Metropolia Ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Sami Nieminen

Web-pohjainen kodin kirjanpitosovellus

Insinöörityö 1.6.2009

Ohjaaja: koulutuspäällikkö Markku Karhu

Metropolia Ammattikorkeakoulu **Insinöörityön tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Sami Nieminen Web-pohjainen kodin kirjanpitosovellus 49 sivua 1.6.2009
Koulutusohjelma	tietotekniikka
Tutkinto	insinööri (AMK)
Ohjaaja	koulutuspäällikkö Markku Karhu

Insinöörityössä oli tavoitteena tutustua Web-pohjaisen sovelluksen suunnitteluun ja toteutukseen käytännönläheisen projektin avulla. Työssä kehitettiin Web-pohjainen kodin kirjanpitosovellus.

Insinöörityö aloitettiin tekemällä sovelluksen määrittely, jossa kuvattiin sovelluksen toiminta käyttäjän näkökulmasta. Seuraavaksi suunniteltiin sovelluksen toteutus, jossa päätettiin muun muassa käytettävät tekniikat ja luokkarakenteet. Suunnittelun jälkeen ohjelmoitiin sovellus sekä suoritettiin testaukset.

Web-sovelluksen palvelinpuoli toteutettiin Java-ohjelmointikielellä, käyttäen MVC (Model-View-Controller)-arkkitehtuuria, jonka avulla ohjelmakoodista tuli selkeämpää ja hallittavampaa.

Asiakasliittymä toteutettiin ExtJSällä, joka on selainriippumaton JavaScript-kirjasto. Kirjaston avulla saatiin luotua hyvännäköinen interaktiivinen käyttöliittymä.

Tietokantakyselyt hoidettiin Hibernate ORM-kerroksen avulla. Tuomalla ylimääräinen kerros sovelluksen ja tietokannan väliin, saatiin aikaiseksi tietokantariippumattomuus sekä piilotettiin ohjelmoijalta tarvittavien tietokantayhteyksien luonti ja SQL-lausekkeet.

Insinöörityön lopputuloksena syntyi toimiva kodin kirjanpitosovellus, jolla on helppo seurata omia tuloja ja menoja.

Projektin kokemukset osoittivat määrittely- ja suunnitteluvaiheiden tärkeyden. Työn aikana opittiin, mitä asioita kannattaa ottaa huomioon eri vaiheissa sekä mihin kannattaa panostaa enemmän. Tärkein vaihe ohjelmistokehityksessä oli määrittely, koska sitä käytettiin koko kehityksen ajan.

Hakusanat	Java, ExtJS, Hibernate

Helsinki Metropolia University of Applied Sciences **Abstract**

Author	Sami Nieminen
Title	Web-based home accounting application
Number of Pages	49
Date	1 June 2009
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor	Markku Karhu, Director of Degree Programme

The aim of this thesis was to explore the design and implementation of a web-based application through a practical-oriented project. A Web-based home accounting application was the final outcome of the project.

The project was started by specifying the application's functionality from the user's perspective. The next step was to design the implementation of the application whereby a decision was made about the technologies and class structures to be used. The next step was the implementation and testing of the application.

The server-side of the Web application was implemented in the Java programming language using MVC (Model-View-Controller) architecture, which made the program code much easier to read and maintain. The client interface was implemented by using ExtJS, a cross-browser JavaScript library for building good-looking rich Internet applications.

Database queries were managed by the Hibernate ORM framework, which is an extra layer between application and database for gaining database independence and hiding the creation of database connection and SQL statements from the developer.

The project yielded an accounting application for easy income and expenditure follow-up. Experiences with the project demonstrated the importance of the specification and design phases. Specification turned out to be the most important phase of the development, as it was used throughout the process.

Keywords	Java, ExtJS, Hibernate

Sisällys

1 Johdanto

Tämän insinöörityön aiheena on tehdä Web-pohjainen kodin kirjanpitosovellus. Työn aihe valittiin omasta mielenkiinnosta tutustua Web-pohjaisten sovellusten kehittämiseen sekä omasta tarpeesta saada toimiva kodin kirjanpitosovellus.

Työn tavoitteena on tutustua Web-pohjaisen sovelluksen suunnitteluun ja toteutukseen käytännönläheisen projektin avulla. Työn kulku jaetaan ohjelmistotuotannon tärkeimpiin vaiheisiin eli määrittelyyn, suunnitteluun, toteutukseen ja testaukseen [1].

Kodin kirjanpitosovelluksen ideana on pystyä käsittelemään kirjanpitoon liittyviä tietoja helposti ja yksinkertaisesti. Ylläpidettäviä asioita ovat päivittäinen kirjanpito, tulot, menot, velat ja tilit.

2 Määrittely

Määrittelyvaiheessa määritellään sovellukselle asetettavat tavoitteet ja vaatimukset. Määrittelyvaiheessa kuvataan kaikki sovelluksen toteuttamat toiminnot, ottamatta kantaa itse toteutukseen millään tavalla. Määrittelyvaiheen tuloksena syntyy määrittelydokumentti, joka toimii pohjana suunnitteluvaiheelle. Määrittelyyn kuuluvat sovelluksen yleiskuvaus, vaatimukset, ohjelman tietosisältö, alustava käyttöliittymän suunnittelu sekä käyttötapaukset. [1;2;3.]

Määrittelydokumentti tehdään yleensä yhteistyössä asiakkaan kanssa, jolloin tehdään kaikki päätökset sovellukselle käyttäjän näkökulmasta. Määrittelydokumentti toimii myös eräänlaisena sopimuksena asiakkaan ja sovelluksen kehittäjän välillä. Ihannetapauksessa määrittelydokumentti on niin tarkka, ettei suunnittelu- ja toteutusvaiheessa tarvitse tehdä mitään päätöksiä sovelluksen ulkonäöstä tai toiminnallisuudesta.

Sovelluksen tekeminen aloitettiin luomalla määrittelydokumentti, joka on liitteessä 1. Määrittelydokumenttia käytettiin apuna jokaisessa sovelluskehityksen vaiheessa, joten se oli oleellinen osa hyvin onnistunutta projektia.

Yleiskuvaus

Yleiskuvauksessa kuvataan sovellukselle asetettavat vaatimukset yleisellä tasolla, eikä yleensä oteta kantaa siihen miten käyttäjä tekee toiminnot tai miltä sovellus näyttää. Yleiskuvaus tehdään yleensä helpottamaan sovelluksen määrittelyä, kun kuvataan yleisesti halutun sovelluksen ominaisuuksia. Yleiskuvauksesta on helpompi hahmottaa sovelluksen kokonaisuus kuin muista määrittelydokumentin vaiheista.

Määrittelyn ensimmäinen vaihe oli luoda yleiskuvaus toteutettavasta sovelluksesta, jossa kuvattiin sovelluksen toiminta sanallisesti. Yleiskuvaus toimi pohjana myöhemmin tehdyille määrittelyille, joissa tarkennettiin sovelluksen toimintaa ja ulkoasua.

Toiminnalliset vaatimukset

Toiminnallisiin vaatimuksiin poimittiin yleiskuvauksesta kaikki sovelluksen vaatimat toiminnallisuudet. Toiminnallisuuksista tehtiin taulukko, johon myös merkittiin toteutuksen priorisointi, joka oli heti toteutettava (1) tai jatkoprojektissa tehtävä (2). Jokaisesta vaatimuksesta tehtiin myös tarkempi kuvaus, jossa vaatimus kuvattiin selkokielisesti.

Tämän vaiheen tarkoituksena on poimia yleiskuvauksesta kaikki toiminnalliset vaatimukset sekä tarvittaessa niitä voidaan vielä tässä vaiheessa helposti lisätä. Näiden vaatimusten pohjalta voi helposti määritellä käyttötapaukset sekä suunnitteluvaiheessa nähdä tarvittava automatisointi.

Muut vaatimukset

Muihin vaatimuksiin poimittiin yleiskuvauksesta kaikki sovelluksen ei-toiminnalliset vaatimukset sekä vaatimuksista tehtiin tarkempi sanallinen kuvaus.

Tämän vaiheen tarkoituksena on kartoittaa sovellukselle asetettavat muut vaatimukset, kuten käyttöympäristö ja suorituskykyvaatimukset.

Tietosisältö

Tietosäiliöön määriteltiin kaikki sovelluksen käyttämät tietovarastot, eli mitä tietoja sovelluksen tuli säilyttää. Tässä vaiheessa myös määriteltiin, mitkä tiedot ovat pakollisia ja mitkä vapaaehtoisia. Määrittelyvaiheessa ei vielä otettu kantaa tietotyyppiin tai säilytystapaan. [3.]

