Praktikarakenduse nõuded ja soovitused rakenduse sise-ehitusele (disain, rakenduse sisemisele ülesehitusele esitatavad nõuded) aines "IDU0200", 2014 kevadsemester.

Viimati muudetud: 17.02.2014

Rakenduse disain ei ole täppisteadus aga on piisavalt palju eeskujusid ja disainimustreid mida järgides on võimalik ehitada professionaalsel tasemel rakendus. Iga disaineri lahendus on siiski teatud osas erinev.

Suuremat osa siin dokumendis viidatud teemadest/mustritest/lahendustest käsitletakse ka harjutustundide või loengute näidetes kuid kattuvus pole kindlasti 100%-line – natuke tuleb ise ka lisaks uurida.

Üldine ideloogia selle rakenduse disainis on **tükeldus** – rakenduse kood peab olema tükeldatud selge skoobiga (ülesandega) allsüstemeideks ja klassideks mille meetodid on väikesemahulised ja täidavad ainult ühte ülesannet. Rakendus koosneb **väikestest**, selge funktsiooniga osadest.

1. MVC mustri järgimine - kontroller.

Rakenduse disain peab järgima MVC-mustrit.

Teie töös peab olema kasutatud vähemalt ühte „Controlleri“ tüüpi komponenti.

> Märkus: kui kasutate rakendusraamistikku (*application framework*) siis võib „Front Controlleri“ funktsiooni täita ka see rakendusraamistik ja sellisel juhul ei ole seda tüüpi kontrollerit vaja ise teha. „Controller“ tüüpi komponendid on rakendusraamistike kasutamisel teil siis nagunii kasutusel (nt. Springi kontrollerid, Ruby on Rails-i kontrollerdi,..)

Rakenduse juhtloogika (kontrolleri osa) peab olema eraldatud eraldi komponentidesse (kontrolleritesse). Komponendi tüüp pole oluline - võib olla servlet, JSP-leht, php-leht või mingit muud tüüpi komponent - sõltuvalt tehnoloogiast mida kasutate.Vajadusel kasutada hierarhilisi kontrollereid: peakontroller ja alamkontrollerid.

Kontroller ei tohi :

nõue 1.1. sisaldada otse andmebaasiga suhtlevat koodi (JDBC/SQL)

nõue 1.2. genereerida väljundit (sisaldada brauserisse saadetatavat HTML-i genereerivat koodi)

2. MVC mustri järgimine – esitluskomponendid ja „mudeli“ väljakutsed.

Esitluskomponentideks (*presentation layer*) nimetame komponente mis otseselt tegelevad kasutajatele pildi genereerimisega/saatmisega. Need komponendid on Java platvormil servlet- või JSP tüüpi (ka siis kui väljundi saatmine brauserisse on usaldatud mõne raamistiku või valmiskirjutatud komponendi hoolde). Selliseid komponente nimetame ka VAATEKS (Viev).

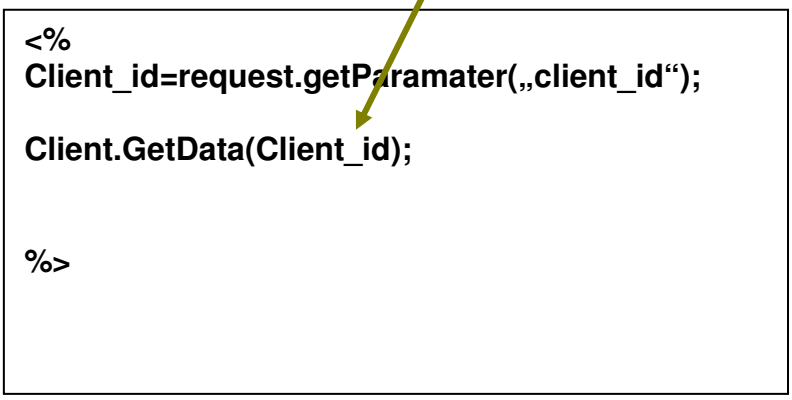
Esitluskomponendile (ehk HTML-kuva genereerimise komponendile) esitatavad nõuded:

nõue 2.3. Esitluskomponent (servlet, JSP) ei tohi sisaldada andmebaasiga suhtlevat koodi.

nõue 2.4. Esitluskomponendid saavad oma andmeid „mudeli“ (siin võib neid nimetada ehk ka andmeid sisaldavateks komponentideks) komponentide poole pöördudes. Sellised pöördumised mudeli poole on võimalik kirjutada esitluskomponendi koodi. Näiteks : JSP-lehekülg mis näitab kliendi andmeid:

```
<%

Client.GetData();


%>
```

Kuid – ei ole lubatud kirjutada esitluskomponentide koodi sisse argumente mida mudel võiks kasutada konkreetsete andmete hankimiseks ja vaatele (ja mis on saadud näiteks HTTP-pöördumisest) . Seega järgnev näide on VALE:



