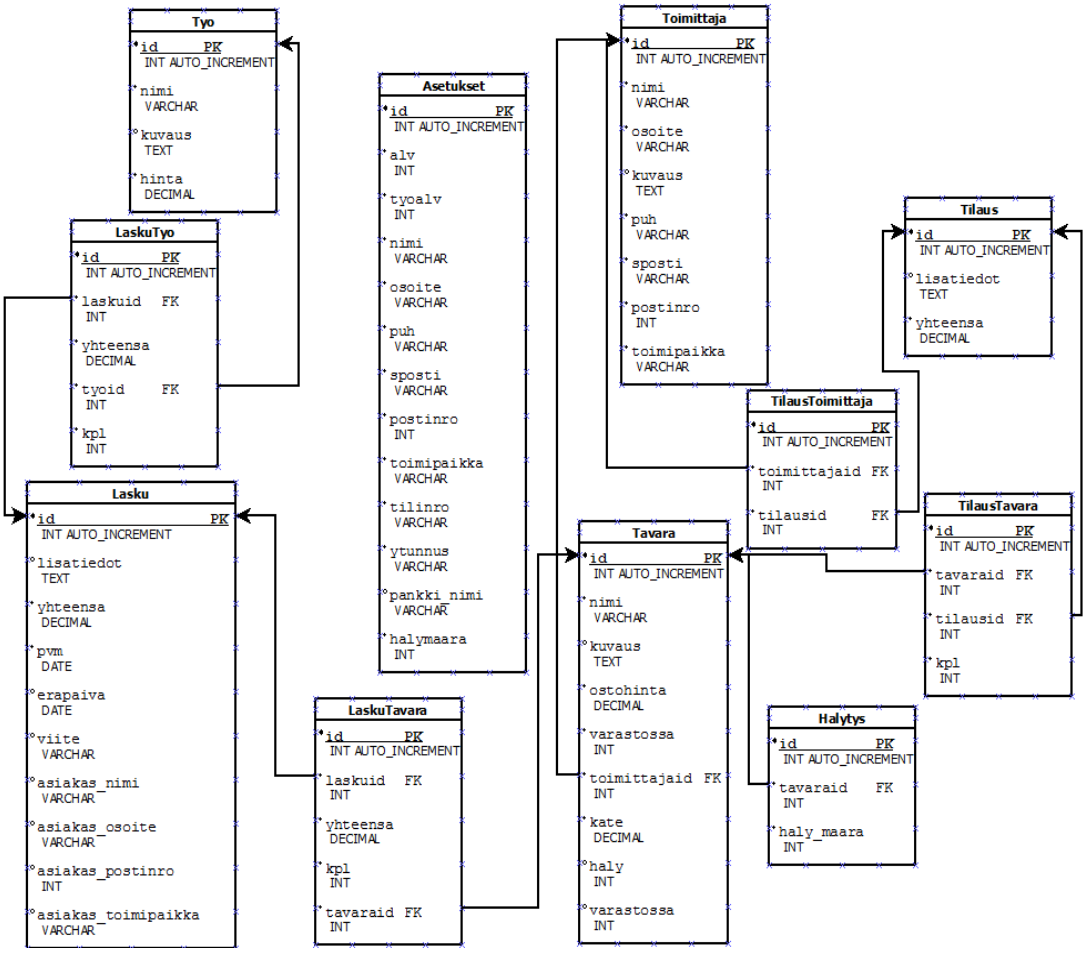
Suorittajat:	Käyttäjä
Esiehdot:	Järjestelmä on käynnissä, tietokanta on toiminnassa
Kuvaus:	Käyttäjä painaa ”muokkaa”-painiketta haluamansa tuotteen kohdalta ja kirjoittaa haluamansa hälytysmäärän ”Hälyttävä saldomäärä”-kenttään ja painaa ”Tallenna”-painiketta [Poikkeus 1: Kentässä kiellettyjä merkkejä].
Poikkeukset:	1. Kentässä kiellettyjä merkkejä: Käyttäjä on syöttänyt ”Hälyttävä saldomäärä”-kenttään kiellettyjä merkkejä. Käyttäjää pyydetään korjaamaan virheellinen syöte ja yrittämään uudelleen.
Jälkiehdot:	Käyttäjä on muuttanut tietyn tuotteen saldohälytystä.