Käyttötapausten kuvaukset

Käyttötapauksissa kuvattiin sovelluksen toiminnallisuus käyttäjän näkökulmasta, eli kuvattiin se, miten käyttäjä tekee toiminnon ja mitä palautetta käyttäjä saa, sekä virhetilanteet. Käyttötapauksissa määriteltiin kolme osaa: alkuehto, kuvaus ja poikkeukset. Alkuehdon pitää täyttyä, että käyttäjä voi suorittaa käyttötapauksen. Kuvauksessa esitettiin tapauksen kulku yksityiskohtaisesti alusta loppuun. Poikkeuksissa kuvattiin sellaisen poikkeuksellisen tapahtuman kulku, johon viitattiin kuvauksessa.

Käyttötapausten määrittely oli tärkeä vaihe, koska siinä kuvattiin toimintojen kulku. Käyttötapaukset toimivat myös toteutusvaiheessa apuna, sekä testaukset suoritettiin käyttötapausten pohjalta.

Käyttöliittymä

Käyttöliittymän määrittelyvaiheessa kuvattiin sovelluksen ulkoasu sekä lomakkeet ja painonapit. Tämän vaiheen tarkoituksena oli määritellä sovelluksen näkyvä osa, jolloin toteutusvaiheessa ei tarvinnut enää ottaa kantaa siihen miten asiat esitetään.

3 Suunnittelu

Määrittelydokumentin pohjalta luotiin seuraavaksi suunnitteludokumentti, joka ottaa enemmän kantaa siihen, miten asiat tehdään. Suunnitteluvaiheessa päätettiin käytettävät teknologiat, tietokantarakenne ja olioluokat. [1.]

Suunnitteludokumentin avulla toteutustyö voidaan jakaa monelle ohjelmoijalle. Suunnitteluvaiheessa päätetään muun muassa eri komponenttien väliset rajapinnat, jolloin komponentit voidaan liittää yhteen kun jokainen ohjelmoija on tehnyt osuutensa.

Suunnitteluvaiheen tarkoituksena oli kuvata, miten asiat tehdään, jolloin ei tarvitse enää toteutusvaiheessa päättää toteutustapaa. Suunnitteluvaiheen avulla saatiin ohjelmakoodille yhtenäinen rakenne sekä rajapinnat.

3.1 Teknologiat

Ensimmäiseksi päätettiin käytettävät teknologiat, jonka jälkeen oli mahdollista suunnitella muut asiat tarkemmin.

Java

Palvelinsovelluksen toteutus päätettiin tehdä Java-ohjelmointikielellä, jolloin palvelinsovelluksen voi ajaa missä tahansa ympäristössä. Kehitysympäristöksi valittiin J2EE (Java 2 Enterprise Edition), joka on suunniteltu palvelinympäristöille.

Syy Java-ohjelmointikielen valintaan oli halu syventää omaa osaamista Java-ohjelmoinnissa.

Hibernate

Tietokannat noudattavat yleensä relaatiomallia, jolloin olioiden sovittaminen tietokantaan on työlästä. ORM (Object-Relational Mapping) -kerroksen tarkoitus on piilottaa ohjelmoijalta tarvittavien tietokantayhteyksien luonti ja SQL-lausekkeet, joilla tieto haetaan. Tuomalla ylimääräinen kerros tietokannan ja sovelluksen väliin saadaan aikaan myös muita hyötyjä kuten kantariippumattomuus ja validointi. [4.]

Hibernate on tällä hetkellä suosituin ORM-ratkaisu Javalle [5]. Hibernate soveltuu parhaiten sovelluksiin, joissa SQL-lausekkeiden luonti jätetään ORM-kerrokselle sekä tietokanta voidaan luoda sovelluksen pohjalta. Toinen suosittu ORM-ratkaisu on iBatis [6]. iBatis vastaavasti soveltuu parhaiten sovelluksiin, joissa kehittäjän tarvitsee hallita SQL-lausekkeita, sekä käytetään jo olemassa olevaa tietokantaa. [7.]

Työhön valittiin Hibernate, koska sovellus määrää tietokantarakenteen sekä SQL-lausekkeiden luonti haluttiin jättää ORM-kerroksen vastuulle.

ExtJS

ExtJS on selainriippumaton JavaScript-kirjasto, jolla voidaan helposti luoda web-sovelluksen käyttöliittymä sekä toiminta asiakaspäähän.

ExtJS-kehyksen valintaan vaikutti sen kehityksen aktiivisuus, tuki, dokumentaatio sekä ohjelmakoodin helppo ymmärrettävyys ja tekemisen yksinkertaisuus [8].

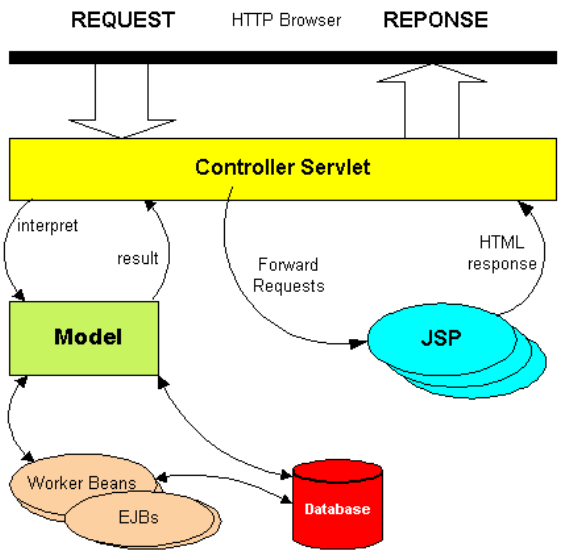
JSON

JSON (**J**ava**S**cript **O**bject **N**otation) on yksinkertainen tiedonsiirtomuoto, jota on helppo käyttää JavaScript-ohjelmissa. Nimestä huolimatta JSON on JavaScriptistä riippumaton, eli sitä voidaan käyttää muissakin ohjelmointikielissä. [9.]

JSON-tiedosiirtomuodon valintaan vaikutti ExtJS-kehyksen täysi JSON-tuki, luettavuus sekä keveys XML-formaattiin verrattuna.

3.2 Arkkitehtuuri

Palvelinsovelluksen rakenne päätettiin toteuttaa MVC (Model-View-Controller) -arkkitehtuurilla, jossa ohjelma jaetaan kolmeen osaan: malliin, näkymään ja ohjaimeen. Kuvassa 1 on esitetty MVC-arkkitehtuuri J2EE-sovelluksessa, jossa esitetään eri osien toiminta arkkitehtuurissa.



Kuva 1: MVC-arkkitehtuuri J2EE-sovelluksessa[10]

Ohjaimen (Controller) tehtävänä on vastaanottaa toimintapyynnöt, validoida syötteet sekä kutsua oikeata mallia pyynnön mukaan, jonka jälkeen ohjain valitsee oikean näkymän mallin paluuarvosta riippuen [10].

Mallin (Model) tehtävänä on huolehtia tiedon ylläpidosta sekä vastata ohjelman logiikasta [10].

Näkymän (View) tehtävänä on määrittää käyttöliittymän ulkoasu ja mallin tietojen esitysmuoto [10].

3.3 Luokkarakenne

Sovelluksen luokat jaettiin viiteen kerrokseen, mikä mahdollistaa helpomman ylläpidon. Jokaisella kerroksella on oma tehtävänsä, jolloin lisätoimintojen tuottaminen sovellukseen on yksinkertaisempi toteuttaa.

Controller

Controller eli ohjain on servlet-tyyppinen luokka, joka vastaanottaa http-pyynnöt ja ohjaa pyynnön eteenpäin. Pyynnön mukana tulee toimintapyyntö, jonka perusteella ohjain osaa ohjata pyynnön tietylle tapahtumaluokalle. [11;12.]

Sovellukselle tehtiin keskitetty hallinta, eli luotiin vain yksi ohjain, jolloin istunnon hallinta helpottui. Esimerkiksi pyynnön sisältäessä parametrin "action=insertKirjanpitoTapahtuma", ohjain ohjaa

pyynnön InsertKirjanpitoTapahtumaAction-luokalle.

Action

Action- eli tapahtumaluokan tarkoituksena on purkaa parametrit pyynnöstä ja välittää ne palveluluokalle sekä valita seuraava näkymä, jonka ohjain palauttaa. Jokaiselle käyttötapaukselle luotiin oma tapahtumaluokka, jossa on metodi "perform", jota ohjain kutsuu. [11;12.]

Esimerkiksi InsertKirjanpitoTapahtumaAction-luokka purkaa pyynnöstä päivämäärän, selitteen, tilin ja summan, jonka jälkeen se kutsuu palveluluokan KirjanpitoTapahtumaService metodia InsertKirjanpitoTapahtuma(pvm, selite, summa, tili).

Service

Service- eli palveluluokan tarkoituksena on hoitaa itse bisneslogiikka. Luokka muuttaa mallien tilaa, kutsuu DAO-luokkia sekä palauttaa ohjaimelle tarvittavat mallit, joita ohjain tarvitsee näkymän luontia varten. [12.]

Jokaiselle näkymälle luotiin oma palveluluokka eli tehtiin viisi palveluluokkaa: KirjanpitoTapahtumaService, TuloService, MenoService, VelkaService, TiliService.