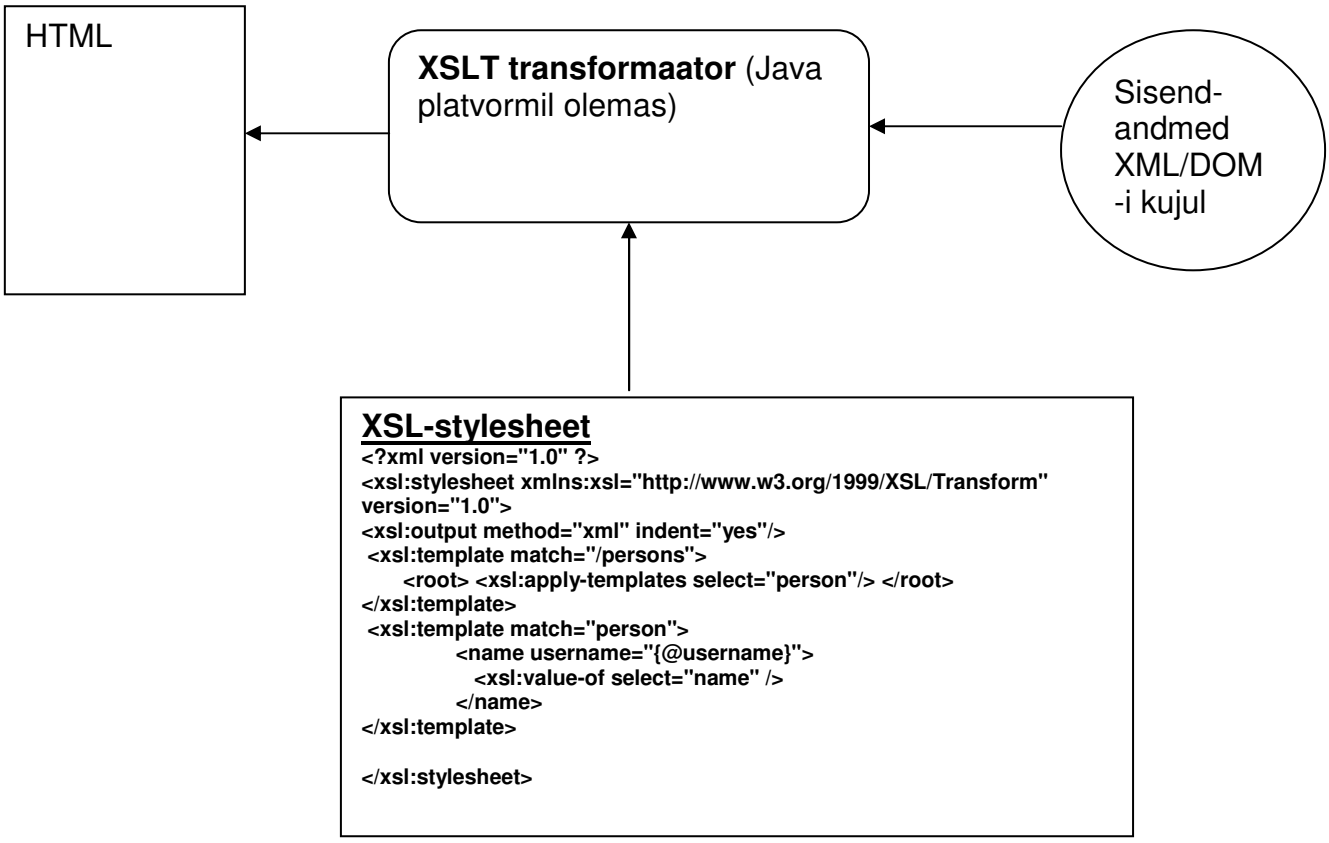
Mudeli initsialiseerimisega (käimatõmbamisega, sisendparameetrite saatmisega) tegeleb rakenduse juhtloogika mis asub kontrollerites.

Kui rakenduse tööjärg jõuab vaate komponentide käivitamiseni (tõmmatakse käima HTML-i genereeriv JSP-leht või servlet) on „Mudel“ juba käivitatud ja mudeli vastavad objektid juba andmetega täidetud.

Kuidas sellist tööjaotust **kontrolleri-mudeli-vaate** vahel konkreetselt teha – selle kohta tuleb näiteid loengus ja harjutustundides.