LIITE 3. Tietokannan ER-Kaaviot

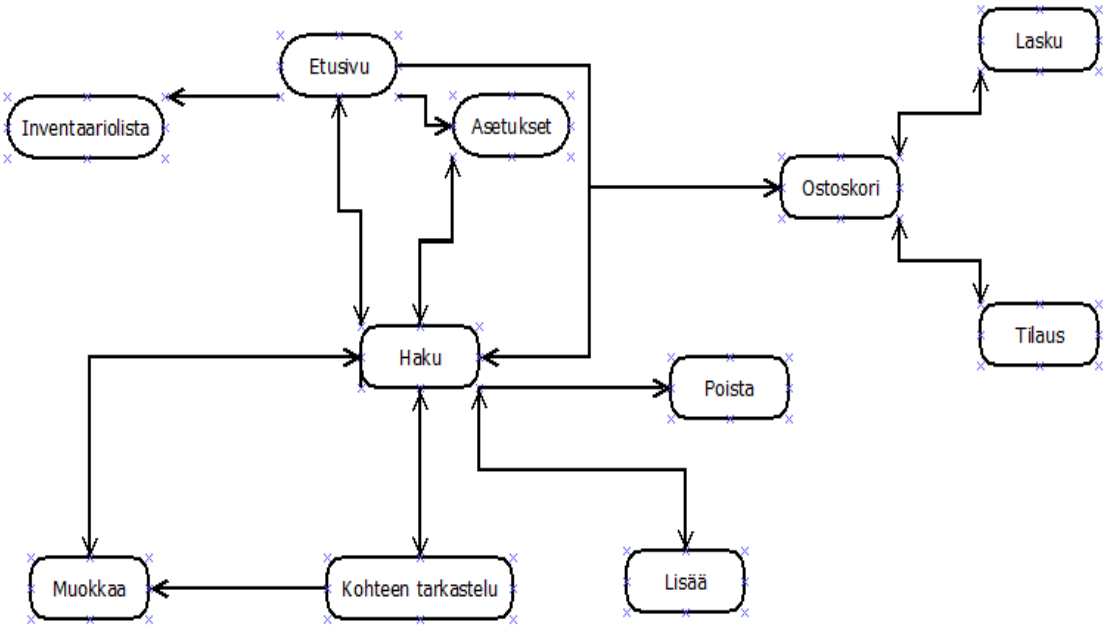




LIITE 4. Relaatiotietokanta



LIITE 5. Käyttöliittymädiagrammi



LIITE 6. Tietokannan luontilauseet

```
CREATE USER ‘X' IDENTIFIED BY 'X';
CREATE DATABASE IF NOT EXISTS korjaamo DEFAULT CHARACTER SET utf8 DEFAULT
COLLATE utf8_general_ci;
GRANT ALL PRIVILEGES ON korjaamo.* TO 'X'@'localhost' IDENTIFIED BY 'X' WITH
GRANT OPTION;

USE korjaamo;

DROP TABLE IF EXISTS laskutavara;
DROP TABLE IF EXISTS laskutyo;
DROP TABLE IF EXISTS halytys;
DROP TABLE IF EXISTS tilaustavara;
DROP TABLE IF EXISTS tilaustoimittaja;
DROP TABLE IF EXISTS asetukset;
DROP TABLE IF EXISTS lasku;
DROP TABLE IF EXISTS tavara;
DROP TABLE IF EXISTS tilaus;
DROP TABLE IF EXISTS toimittaja;
DROP TABLE IF EXISTS tyo;

CREATE TABLE asetukset (
id INT NOT NULL AUTO_INCREMENT,
alv DECIMAL(5,2) NOT NULL,
tyoalv DECIMAL(5,2) NOT NULL,
nimi VARCHAR(30) NOT NULL,
osoite VARCHAR(50) NOT NULL,
puh VARCHAR(30) NOT NULL,
sposti VARCHAR(50) NOT NULL,
```

```
postinro VARCHAR(5) NOT NULL,
toimipaikka VARCHAR(20) NOT NULL,
tilinro VARCHAR(30) NOT NULL,
ytunnus VARCHAR(15) NOT NULL,
pankki_nimi VARCHAR(30) NULL,
haly_maara INT NOT NULL,
PRIMARY KEY(id)
) ENGINE=innodb;

CREATE TABLE tyo (