Esimerkiksi uuden kirjanpitotapahtuman luonnissa kutsutaan KirjanpitoTapahtumaService-olion metodia insertKirjanpitoTapahtuma, jolloin luodaan ensin uusi KirjanpitoTapahtuma-olio, johon sijoitetaan annetut arvot. Tili-olio haetaan tietokannasta TiliDAOn avulla,

muutetaan tilin saldoa ja päivitetään olio tietokantaan TiliDAOn avulla. KirjanpitoTapahtuma-olio tallennetaan tietokantaan KirjanpitoTapahtumaDAO-luokan avulla.

DAO

DAO- (Data Access Object) eli tietokantarajapintaluokka huolehtii tietokantapyynnöistä, jolloin erotetaan tietokannan käsittely itse palveluluokasta. Tällä menettelyllä saatiin ohjelmakoodista helpommin ymmärrettävää ja ylläpidettävää. Jos tietokannan käsittelyä halutaan jostain syystä vaihtaa, niin ei tarvitse muuttaa kuin DAO-luokkia. [12.]

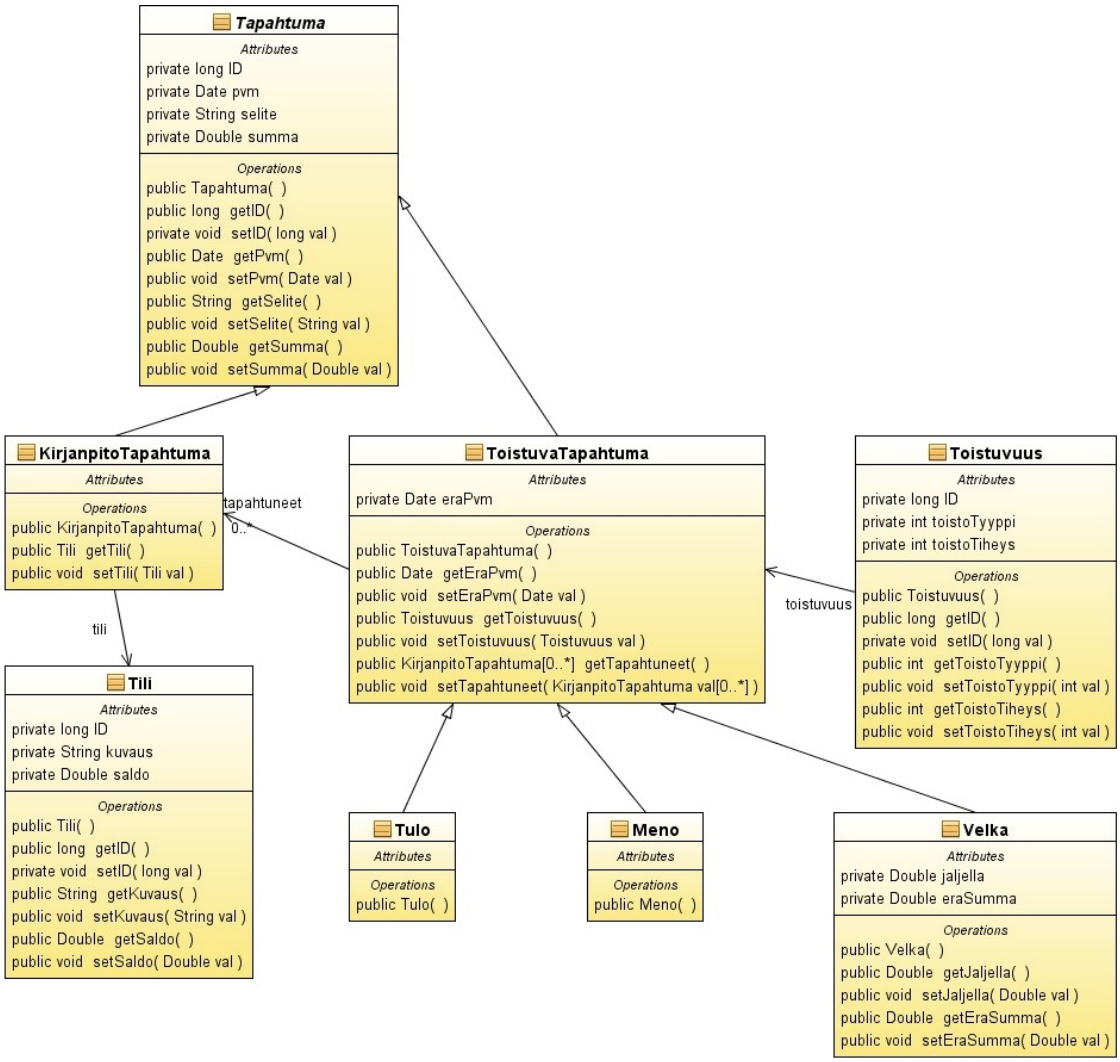
Jokaiselle lopulliselle luokalle tehtiin oma DAO-luokka eli sovellukseen tehtiin viisi DAO-luokkaa: KirjanpitoTapahtumaDAO, TuloDAO, MenoDAO, VelkaDAO, TiliDAO.

Esimerkiksi uuden kirjanpitotapahtuman luonnissa kutsutaan KirjanpitoTapahtumaDAO-olion metodia insert(KirjanpitoTapahtuma), jolloin metodi huolehtii Hibernaten tarvitsemista toiminnoista ja tallentaa olion tietokantaan Hibernaten välityksellä.

VO

VO- (Value Object) eli tietosäiliöluokat ovat POJO (Plain Old Java Object) -luokkia, eli ne sisältävät muuttujia sekä haku- ja asetus-metodit jokaiselle muuttujalle. Tietokannan sisältö esitetään ja muutetaan näiden luokkien avulla.

Kuvassa 2 on VO-luokkien luokkakaavio, joka muutettiin ORM-kerroksen avulla relaatiomalliin tietokantaan.



Kuva 2: VO-luokkien luokkakaavio

3.4 Kantarakenne ja rajapinnat

Kantarakenteen luonti jätettiin Hibernaten vastuulle, joka myös huolehtii tietotyyppien määrittelyistä VO-luokkien avulla. Hibernaten kartoitus (Mapping) -tiedostoon määriteltiin oliorakenne, josta Hibernate luo relaatiorakenteen.

Asiakas- ja palvelinpuolen välinen keskustelu hoidettiin Ajax-pyynnöillä, jossa asiakaspuoli lähettää pyynnön ja palvelinpuoli

vastaa. Asiakaspyynnöt ovat http-parametrejä ja vastaus on JSON-muotoista tietoa.

Pyyntö sisältää action-parametrin, jolla määritellään, mikä action-luokka käsittelee pyynnön, jonka lisäksi pyyntö sisältää tapahtuman tarvitsemat parametrit. Jos action-parametri puuttuu, palautetaan aloitussivu.

Vastauksena tuleva JSON-objekti sisältää parametrin success, jolla kerrotaan JavaScript-kehykselle, onnistuiko toimintapyynnön suorittaminen palvelinpuolella.

4 Toteutus

Toteutusvaiheessa tehdään itse sovelluksen toteutusosuus, joka on määritelty ja suunniteltu tähän mennessä. Hyvin suunnitellun kehitysprojektin toteutusvaihe pitäisi olla yksinkertaisin osuus ohjelmistokehityksessä.

Kehitysympäristönä oli käytössä NetBeans 6.5, johon lisättiin tuki Hibernatelle. Ajoympäristönä toimi Apache Tomcat 6.0 ja Java JDK 6. Tietokantapalvelimena toimi MySQL-palvelin, johon luotiin kirjanpito-niminen tietokanta.

Ensimmäiseksi NetBeansissä luotiin uusi Web-sovellusprojekti, johon lisättiin JSON- ja Hibernate-kirjastot.

4.1 Palvelinpuoli

Ohjelmointiosuus aloitettiin tekemällä VO-luokat, koska ne ovat muista riippumattomia luokkia. Luokat toteutettiin täysin samalla tavalla kuin luokkakaaviossa oli suunniteltu.

Seuraavaksi luotiin HibernateUtil-luokka, jonka tehtävänä on alustaa Hibernaten SessionFactory-olio [13]. HibernateUtil-luokka luotiin NetBeansin omalla toiminnolla lisäämällä projektiin uusi HibernateUtil.class-tiedosto. NetBeansin luomassa luokassa oli pieni virhe SessionFactory-olion alustuksessa, jossa rivi

```
sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();
```

piti muuttaa muotoon

```
sessionFactory = new Configuration().configure().buildSessionFactory();
```

Seuraavaksi tehtiin DAO-luokat, joilla hallitaan Hibernaten toimintaa. Hibernaten tehtävänä on tietokantakyselyiden luominen, jolloin kaikki DAO-luokat ovat samanlaisia. Niinpä luotiin geneerinen DAO-luokka, josta kaikki DAO-luokat periytyvät, eikä tarvitse kirjoittaa samoja komentoja moneen kertaan, mutta tietokantakäsittelijä voidaan tarvittaessa vaihtaa. [14.]