3. Esitluskomponent ja XSLT. Andmete eraldamine formaatimisloogikast

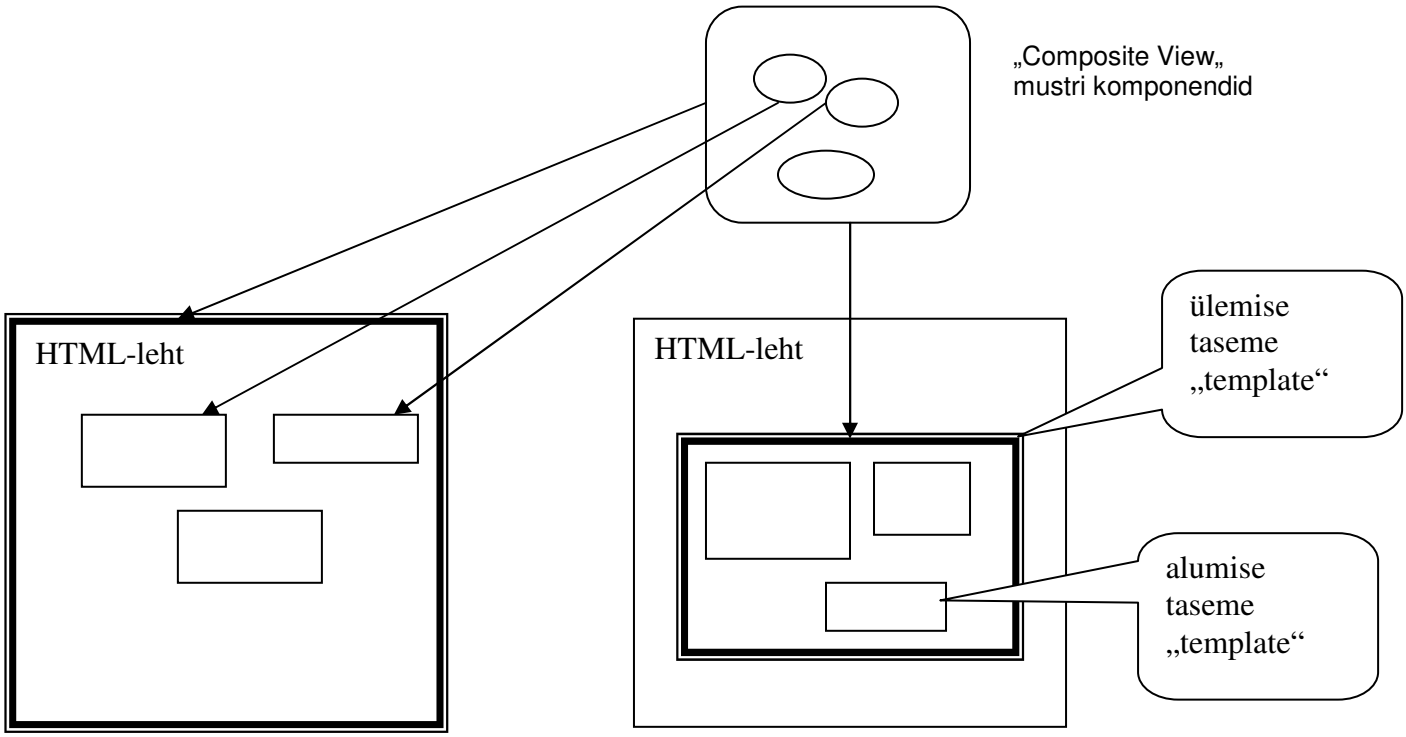
nõue 3.5 Vähemalt ühe veebilehe või kasutajavormi või alamvormi HTML-i genereerimiseks kasutada XSLT-teisendust.



XSLT puhul teisendatakse sisendandmed XML-i ja XSL transformaator konvreteerib sisendandmed XSL-stylesheed-is sisalduvate reeglite alusel HTML-iks (või uueks XML-iks või *plain text*-iks – aga neid võimalusi pole meil vaja).

4. „Composite View“ mustri kasutamine „vaates“.

<u>nõue 4.6.</u> Vähemalt ühe veebilehe (või keerulisema html-vormi või lehel alamosa) genereerimise realiseerimisel tuleks kasutada „Composite View“ mustrit.



Komposiitmustri idee on lühidalt selles et keerulise, mitmest alam-osast koosneva HTML-vaate genereerimisel kasutatakse nende alam-osade (veebilehe või html-vormi osade) genereerimiseks eraldi veebilehe osasid („templates“) mis koostöös genereerivad tervikvaate – HTML-lehe (näiteks suure, keerulise HTML-vorm mis omakorda koosneb mitmetest alamvormidest).

Võib öelda ka nii – tegemist on šablooniga („template“) mis koosneb omakorda šabloonidest.

5. „Validator“-tüüpi objekti(klassi) kasutamine.

Praktikaülesandes on mitmeid tegevusi mille sisuks on andmete salvestamine – brauserist , HTML-vormist saadetud andmed võetakse serveri poole peal vastu ja nad tuleb andmebaasi salvestada (näiteks „kliendi andmete salvestamine“, „kliendi lisamine“ ,..).

HTML-vormist serverisse saabunud andmeid on ebausaldusväärsed ja nende andmetüübiks on tekst. Andmeid (nende formaate, pikkusi, HTML-koodi ja Javascripti sisaldumist) tuleb kontrollida

nõue 5.7. asutage vähemalt ühes sellises andmete salvestamise tegevuses „validator“-tüüpi objekti.

Validator-objekti sisendiks on kontrollimist vajavad andmeid.

Validator objekti funktsionaalsuseks on sisendandmete kontroll.