id INT NOT NULL AUTO_INCREMENT,
nimi VARCHAR(30) NOT NULL,
kuvaus TEXT NULL,
hinta DECIMAL(7,2) NOT NULL,
PRIMARY KEY(id)
) ENGINE=innodb;

CREATE TABLE lasku (
id INT NOT NULL AUTO_INCREMENT,
asiakas_nimi VARCHAR(50) NULL,
asiakas_osoite VARCHAR(50) NULL,
asiakas_postinro VARCHAR(5) NULL,
asiakas_toimipaikka VARCHAR(20) NULL,
lisatiedot TEXT NULL,
yhteensa DECIMAL(7,2) NOT NULL,
pvm VARCHAR(10) NOT NULL,
erapaiva VARCHAR(10) NULL,
viite VARCHAR(20) NULL,
PRIMARY KEY (id)
) ENGINE=innodb;
```

```
CREATE TABLE toimittaja (
id INT NOT NULL AUTO_INCREMENT,
nimi VARCHAR(30) NOT NULL,
osoite VARCHAR(50) NOT NULL,
kuvaus TEXT NULL,
puh VARCHAR(15) NOT NULL,
sposti VARCHAR(50) NOT NULL,
postinro VARCHAR(5) NOT NULL,
toimipaikka VARCHAR(20) NOT NULL,
PRIMARY KEY (id)
) ENGINE=innodb;

CREATE TABLE tavara (
id INT NOT NULL AUTO_INCREMENT,
toimittajaid INT NOT NULL,
nimi VARCHAR(30) NOT NULL,
kuvaus TEXT NULL,
ostohinta DECIMAL(7,2) NOT NULL,
kate DECIMAL(7,2) NOT NULL,
varastossa INT NULL,
haly INT NULL,
FOREIGN KEY(toimittajaid) REFERENCES toimittaja(id)
ON UPDATE CASCADE ON DELETE CASCADE,
PRIMARY KEY (id)
) ENGINE=innodb;

CREATE TABLE tilaus  (
id INT NOT NULL AUTO_INCREMENT,
lisatiedot TEXT NULL,
yhteensa DECIMAL(7,2) NOT NULL,
```