Seuraavana ohjelmoitiin palveluluokat, jotka toteuttavat itse sovelluksen bisneslogiikan. Jokaista käyttötapausta varten luotiin oma metodi siihen näkymän luokkaan, johon käyttötapaus liittyy. Jokainen metodi ohjelmoitiin toimimaan käyttötapauksissa määritellyllä tavalla.

Palveluluokkien toteutuksen jälkeen ohjelmoitiin action-luokat, joiden tarkoituksena on purkaa pyynnöstä parametrit ja kutsua palveluolion metodia sekä palauttaa ohjaimelle seuraava näkymä. Jokaiselle käyttötapaukselle luotiin oma action-luokka, jotta ylläpito helpottuisi.

Seuraavaksi ohjelmoitiin FrontController, joka vastaa tulevien pyyntöjen välityksestä action-luokille. Alustusosassa jokaisesta action-luokasta luodaan olio ja sijoitetaan ne HashMap-taulukkoon, josta ohjain löytää käyttäjän pyytämän action-olion helposti. Ohjain siis purkaa pyynnöstä action-parametrin, jonka perusteella se valitsee tapahtumaolion ja kutsuu sen perform-metodia. Metodi palauttaa seuraavan näkymän, johon ohjain siirtää pyynnön. [11.]

Viimeisenä palvelinpuolen toteutuksena tehtiin JSP (JavaServer Pages) -sivut, joiden avulla mallit muutetaan esitettävään muotoon. Ohjain-

luokka siirtää pyynnöt JSP-sivulle, jonka se valitsee pyynnön toteutumisesta riippuen. Tässä projektissa luotiin kahdeksan JSP-sivua, jotka ovat:

- index.jsp, jolla palautetaan käyttäjälle alustussivu, joka lataa JavaScript-tiedostot ja käynnistää sovelluksen asiakaspuolen
- error.jsp, jolla palautetaan JSON-objekti, joka kertoo käyttöliittymälle pyynnön epäonnistuneen
- success.jsp, jolla palautetaan JSON-objekti, joka kertoo käyttöliittymälle pyynnön onnistuneen
- viisi list*.jsp-sivua, joilla näkymän malli muunnetaan JSON-muotoon käyttöliittymää varten.

4.2 Hibernate-asetukset

Hibernaten asetukset määriteltiin XML-tiedostoon, jolle annettiin nimeksi hibernate.cfg.xml. Tiedostoon määriteltiin käytettävä protokolla, ajuri, yhteysosoite, käyttäjätunnus ja salasana, Hibernaten oikeus päivittää tietokanta sekä olioiden kartoitustiedosto. Esimerkissä 1 on kuvattu sovelluksessa käytetty asetustiedosto.

Esimerkki 1: hibernate.cfg.xml-tiedosto

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://10.10.15.2/kirjanpito</property>
    <property name="hibernate.connection.username">kirjanpito</property>
    <property name="hibernate.connection.password">kirjanpito</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <mapping resource="hibernate.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Seuraavaksi määriteltiin kartoitustiedosto, johon määriteltiin käytettävä luokkarakenne. Tiedostoon kuvattiin suoraan luokkakaavion tiedot, jolloin Hibernate osaa käsitellä kaikkia tietoja ja luoda tarvittavan tietokantarakenteen. Esimerkissä 2 on kartoitustiedosto, johon on kuvattu sovelluksen VO-luokkakaavio. Aliluokkien liittäminen pääluokkaan toteutettiin joined-subclass-määrittelyllä, joka luo jokaisesta luokasta oman taulun ja yhdistää avaimella ne toisiinsa. Syy tämän toteutustavan valintaan oli halu periytymisen säilyttämiseen ja automaattisten ID-tunnuksen luontien mahdollisuus. [15.]

Esimerkki 2: Hibernaten kartoitus-tiedosto

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://
hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="model.Tapahtuma" abstract="true">
    <id name="ID">
      <generator class="native"/>
    </id>

    <property name="pvm" />
    <property name="selite" />
    <property name="summa" />
  </class>

  <joined-subclass name="model.KirjanpitoTapahtuma" extends="model.Tapahtuma">
    <key column="ID" />
    <many-to-one class="model.Tili" name="tili"/>
  </joined-subclass>

  <joined-subclass name="model.ToistuvaTapahtuma" extends="model.Tapahtuma"
abstract="true">
    <key column="ID" />
    <property name="eraPvm" />
    <component name="toistuvuus">
      <property name="toistoTiheys" />
    </component>
  </joined-subclass>

  <joined-subclass name="model.Meno" extends="model.ToistuvaTapahtuma">
    <key column="ID" />
  </joined-subclass>

  <joined-subclass name="model.Tulo" extends="model.ToistuvaTapahtuma">
    <key column="ID" />
  </joined-subclass>

  <joined-subclass name="model.Velka" extends="model.ToistuvaTapahtuma">
    <key column="ID" />
    <property name="eraSumma" />
    <property name="jaljella" />
  </joined-subclass>

  <class name="model.Tili">
    <id name="ID">
      <generator class="native"/>
    </id>