Validator-objekti väljundiks on andmete kontrolli tulemused. Andmete kontrolli tulemused koosnevad :

1) infost (mingi muutuja väärtusest) selle kohta kas andmeid tervikuna olid sobivad (näiteks – „kliendi andmed on korrektsed“, client_ok=1 või midagi taolist)

2) infost iga konkreetse sisend-andmete atribuudi (andmevälja) kohta mis sisaldab kontrolli tulemust ja veateadet

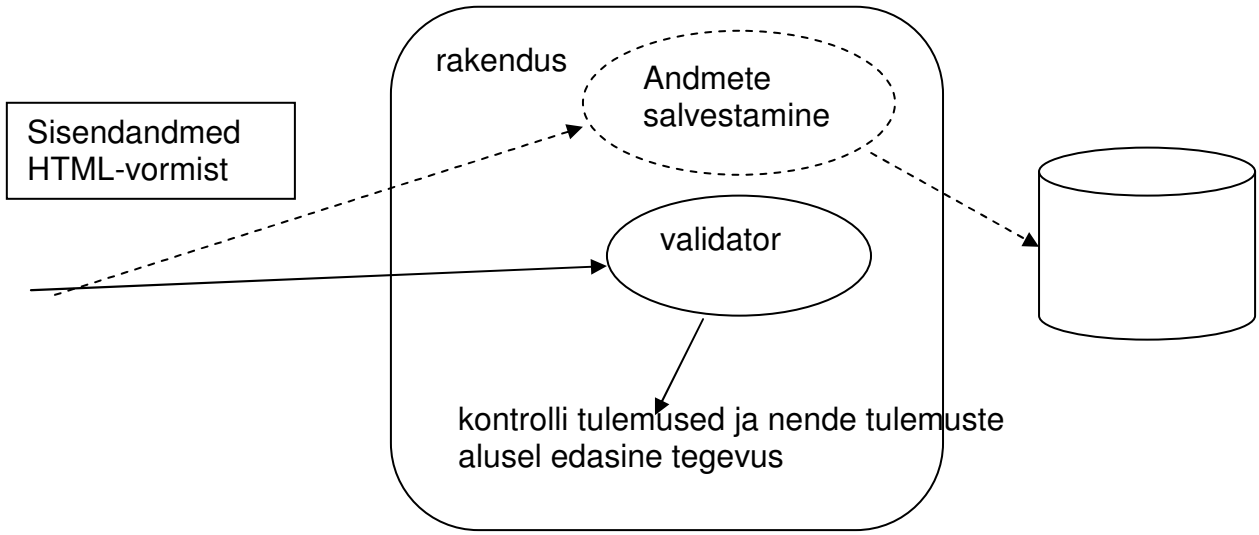
Näiteks:

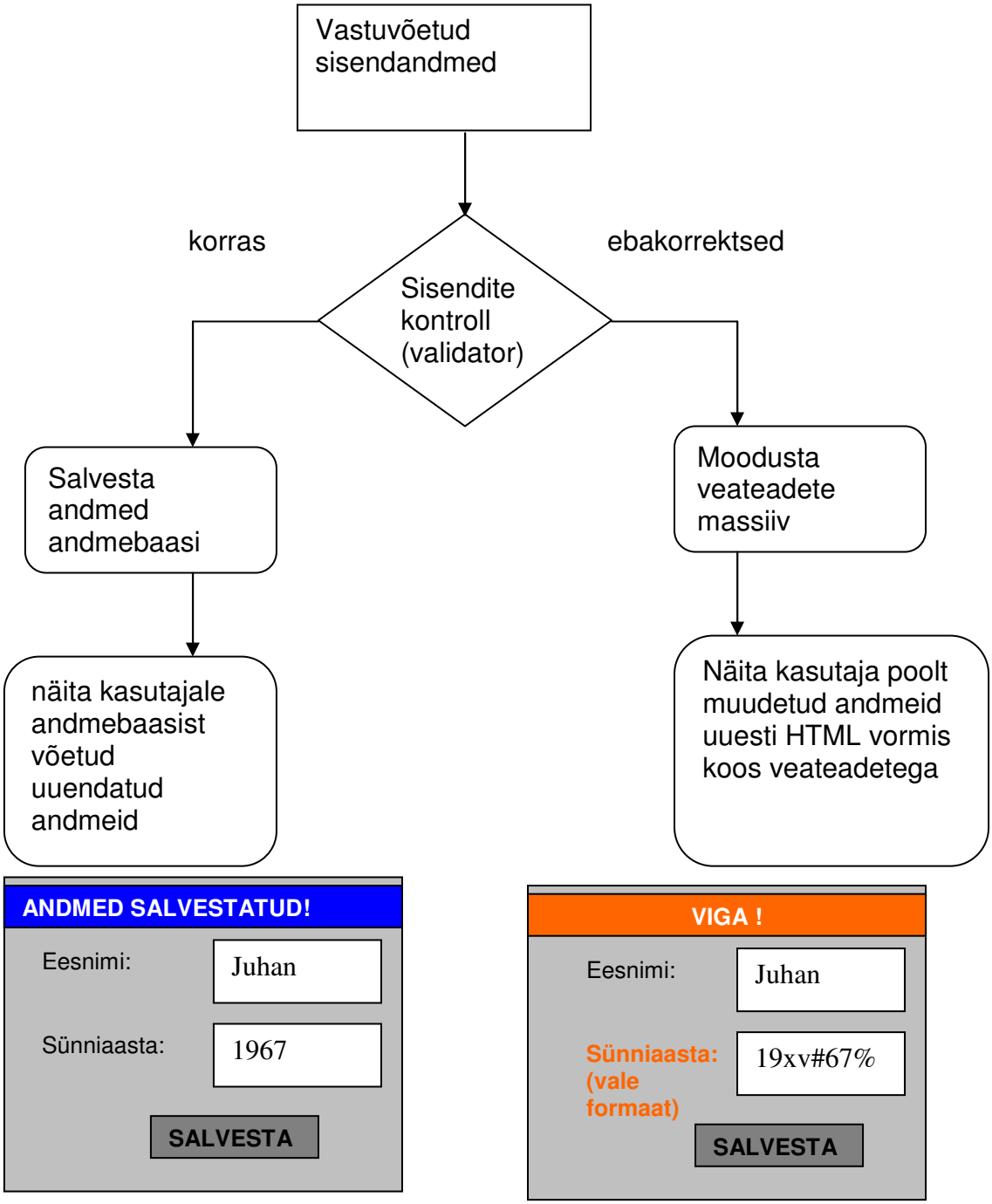
first_name_ok = 0 „Eesnimi sisaldas numberid“
birth_date_ok = 0 „Kuupäeva formaat vale“
last_name_ok = 0 „Perekonnanimi tühi“
aadress_ok = 1 „“