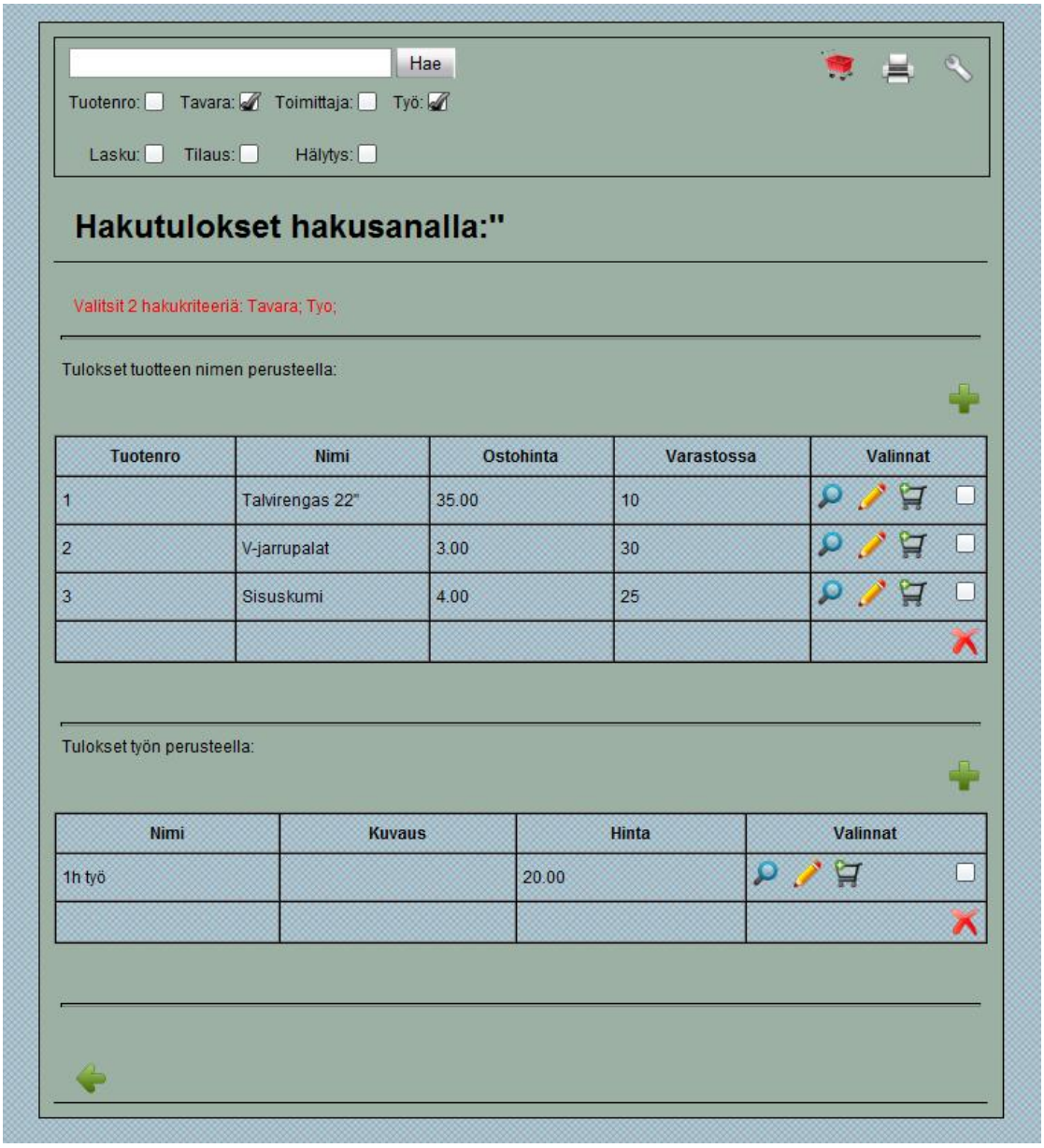
```
PRIMARY KEY(id)
) ENGINE=innodb;

CREATE TABLE laskutavara (
id INT NOT NULL AUTO_INCREMENT,
laskuid INT NOT NULL,
yhteensa DECIMAL(7,2) NOT NULL,
kpl INT NOT NULL,
tavaraid INT NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY(laskuid) REFERENCES lasku(id)
ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY(tavaraid) REFERENCES tavara(id)
ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=innodb;

CREATE TABLE laskutyo (
id INT NOT NULL AUTO_INCREMENT,
laskuid INT NOT NULL,
hinta DECIMAL(7,2) NOT NULL,
tyoid INT NOT NULL,
kpl INT NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY(laskuid) REFERENCES lasku(id)
ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY(tyoid) REFERENCES tyo(id)
ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=innodb;

CREATE TABLE halytys (
id INT NOT NULL AUTO_INCREMENT,
```

```
tavaraid INT NOT NULL,
halymaara INT NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY(tavaraid) REFERENCES tavara(id)
ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=innodb;

CREATE TABLE tilaustavara (
id INT NOT NULL AUTO_INCREMENT,
tavaraid INT NOT NULL,
tilausid INT NOT NULL,
kpl INT NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY (tavaraid) REFERENCES tavara(id)
ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY (tilausid) REFERENCES tilaus(id)
ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=innodb;

CREATE TABLE tilaustoimittaja (
id INT NOT NULL AUTO_INCREMENT,
toimittajaid INT NOT NULL,
tilausid INT NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY (tilausid) REFERENCES tilaus(id)
ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY (toimittajaid) REFERENCES toimittaja(id)
ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=innodb;
```

```
INSERT INTO
asetuk-
set(alv,tyoalv,nimi,osoite,puh,sposti,postinro,toimipaikka,tilinro,ytunnus,pankki_nim
i,haly_maara) VALUES ("X","X","X","X","X","X","X","X","X","X","X");
```

LIITE 7. Käyttöliittymän näkymät

Hakunäkymä (testidatalla)



Muokkausnäkymä (testidatalla)

Ostoskorinäkymä (testidatalla)