    <property name="kuvaus" />
    <property name="saldo" />
  </class>
</hibernate-mapping>
```

4.3 Asiakaspuoli

Ensimmäiseksi asiakaspuolen käyttöliittymää varten lisättiin ExtJS-kirjasto web-hakemistoon, josta selain saa ladattua ne toimiakseen. Koko ExtJS-kirjaston toiminta perustuu olioihin, joille annettiin luontivaiheessa parametrejä, joiden mukaan ne toimivat.

Asiakasliittymää varten luotiin jokaiselle näkymälle oma JavaScript-tiedosto sekä application.js, joka hoitaa käyttöliittymän alustuksen. ExtJS:n käyttöönotto tapahtui luomalla uusi Ext.Viewport-olio, johon määriteltiin käyttöliittymän ulkoasu sekä siihen liittyvät komponentin. Käyttöliittymän ulkoasu tehtiin määrittelyvaiheen ulkoasun mukaiseksi. Viewport-olioon asetettiin valikko, joka sisältää jokaista näkymää varten painikkeet. Painikkeisiin asetettiin kuuntelijat kutsumaan näkymän alustusta käyttäjän painaessa nappia.

Kirjanpidon JavaScript-tiedostoon tehtiin kirjanpito-luokka, jossa on julkinen Init-metodi, jota valikon kuuntelija kutsuu. Lisäksi kirjanpito-luokkaan tehtiin yksityiset metodit Render, CreateForm sekä DeleteItem. Render-metodi luo uuden taulukko-olion ja sijoittaa siihen tarvittavat asetukset ja komponentit, jonka jälkeen taulukko-olio hoitaa käyttöliittymän bisneslogiikan. CreateForm-metodi luo uuden lomakeikkunan, jota käyttäjä käyttää kirjanpitotapahtuman luomiseen tai muokkaamiseen. DeleteItem-metodilla lähetetään palvelimelle pyyntö poistaa kirjapitotapahtuma.

Seuraavaksi tehtiin muutkin JavaScript-luokat samalla periaatteella kuin kirjanpitoluokkakin. Koko käyttöliittymän toimintalogiikka tehtiin noudattamaan käyttötapauksissa kuvattua kulkua. Käyttöliittymän toteutuksessa käytettiin apuna ExtJS:n API-dokumentaatiota, joka on erittäin kattava sekä selkeä.

Viimeisenä tehtiin asennus- ja käyttöohje, joka löytyy liitteestä 2.

5 Testaus

Ohjelmakoodin testausta varten luotiin JUnit-testiluokkia, joilla testattiin palvelinpuolen malli-luokkia. JUnit on yksinkertainen kehys toistettavien yksikkötestien tekemiseen Java-luokille [16].

VO-luokkia varten testausrutiineja ei tehty niiden yksinkertaisen rakenteen vuoksi. Niiden oikeellisuuden NetBeans-kehitysympäristö tarkistaa automaattisesti.

DAO-luokkia varten luotiin JUnit-testiluokkia, jotka testaavat DAO-luokkien toiminnallisuuden. Jokaisen DAO-luokan kaikkien metodien toiminta testattiin JUnit-luokilla, jolloin saatiin virheelliset tietokantakäsittelyt tietoon helposti.

DAO-luokkien JUnit-testien avulla löytyi virhe kirjanpitotapahtumien listauksessa, jos tapahtuma oli kohdistettu johonkin tiliin. Virhe johtui Hibernaten tavasta noutaa tietoja tietokannasta VO-olioille. Jokaiselle luokalle ja muuttujalle piti asettaa parametri "lazy=false" kartoitus-tiedostoon.

Myös palveluluokille tehtiin JUnit-testit, joilla testataan palveluluokan toimivuus. Korkeamman tason (action ja ohjain) -luokille ei enää tehty JUnit-testejä, vaan niiden toiminta testattiin kokeilemalla ohjelman toimintoja.

Lopullinen testaus suoritettiin käyttämällä ohjelmaa sekä tarkistamalla, että ohjelma suoriutuu kaikista sille asetetuista toiminnallisista vaatimuksista.

6 Yhteenveto

Insinöörityön tavoitteena oli tutustua Web-sovelluksen suunnitteluun ja toteutukseen käytännönläheisen projektin avulla. Työssä kehitettiin kodin kirjanpitosovellus, jolla käsitellään kodin kirjanpidossa tarvittavia tietoja. Työlle asetetut tavoitteet täyttyivät ja tuloksena oli toimiva sovellus. Työssä vei eniten aikaa määrittely- ja suunnitteluosuus, jonka jälkeen itse sovelluksen ohjelmointi oli pienin osuus.

Työssä opittiin, mitä Web-sovelluksen kehittämisessä kannattaa ottaa huomioon määrittely- ja suunnitteluvaiheessa. Koko ohjelmistokehityksen elinkaaren kannalta tärkein osuus oli hyvin tehdyllä määrittelyllä, jota käytettiin kaikissa muissa osuuksissa. Myös hyvällä suunnittelulla saatiin aikaiseksi se, että jatkokehitys ja ylläpito helpottuivat huomattavasti luokkajakojen avulla.

Työn toteutusvaiheen aikana jouduttiin monta kertaa tekemään määrittelyä ja suunnittelua, jolloin opittiin mitä kannattaa ottaa huomioon näissä vaiheissa. Ohjelmointityön ulkoistaminen ei olisi onnistunut työssä tehdyillä määrittelyillä, johtuen sen vaatimista jatkuvista täydentämisistä.

Hibernaten ORM-kerroksen avulla saatiin ylimääräinen kerros sovelluksen ja tietokannan väliin, joka osoittautui erittäin käytännölliseksi, ja tietokanta voitiin käsitellä oliona. ExtJS-kirjaston avulla saatiin luotua sovellukselle hyvännäköinen ulkoasu ja hyvä käytettävyys.

Tulevaisuudessa sovellukseen tullaan tekemään uusia ominaisuuksia sekä parantelemaan vanhoja toiminnallisuuksia. Sovellukseen tullaan lisäämään muun muassa monikielisyyden tuki, joka olisi kannattanut

ottaa huomioon jo sovelluksen suunnittelussa, koska sen lisääminen valmiiseen sovellukseen on työläämpää. Erilaisten raportointi-mahdollisuuksien tekeminen ja skannattujen kuittien tallentaminen tietokantaan ovat seuraavia kehittämiskohteita.

Lähteet

1 Ohjelmistotuotanto. (WWW-dokumentti.) Wikipedia. <http://fi.wikipedia.org/wiki/Ohjelmistotuotanto>. 6.3.2009. Luettu 20.3.2009.

2 Tuikka, Tommi. Internet-projekti. (WWW-dokumentti.) <http://sinuhe.jypoly.fi/~tuito/i_opinnot/>. Luettu 20.3.2009.

3 Nenonen , Jaakko. Määrittelydokumentti. (WWW-dokumentti.) <http://www.cs.helsinki.fi/u/jnenonen/alabra/maardoc.html>. 24.7.2006. Luettu 20.3.2009.

4 Pekkala, Lauri. Pysyvyyden toteuttaminen Java-sovelluksissa erityisesti ORM-menetelmän avulla . (WWW-dokumentti.) <http://www.cs.uta.fi/research/thesis/masters/Pekkala_Lauri.pdf>. 2004 . Luettu 20.3.2009.

5 Open Source Persistence Frameworks in Java . (WWW-dokumentti.) Java-Source.net. <http://java-source.net/open-source/persistence> Luettu 14.5.2009.

6 S. Sangeetha. iBATIS, Hibernate, and JPA: Which is right for you? (WWW-dokumentti.) <http://www.javaworld.com/javaworld/jw-07-2008/jw-07-orm-comparison.html>. 15.7.2008. Luettu 14.5.2009.

7 Object Relational Mapping Tools: Hibernate vs iBATIS . (WWW-dokumentti.) Software Wikipedia. <http://ormtools.blogspot.com/search/label/Hibernate%20vs%20iBATIS>. 5.2.2008. Luettu 14.5.2009.

8 JavaScript UI framework comparison or Why I choose ExtJS & JQuery. (WWW-dokumentti.) The Fictional Realm . <http://fictionalrealm.com/2009/04/javascript-ui-framework-comparison-or-why-i-choose-extjs-jquery/>. 19.4.2009. Luettu 14.5.2009.

9 JSON. (WWW-dokumentti.) JSON.<http://www.json.org/>. Luettu 14.5.2009.

10 Lloyd, David. What is MVC? (WWW-dokumentti.) <http://www.jcorporate.com/expresso/doc/edg/edg_WhatIsMVC.html>. 2004. Luettu 20.3.2009.

11 Designing Enterprise Applications with the J2EE Platform. (WWW-dokumentti.) Sun Microsystems. <http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html>. 2002. Luettu 20.3.2009.

12 Designing the Application . (WWW-dokumentti.) Oracle. <http://download.oracle.com/docs/cd/B15904_01/core.1012/b14000/design.htm>. 2004. Luettu 20.3.2009.

13 Introdution to Hibernate. (WWW-dokumentti.) Red Hat Middleware. <http://docs.jboss.org/hibernate/stable/core/reference/en/html/tutorial-firstapp.html>. 2004. Luettu 20.3.2009.