On loogiline arvata et selliseid validatori väljundandmeid - infot iga konkreetse andmevälja korrektsuse kohta ja vastavat veateadet (kui konkreetse andmevälja sisu oli ebakorrktne) on hea kasutada kasutajale HTML-vormi andmete uuesti-näitamisel – nüüd juba koos vigaste andmeväljade esiletõstmise ja ekraanile trükitud veateadetega.

Teie ülesanne on vähemalt ühe andmete sisestuse/muudatuse korral teha nii et vigaste andmete salvestamise katsel püütaks vead validatori poolt kinni ja näidataks IGA andmevälja juures (mille sisu oli ebakorrektne) veateadet (vaata allpool olevat kasutajavormide näidet)

Lisage oma rakenduse disaini üks selline validator-objekt nii et selle validatori kontrolli tulemusi (väljundandmeid) kasutataks teie rakenduses selleks et otsustada mida teha edasi.




Vastuvõetud sisendandmed
Sisendite kontroll (validator)
korras
ebakorrektsed
Salvesta andmed andmebaasi
Moodusta veateadete massiiv
näita kasutajale andmebaasist võetud uuendatud andmeid
Näita kasutaja poolt muudetud andmeid uuesti HTML vormis koos veateadetega
ANDMED SALVESTATUD!
Eesnimi:
Juhan
Sünniaasta:
1967
SALVESTA
VIGA !
Eesnimi:
Juhan
Sünniaasta: (vale formaat)
19xv#67%
SALVESTA

6. Andmebaasiühendus konfigureeritav.

nõue 6.8. Rakenduse andmebaasiühenduse parameetrid (andmebaasi serveri aadress, andmebaasi nimi, andmebaasi port, andmebaasi kasutajanimi ja parool) ei tohiks olla koodi sisse kirjutatud vaid need peaks olema salvestatud konfiguratsioonifailis. Väga hästi sobib selleks näiteks Java platvormil kasutatavad „properties“-tüüpi failid (vaata selle kohta harjutustundides tulevaid näiteid)

Sobib ka suvaline muu teie end poolt valitud faili tüüp (xml, txt), nimi ja asukoht.