14 Generic DAO example. (WWW-dokumentti.) BeJUG. <http://www.bejug.org/confluenceBeJUG/display/BeJUG/Generic+DAO+example>. 20.3.2009. Luettu 30.3.2009.

15 Basic O/R Mapping . (WWW-dokumentti.) Red Hat Middleware. <http://docs.jboss.org/hibernate/stable/core/reference/en/html/mapping.html>. 2004. Luettu 20.3.2009.

16 JUnit. (WWW-dokumentti.) <http://junit.sourceforge.net/>. Luettu 15.5.2009.

Yleiskuvaus

Sovelluksen tulee olla yksinkertainen ja helppokäyttöinen, sekä sovelluksen käyttöönoton tulee olla yksinkertainen ja lyhyt operaatio. Sovelluksella tulee pystyä käsittelemään kodin kirjanpidossa tarvittavia tietoja, joita ovat kirjanpito, tulot, menot, velat ja tilit.

Kirjanpitoon kirjataan kaikki tulot ja menot, joita pitää pystyä tarkastelemaan taulukon muodossa sekä lisäämään, muokkaamaan ja poistamaan. Näytettävien tapahtumien määrä pitää voida rajoittaa erilaisilla hakukriteereillä, joita voi olla esimerkiksi tietty ajanjakso tai tapahtuman selite. Jokainen tapahtuma pitää voida kohdistaa johonkin tiliin, jolloin tapahtuma muuttaa tilin saldoa kyseisellä summalla.

Tuloihin kirjataan tulevat tulot, kuten palkat, joita pitää pystyä tarkastelemaan taulukon muodossa sekä lisäämään, muokkaamaan ja poistamaan. Tuloihin pitää saada asetettua toistomahdollisuus, jolloin tulotapahtuman hyväksymisen jälkeen tulolle asetetaan automaattisesti uusi eräpäivä. Tuloissa oleva tieto ei siirry automaattisesti kirjanpitoon eräpäivänä, vaan se pitää erityisesti hyväksyä, jolloin on mahdollisuus muuttaa kirjanpitoon kirjoitettavia tietoja tapahtumasta.

Menoihin kirjataan tulevat menot, kuten laskut, joita pitää pystyä tarkastelemaan taulukon muodossa sekä lisäämään, muokkaamaan ja poistamaan. Menoihin pitää saada asetettua toistomahdollisuus, jolloin menotapahtuman hyväksymisen jälkeen menolle asetetaan automaattisesti uusi eräpäivä. Menoissa oleva tieto ei siirry automaattisesti kirjanpitoon asetettuna eräpäivänä, vaan se pitää erityisesti hyväksyä, jolloin on mahdollisuus muuttaa kirjanpitoon kirjoitettavia tietoja tapahtumasta.

Velkoihin kirjataan kaikki otetut ja annetut velat, joita pitää pystyä tarkastelemaan taulukon muodossa sekä lisäämään, muokkaamaan ja poistamaan. Velan takaisinmaksaminen pitää olla mahdollista suorittaa kokonaan tai osittain, jolloin säilytetään tieto kokonaisvelasta sekä jäljellä olevasta velasta. Velalle pitää myös voida asettaa maksuväli, jolloin siihen päivitetään automaattisesti eräpäivä maksun hyväksymisen yhteydessä. Velan maksutapahtuma ei siirry automaattisesti kirjanpitoon, vaan se pitää erityisesti hyväksyä maksetuksi, jolloin on mahdollisuus muuttaa kirjanpitoon kirjoitettavia tietoja tapahtumasta sekä mahdollisuus muuttaa maksutapahtuman summaa.

Tileihin kirjataan kaikki tilit, joita pitää pystyä tarkastelemaan taulukon muodossa sekä lisäämään, muokkaamaan ja poistamaan.

Toiminnalliset vaatimukset

Sovelluksen vaatimat toiminnallisuudet on luetteloitu taulukossa 1, johon on myös merkitty toteutuksen prioriteetti. Prioriteettitasot ovat heti toteutettava (1) tai jatkoprojektissa tehtävä (2).

Taulukko 1: Toiminnalliset vaatimukset

Tunnus	Vaatimus	Prioriteetti
TV1	Kirjanpitotapahtumien tarkastelu	1
TV2	Kirjanpitotapahtumien hallinta	1
TV3	Kirjanpitotapahtumien rajoittaminen näkymässä	2
TV4	Kirjanpitotapahtumien kohdistaminen tiliin	1
TV5	Tulojen tarkastelu	1
TV6	Tulojen hallinta	1
TV7	Tulojen hyväksyntä	1
TV8	Tapahtumien toisto	1
TV9	Menojen tarkastelu	1
TV10	Menojen hallinta	1
TV11	Menojen hyväksyntä	1
TV12	Velkojen tarkastelu	1
TV13	Velkojen hallinta	1
TV14	Velkojen maksaminen	1
TV15	Tilien tarkastelu	1
TV16	Tilien hallinta	1

TV1 *Kirjanpitotapahtumien tarkastelu*
Käyttäjän tulee voida tarkastella kirjanpitotapahtumia taulukon muodossa.

TV2 *Kirjanpitotapahtumien hallinta*
Käyttäjän tulee voida lisätä, muokata sekä poistaa kirjanpitotapahtumia.

TV3 *Kirjanpitotapahtumien rajoittaminen näkymässä*
Käyttäjän tulee voida asettaa hakukriteereitä, jotka rajoittavat näkyvien kirjanpitotapahtumien määrää taulukossa.

TV4 *Kirjanpitotapahtumien kohdistaminen tiliin*
Käyttäjän tulee voida kohdistaa kirjanpitotapahtuma johonkin tiliin, jolloin tilin saldoa muutetaan automaattisesti kirjanpitotapahtuman summan määrällä.

TV5 *Tulojen tarkastelu*
Käyttäjän tulee voida tarkastella tuloja taulukon muodossa.

TV6 *Tulojen hallinta*
Käyttäjän tulee voida lisätä, muokata sekä poistaa tuloja.

TV7 *Tulojen hyväksyntä*
Käyttäjän tulee voida hyväksyä tulo maksetuksi, jolloin tulosta luodaan uusi kirjanpitotapahtuma.

TV8 *Tapahtumien toisto*
Tulo-, meno- ja velkatapahtumille tulee voida asettaa toistuvuusväli, jolloin tapahtuman hyväksynnän yhteydessä sille asetetaan automaattisesti uusi eräpäivä. Toistuvuusväli ilmoitetaan päivinä.

TV9 *Menojen tarkastelu*
Käyttäjän tulee voida tarkastella menoja taulukon muodossa.

TV10 *Menojen hallinta*
Käyttäjän tulee voida lisätä, muokata sekä poistaa menoja.

TV11 *Menojen hyväksyntä*
Käyttäjän tulee voida hyväksyä meno maksetuksi, jolloin menosta luodaan uusi kirjanpitotapahtuma.

TV12 *Velkojen tarkastelu*
Käyttäjän tulee voida tarkastella velkoja taulukon muodossa.

TV13 *Velkojen hallinta*
Käyttäjän tulee voida lisätä, muokata sekä poistaa velkoja.

TV14 *Velkojen maksaminen*
Käyttäjän tulee voida hyväksyä velka tai sen osa maksetuksi, jolloin jäljellä olevaa summaa muutetaan maksetun summan verran sekä tapahtumasta luodaan uusi kirjanpitotapahtuma.

TV15 *Tilien tarkastelu*
Käyttäjän tulee voida tarkastella tilejä taulukon muodossa.

TV16 *Tilien hallinta*
Käyttäjän tulee voida lisätä, muokata sekä poistaa tilejä.

Muut vaatimukset

MV1 *Käyttöjärjestelmästä riippumaton*
Sovelluksen tulee olla käyttöjärjestelmästä riippumaton, jolloin se toimii niin Microsoft Windows- kuin Linux-alustallakin.

MV2 *Käyttöönoton helppous*
Sovelluksen käyttöönoton tulee olla helppoa ja yksinkertaista, jolloin käyttäjän ei tarvitse tuhlata aikaa järjestelmän konfigurointiin.

MV3 *Helppokäyttöisyys*
Sovelluksen käyttämisen tulee olla mahdollisimman suoraviivaista ja helppoa, sekä käyttöliittymän tulee olla yksinkertainen.

Tietosisältö

Kirjanpitotapahtuman tulee sisältää tiedot tapahtuman

päivämäärästä, selitteestä, summasta sekä tapahtumaan kohdistetusta tilistä. Ainoastaan tapahtumaan kohdistettu tili on vapaaehtoinen tieto, muut ovat pakollisia tapahtumatietoja.

Tulon tulee sisältää tiedot tulon eräpäivästä, selitteestä, summasta sekä toistuvuusvälistä, joka on tulon ainoa vapaaehtoinen tieto.

Menon tulee sisältää tiedot menon eräpäivästä, selitteestä, summasta sekä toistuvuusvälistä, joka on menon ainoa vapaaehtoinen tieto.

Velan tulee sisältää tiedot velan ottopäivämäärästä, selitteestä, velan alkuperäisestä määrästä, jäljellä olevasta summasta, eräpäivästä, seuraavan erän määrästä sekä toistuvuusvälistä. Vapaaehtoisia tietoja ovat eräpäivä, erän määrä sekä toistuvuusväli.

Tilin tulee sisältää tiedot tilin kuvauksesta sekä saldosta, jotka ovat molemmat pakollisia tietoja.

Käyttötapausten kuvaukset

KT1 *Kirjanpitotapahtumien listaus*

Kuvaus
Käyttäjä napsauttaa valikosta kirjanpito-valintaa, jolloin avautuu taulukko kirjanpitotapahtumista.

KT2 *Kirjanpitotapahtuman lisäys*

Alkuehdot
Käyttäjä on kirjanpitotapahtumien listauksessa.

Kuvaus

Käyttäjä napsauttaa lisää-nappia työkaluriviltä, jolloin avautuu Kirjanpitotapahtuman lisäys -lomake. Käyttäjä täyttää lomakkeeseen tiedot ja napsauttaa lisää-nappia (Poikkeus: 1, 2). Kirjanpitotapahtuma tallennetaan järjestelmään, kirjanpitotapahtumaan kohdistetun tilin saldoa muutetaan (Poikkeus: 3) ja kirjanpitotapahtumien listaus päivitetään.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa tapahtumien listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.
3. Käyttäjä ei valinnut tapahtumaan kohdistettua tiliä, jolloin tilin saldoa ei muuteta.

KT3 *Kirjanpitotapahtuman tietojen muokkaus*

Alkuehdot

Käyttäjä on kirjanpitotapahtumien listauksessa.

Kuvaus

Käyttäjä valitsee muokattavan kirjanpitotapahtuman rivin, jolloin muokkaa-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa muokkaa-nappia, jolloin avautuu Kirjanpitotapahtuman muokkaus -lomake, jossa on muokattavan tapahtuman tiedot valmiina. Käyttäjä tekee haluamansa muutokset ja napsauttaa tallenna-nappia (Poikkeus: 1, 2). Kirjanpitotapahtuman summa poistetaan siihen kohdistetun tilin saldosta (Poikkeus: 3), muutokset tallennetaan järjestelmään, kirjanpitotapahtumaan kohdistetun tilin saldoa muutetaan (Poikkeus: 4) ja kirjanpitotapahtumien listaus päivitetään.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa tapahtumien listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.
3. Muokattava kirjanpitotapahtuma ei ole kohdistettu tiliin, jolloin tilin saldoa ei muuteta.
4. Käyttäjä ei valinnut tapahtumaan kohdistettua tiliä, jolloin tilin saldoa ei muuteta.

KT4 *Kirjanpitotapahtuman poisto*

Alkuehdot

Käyttäjä on kirjanpitotapahtumien listauksessa.

Kuvaus

Käyttäjä valitsee poistettavan kirjanpitotapahtuman rivin, jolloin poista-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa hiirellä poista-nappia, jolloin käyttäjältä pyydetään vahvistus tapahtuman poistoon. Käyttäjä hyväksyy tapahtuman poiston (Poikkeus: 1). Tapahtuma poistetaan järjestelmästä, tapahtuman summa poistetaan siihen kohdistetun tilin saldosta (Poikkeus: 2) ja kirjanpitotapahtumien listaus päivitetään.

Poikkeukset

1. Käyttäjä ei hyväksy poistoa, jolloin käyttäjä palaa tapahtumien listaukseen.
2. Tapahtumaa ei ole kohdistettu mihinkään tiliin, jolloin tilin saldoa ei muuteta.

KT5 *Tulojen listaus*

Kuvaus
Käyttäjä napsauttaa valikosta tulot-valintaa, jolloin avautuu taulukko tuloista.

KT6 *Tulon lisäys*

Alkuehdot
Käyttäjä on tulojen listauksessa.

Kuvaus
Käyttäjä napsauttaa lisää-nappia työkaluriviltä, jolloin avautuu Tulon lisäys -lomake. Käyttäjä täyttää lomakkeeseen tiedot ja napsauttaa lisää-nappia (Poikkeus: 1, 2). Tulo tallennetaan järjestelmään ja tulojen listaus päivitetään.

Poikkeukset
1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa tulojen listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.

KT7 *Tulon tietojen muokkaus*

Alkuehdot
Käyttäjä on tulojen listauksessa.

Kuvaus
Käyttäjä valitsee muokattavan tulon rivin, jolloin muokkaa-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa muokkaa-nappia, jolloin avautuu Tulojen muokkaus -lomake, jossa on muokattavan tulon tiedot valmiina. Käyttäjä tekee haluamansa muutokset ja napsauttaa tallenna-nappia (Poikkeus: 1, 2). Tulon tiedot päivitetään järjestel-

mään ja tulojen listaus päivitetään.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa tulojen listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.

KT8 *Tulon poisto*

Alkuehdot

Käyttäjä on tulojen listauksessa.

Kuvaus

Käyttäjä valitsee poistettavan tulon rivin, jolloin poista-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa poista-nappia, jolloin käyttäjältä pyydetään vahvistus tulon poistoon. Käyttäjä hyväksyy tulon poiston (Poikkeus: 1). Tulo poistetaan järjestelmästä ja tulojen listaus päivitetään.

Poikkeukset

1. Käyttäjä ei hyväksy poistoa, jolloin käyttäjä palaa tulojen listaukseen.

KT9 *Tulon hyväksyntä*

Alkuehdot

Käyttäjä on tulojen listauksessa.

Kuvaus

Käyttäjä valitsee hyväksytettävän rivin, jolloin maksuun-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa maksuun-nappia, jolloin

käyttäjälle avautuu Tulon maksu -lomake. Käyttäjä syöttää tarvittavat tiedot lomakkeelle ja napsauttaa hyväksy-nappia (Poikkeus: 1). Järjestelmä luo tulosta uuden kirjanpitotapahtuman, muuttaa tapahtumaan kohdistetun tilin saldoa (Poikkeus: 2), päivittää tuloon uuden eräpäivän (Poikkeus: 3) ja päivittää tulojen listauksen.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin käyttäjä palaa tulojen listaukseen.
2. Käyttäjä ei valinnut tulolle kohdistettua tiliä, jolloin tilin saldoa ei muuteta.
3. Tulolle ei ole asetettu toistuvuusväliä, jolloin tulolle ei lasketa uutta eräpäivää, vaan tulo poistetaan järjestelmästä.

KT10 *Menojen listaus*

Kuvaus

Käyttäjä painaa valikosta menot-valintaa, jolloin avautuu taulukko menoista.

KT11 *Menon lisäys*

Alkuehdot

Käyttäjä on menojen listauksessa.

Kuvaus

Käyttäjä napsauttaa lisää-nappia työkaluriviltä, jolloin avautuu Menon lisäys -lomake. Käyttäjä täyttää lomakkeeseen tiedot ja napsauttaa lisää-nappia (Poikkeus: 1, 2). Meno tallennetaan järjestelmään ja menojen listaus päivitetään.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa menojen listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.

KT12 *Menon tietojen muokkaus*

Alkuehdot

Käyttäjä on menojen listauksessa.

Kuvaus

Käyttäjä valitsee muokattavan menon rivin, jolloin muokkaa-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa muokkaa-nappia, jolloin avautuu Menon muokkaus -lomake, jossa on muokattavan menon tiedot valmiina. Käyttäjä tekee haluamansa muutokset ja napsauttaa tallenna-nappia (Poikkeus: 1, 2). Menon tiedot päivitetään järjestelmään ja menojen listaus päivitetään.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa menojen listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.

KT13 *Menon poisto*

Alkuehdot

Käyttäjä on menojen listauksessa.

Kuvaus

Käyttäjä valitsee poistettavan menon rivin, jolloin poista-nappi

aktivoituu työkalurivillä. Käyttäjä napsauttaa poista-nappia, jolloin käyttäjältä pyydetään vahvistus menon poistoon. Käyttäjä hyväksyy menon poiston (Poikkeus: 1). Meno poistetaan järjestelmästä ja menojen listaus päivitetään.

Poikkeukset

1. Käyttäjä ei hyväksy poistoa, jolloin käyttäjä palaa menojen listaukseen.

KT14 *Menon hyväksyntä*

Alkuehdot

Käyttäjä on menojen listauksessa.

Kuvaus

Käyttäjä valitsee hyväksytettävän rivin, jolloin maksuun-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa maksuun-nappia, jolloin käyttäjälle avautuu Menon maksu -lomake. Käyttäjä syöttää tarvittavat tiedot lomakkeelle ja napsauttaa hyväksy-nappia (Poikkeus: 1). Järjestelmä luo menosta uuden kirjanpitotapahtuman, muuttaa tapahtumaan kohdistetun tilin saldoa (Poikkeus: 2), päivittää menoon uuden eräpäivän (Poikkeus: 3) ja päivittää menojen listauksen.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin käyttäjä palaa menojen listaukseen.
2. Käyttäjä ei valinnut menolle kohdistettua tiliä, jolloin tilin saldoa ei muuteta.
3. Menolle ei ole asetettu toistuvuusväliä, jolloin menolle ei lasketa uutta eräpäivää, vaan meno poistetaan järjestelmästä.