7. Täiendavate disainimustrite kasutamine.Vähemalt 3 mustrit veel.

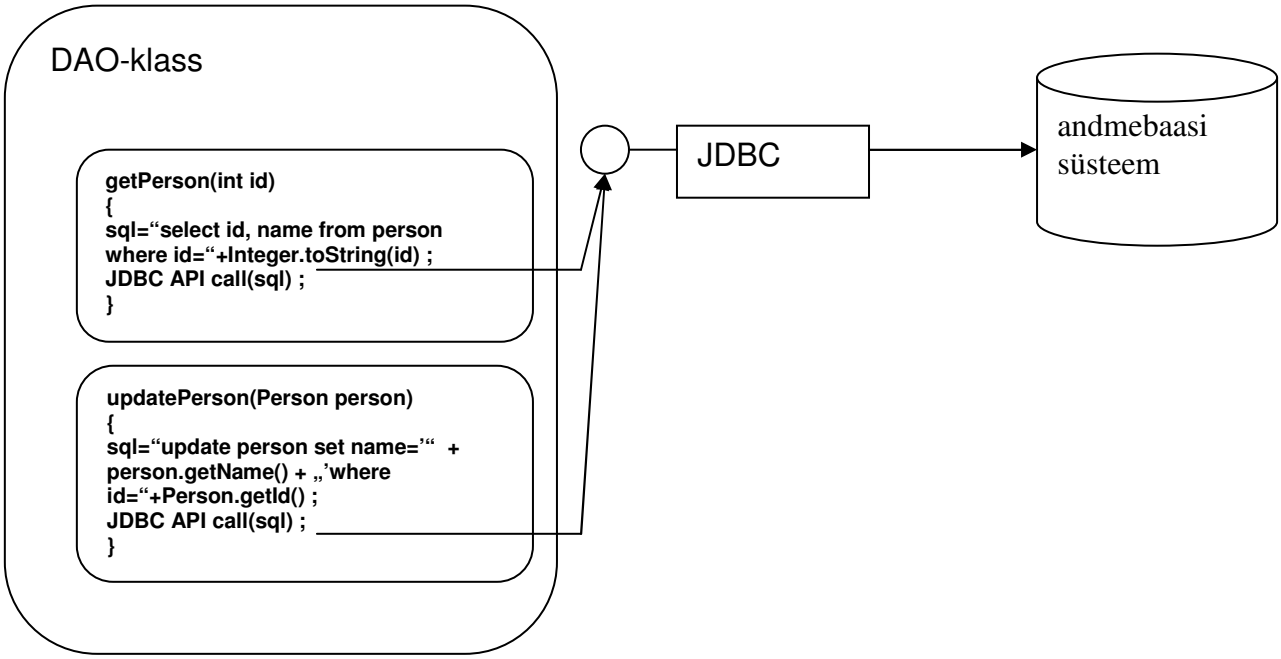
nõue 6.9. Kasutage rakenduses täiendavalt selliseid disainimustreid mida selles materjalis mainitud ei ole (siin juhendis on mainitud järgmiseid idsianimustreid : „MVC“, „Composite View“, „Data Access Object“).Selliseid täiendavaid, teie enda poolt valitud mustreid peaks olema vähemalt 3. Proovige oma rakenduses neid vähemalt kolme mustrit kasutada.

Pange rakenduse dokumentatsiooni kirja kus teie rakenduses nende mustrite realisatsioonid on (vaata alt nõuet **14.17**)

8. ORM-tarkvara kasutamine + DAO/SQL (nendele kes kasutavad Javat).

<u>nõue 9.10.</u>

- Osade andmebaasiga seotud operatsioonide realiseerimiseksl tuleb andmebaasiga suhtlemiseks kasutada ORM-vahekihi tarkvara. (Object Relational Mapping)
- Osade andmebaasidega seotud operatsioonide realiseerimiseks tuleb kasutada JDBC-d (Java Database Connectivity) ja SQL lauseid. JDBC-d tuleb kasutada ***Data Access Object*** mustrit (DAO) realiseeriva klassi sees.(Ehk siis teiste sõnadega – andmebaasiga suhtleb DAO klass mille meetodid sisaldavad andmebaasi poole pöördumisi SQL-lausetega üle JDBC API)

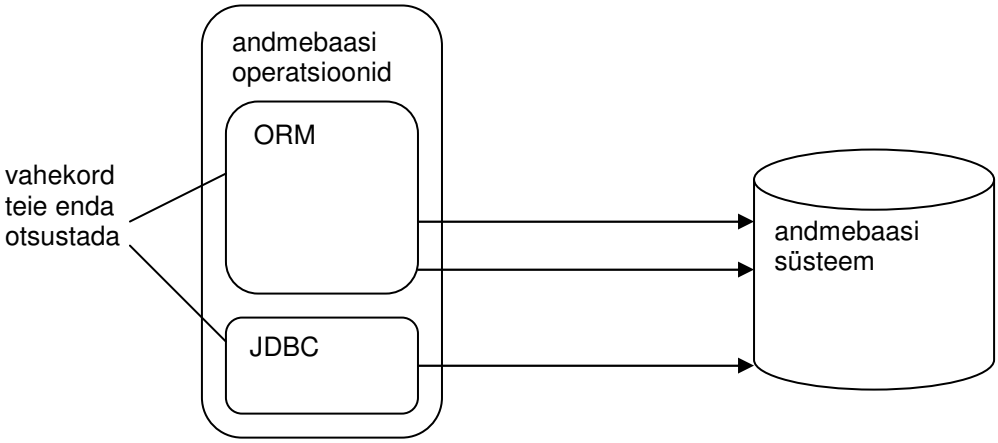


Märkuseid:

Rakenduse ORM-i kasutavas andmebaasiga suhtlemise osas võib ka kasutada DAO-mustrit, üldiselt ongi nii et ORM-vahevara kasutamine ei tähenda seda et vajadus DAO-mustri järele kaoks.

Tõenäoliselt te töö käigus leiate et ORM vahevara kasutamine on lihtsam kui JDBC-kasutamine.Te võite enamuse andmebaasiga suhtlusest realiseerida ORM-i abil , nii et ainult üks DAO klassi teete JDBC kasutamisega. Kui soovite siis võite teha ka vastupidi – enamus andmebaasi operatsioone on JDBC kasutamisega ja ainult mõned ORM-i kasutamisega.

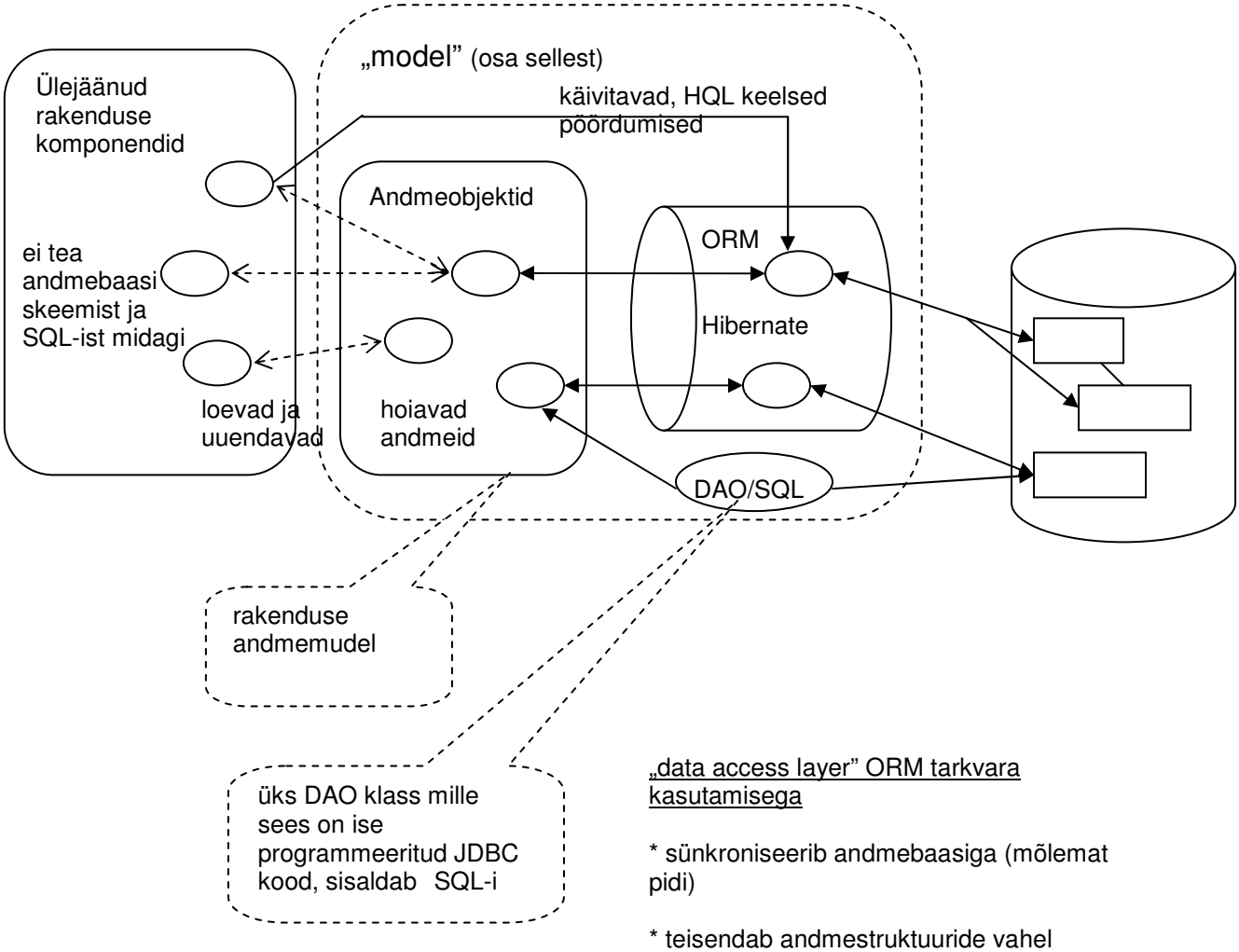
Oluline on et teie rakenduses oleks kasutatud mõlemat andmebaasiga suhtlemise meetodit – madalatasemelise JDBC API väljakutsetele ehitatud DAO klasse ja ORM-i kasutavaid komponente.



ORM – Object-Relational-Mapping , objekt-relatsioonilise teisenduse tarkvara/vahekiht.

Java platvormil on kõige levinumaks ORM-tarkvaraks Hibernate (https://www.hibernate.org/) mille kasutamiseks tuleb juttu ka harjutustundides.

SQL keeles andmebaasiga suhtlemise nõue (ja DAO objekti tegemise nõue) ei kehti nende rakendusraamistike/platvormide kasutajatele mille puhul ongi vaikimisi andmebaasiga suhtlemiseks olemas sellesse raamistikku sisse ehitatud ORM funktsionaalsus ja kus SQL-i üldjuhul ei kasutata (Ruby on Rails, Python/Django)



„data access layer" ORM tarkvara kasutamisega

* sünkroniseerib andmebaasiga (mõlemat pidi)

* teisendab andmestruktuuride vahel (rakendus <-> db), struktuurid ei ole (päris) samad

* sünkroniseerimist ja SQL-keelset suhtlust db serveriga ole vaja ise kirjutada – ORM teeb (konfiguratsiooni abil) ära.