KT15 *Velkojen listaus*

Kuvaus
Käyttäjä napsauttaa valikosta velat-valintaa, jolloin avautuu taulukko veloista.

KT16 *Velan lisäys*

Alkuehdot
Käyttäjä on velkojen listauksessa.

Kuvaus
Käyttäjä napsauttaa lisää-nappia työkaluriviltä, jolloin avautuu Velan lisäys -lomake. Käyttäjä täyttää lomakkeeseen tiedot ja napsauttaa lisää-nappia (Poikkeus: 1, 2). Velka tallennetaan järjestelmään ja velkojen listaus päivitetään.

Poikkeukset
1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa velkojen listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.

KT17 *Velan tietojen muokkaus*

Alkuehdot
Käyttäjä on velkojen listauksessa.

Kuvaus
Käyttäjä valitsee muokattavan velan rivin, jolloin muokkaa-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa muokkaa-nappia, jolloin avautuu Velan muokkaus -lomake, jossa on muokattavan velan tiedot

valmiina. Käyttäjä tekee haluamansa muutokset ja napsauttaa tallenna-nappia (Poikkeus: 1, 2). Velan tiedot päivitetään järjestelmään ja velkojen listaus päivitetään.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa velkojen listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.

KT18 *Velan poisto*

Alkuehdot

Käyttäjä on velkojen listauksessa.

Kuvaus

Käyttäjä valitsee poistettavan velan rivin, jolloin poista-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa poista-nappia, jolloin käyttäjältä pyydetään vahvistus velan poistoon. Käyttäjä hyväksyy velan poiston (Poikkeus: 1). Velka poistetaan järjestelmästä ja velkojen listaus päivitetään.

Poikkeukset

1. Käyttäjä ei hyväksy poistoa, jolloin käyttäjä palaa velkojen listaukseen.

KT19 *Velan lyhennys*

Alkuehdot

Käyttäjä on velkojen listauksessa.

Kuvaus

Käyttäjä valitsee hyväksytettävän rivin, jolloin maksuun-nappi

aktivoituu työkalurivillä. Käyttäjä napsauttaa maksuun-nappia, jolloin käyttäjälle avautuu Velan maksu -lomake. Käyttäjä syöttää tarvittavat tiedot lomakkeelle ja napsauttaa hyväksy-nappia (Poikkeus: 1). Järjestelmä luo velasta uuden kirjanpitotapahtuman, muuttaa tapahtumaan kohdistetun tilin saldoa (Poikkeus: 2), päivittää velkaan uuden eräpäivän (Poikkeus: 3), päivittää velan jäljellä olevan summan (Poikkeus: 4) ja päivittää velkojen listauksen.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin käyttäjä palaa velkojen listaukseen.
2. Käyttäjä ei valinnut maksulle kohdistettua tiliä, jolloin tilin saldoa ei muuteta.
3. Velalle ei ole asetettu toistuvuusväliä, jolloin velalle ei lasketa uutta eräpäivää.
4. Velka on kokonaan maksettu, jolloin velka poistetaan järjestelmästä.

KT20 *Tilien listaus*

Kuvaus

Käyttäjä napsauttaa valikosta tilit-valintaa, jolloin avautuu taulukko tileistä.

KT21 *Tilin lisäys*

Alkuehdot

Käyttäjä on tilien listauksessa.

Kuvaus

Käyttäjä napsauttaa lisää-nappia työkaluriviltä, jolloin avautuu Tilin lisäys -lomake. Käyttäjä täyttää lomakkeeseen tiedot ja napsauttaa

lisää-nappia (Poikkeus: 1, 2). Tili tallennetaan järjestelmään ja tilien listaus päivitetään.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa tilien listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.

KT22 *Tilin tietojen muokkaus*

Alkuehdot

Käyttäjä on tilien listauksessa.

Kuvaus

Käyttäjä valitsee muokattavan tilin rivin, jolloin muokkaa-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa muokkaa-nappia, jolloin avautuu Tilin muokkaus -lomake, jossa on muokattavan tilin tiedot valmiina. Käyttäjä tekee haluamansa muutokset ja napsauttaa tallenna-nappia (Poikkeus: 1, 2). Tilin tiedot päivitetään järjestelmään ja tilien listaus päivitetään.

Poikkeukset

1. Käyttäjä napsauttaa peruuta-nappia, jolloin lomake sulkeutuu ja käyttäjä palaa tilien listaukseen.
2. Käyttäjä jättää pakollisia tietoja täyttämättä, jolloin käyttäjää pyydetään täyttämään puuttuvat tiedot lomakkeelle.

KT23 *Tilin poisto*

Alkuehdot

Käyttäjä on tilien listauksessa.

Kuvaus

Käyttäjä valitsee poistettavan tilin rivin, jolloin poista-nappi aktivoituu työkalurivillä. Käyttäjä napsauttaa poista-nappia, jolloin käyttäjältä pyydetään vahvistus tilin poistoon. Käyttäjä hyväksyy tilin poiston (Poikkeus: 1). Tili poistetaan järjestelmästä ja tilien listaus päivitetään.

Poikkeukset

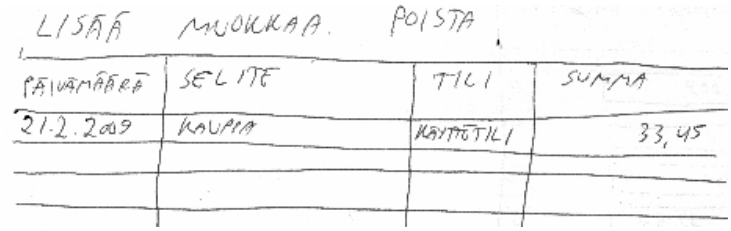
1. Käyttäjä ei hyväksy poistoa, jolloin käyttäjä palaa tilien listaukseen.

Käyttöliittymä

Käyttöliittymästä tehdään yksinkertainen ja selkeä, jotta sen käytön opetteluun ei tuhlautuisi liikaa aikaa. Kuvassa 1 on hahmotelma päänäkymästä, joka koostuu otsikosta, valikosta sekä vaihtuvasta sisältöikkunasta.

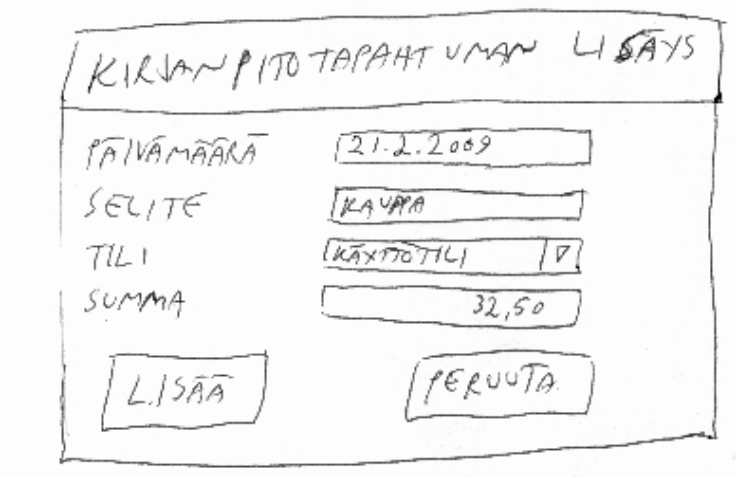
Kuva 1: Käyttöliittymän ulkoasu

Sisältöikkunan sisältö vaihtuu valikon valintojen perusteella. Sisältöikkunan yläreunaan tulee työkalurivi, jossa on näkymän tarvitsemat toiminnot. Jäljelle jäävään osaan tulee taulukko näkymän tiedoista. Kuvassa 2 on kirjanpitolistauksen hahmotelma, joka toimii myös muiden näkymien pohjana.



Kuva 2: Kirjanpitotapahtumien listaus

Kuvassa 3 on hahmotelma Kirjanpitotapahtuman lisäys -lomakkeesta, joka toimii myös pohjana muille lisäyslomakkeille.



Kuva 3: Kirjanpitotapahtuman lisäys -lomake

Muokkauslomake on muuten sama kuin lisäyslomake, paitsi otsikko on "Kirjanpitotapahtuman muokkaus" ja lisää-napin teksti on "tallenna" sekä lomakkeelle asetetaan automaattisesti muokattavan tapahtuman tiedot.

Poistovahvistus on yksinkertainen viestilaatikko, jossa lukee "Haluatko varmasti poistaa valitun rivin" sekä sisältää valinnat "kyllä" ja "ei".

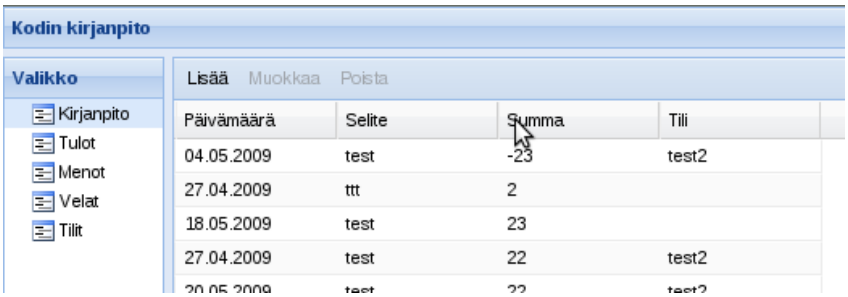
Asentaminen

Sovellus tarvitsee toimiakseen Java JDK 6-ajoympäristön, tietokannan sekä tomcatin. Sovellus toimitetaan WAR-pakettina, joka sijoitetaan tomcatin webapps-hakemistoon. Tomcat purkaa ja ottaa käyttöön sovelluksen automaattisesti.

Tomcat purkaa sovelluksen webapps\kirjanpito-hakemistoon, josta myös löytyy Hibernaten konfigurointi-tiedosto hakemistosta webapps\kirjanpito\WEB-INF\classes, johon sijoitetaan käytettävän tietokantayhteyden tiedot.

Käyttö

Sovelluksen käyttö tapahtuu selaimen avulla. Kun palvelinpuoli on käynnistetty, asiakasliittymään voidaan ottaa yhteys selaimella menemällä esimerkiksi osoitteeseen http://localhost:8080/kirjanpito.

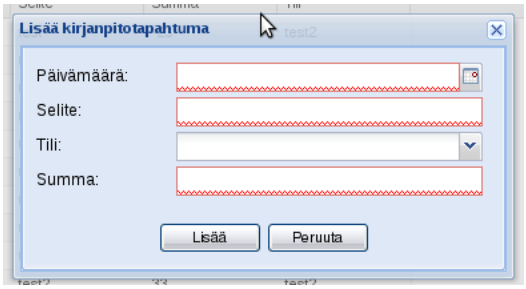


Kuva 1: Kirjanpitonäkymä

Kuvassa 1 näkyy kirjanpitonäkymä, jossa on taulukko kirjanpitotapahtumista sekä ylhäällä työkalurivi, jossa on näkymän

toiminnot. Vasemmassa laidassa olevasta valikosta valitaan haluttu näkymä.

Rivien lisäys onnistuu napsauttamalla lisää-nappia, jolloin avautuu kuvan 2 mukainen lomake. Lisäyksen voi peruuttaa painamalla peruuta-nappia, tai hyväksyä painamalla lisää-nappia.



Kuva 2: Kirjanpitotapahtuman lisäys -lomake

Muokkaa- ja poista-napit tulevat aktiivisiksi, kun valitsee jonkin rivin listauksesta, napsauttamalla riviä hiirellä. Jonka jälkeen valitun rivin voi poistaa painamalla poista-nappia. Rivin muokkaus onnistuu painamalla muokkaa-nappia, jolloin avautuu Tapahtuman muokkaus-lomake.

Lisäksi tulo-, meno- ja velkanäkymissä on maksuun-nappi, joka myös aktivoituu rivivalinnan jälkeen. Tällä toiminnolla toimeenpannaan ennakkoon ilmoitettu tapahtuma, jonka tietoja pystyy muokkaamaan avautuneesta lomakkeesta.