9. SQL-protseduuri („andmebaasi salvestatud protseduuri") kasutamine .

nõue 9.11. Kasutage ühes andmeuuendus-tegevuses SQL-protseduuri väljakutsumist . Töö andmetega (SQL-laused) teeb siis ära andmebaasi salvestatud

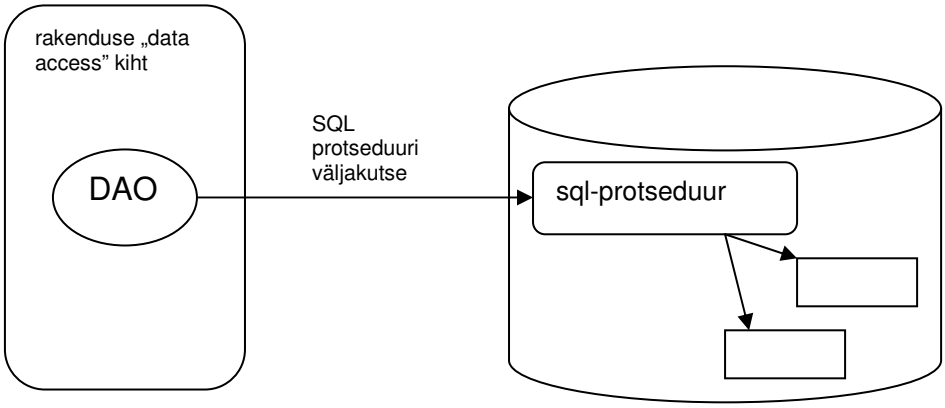
SQL-protseduur ja rakenduses ei sisaldu selle tegevusega seotud SQL-lauseid – rakenduses on ainult SQL-protseduuri väljakutse.

Nii PostgreSQL , Oracle kui MySQL toetavad SQL-protseduuride kasutamist..

Näiteks – kliendi andmete muutmine SQL-protseduurina.

Mõelge selle peale millised on SQL-protseduuride kasutamise eelised ja puudused – see teema võib tulla töö kaitsmisel küsimuseks.

> Kuna te peate tegema vähemalt ühe Data Access Object-i mis kasutab JDBC liidest andmebaasiga suhtlemiseks siis on mõistlik see üks (või soovi korral rohkem) SQL protseduuri väljakutset panna mõnda selle JDBC-DAO klassi meetodisse. Põhimõtteliselt saab siiski ka läbi ORM-i protseduure välja kutsuda.



Rakendus läheb nüüd mõnes mõttes mitmekihilisemaks ja keerulisemaks:

DAO + andmetabelid

asemel tekib

DAO + SQL protseuur + andmetabelid

SQL-protseduuri kasutamine on praktikaülesandesse sisse toodud mitte sellepärast et see oleks kuidagi halvem või parem kui otse DAO-st (üldisemalt-rakendusest) andmetabelutega töötamine – protseduuride kasutamisel on nii eeliseid kui puuduis. Praktikaülesande nõudeks on SQL-protseduuri kasutamine tehtud sellepärast et praktikilistes rakendusarhitektuurides selliseid protseduure erinevatel põhjustel tihti kasutatakse ja seepärast võiks olla ettekujutus kuidas sellist protseduuri siis oma rakenduse külge ühendada.

SQL protseduuri ei pea kasutama rohkem kui üks kord rakenduses – näiteks mingi üksik andmete lisamine, muutmine või kustutamine võiks olla tehtud SQL-protseduuriga.

10. AJAX-stiilis väljakutse kasutamine andmete kuvamiseks, salvestamiseks ja salvestamistulemuste näitamiseks ilma veebilehe ümberlaadimiseta.

nõue 10.12. Kasutage XMLHttpRequest objekti (nn. „AJAX-väljakutse“) andmete tõmbamiseks serveril, andmete tagasisaatmiseks serverile ja selle tagasisaatmise tulemuste näitamiseks veebilehel.

1. Valige oma rakenduses suvaline infoobjekt (klient, leping, toode, kataloog) ja tehke nii et seda saab veebilehel välja valida (klõpsates näitesk toote nime peal).

2. **Esimene AJAX-i väljakutse**. Väljavalitud infoobjekti andmed küsitakse serverist AJAX-väljakutse abil – see tähendab et toote (antud näites auto) nimele klõpsates ei laadita veebilehte ümber vaid käivitatakse Javascript mis saadab serverile HTTP-pöördumise , kasutades XMLHttpRequest objekti. AJAX-stiilis HTTP-pöördumine võib olla nii GET kui POST-tüüpi, see pöördumine saadetakse mingile serverirakenduse komponendile mille poole on võimalik pöörduda HTTP-pöördumise abil (Java veebirakenduses siis **servlet** või **JSP-leht**)

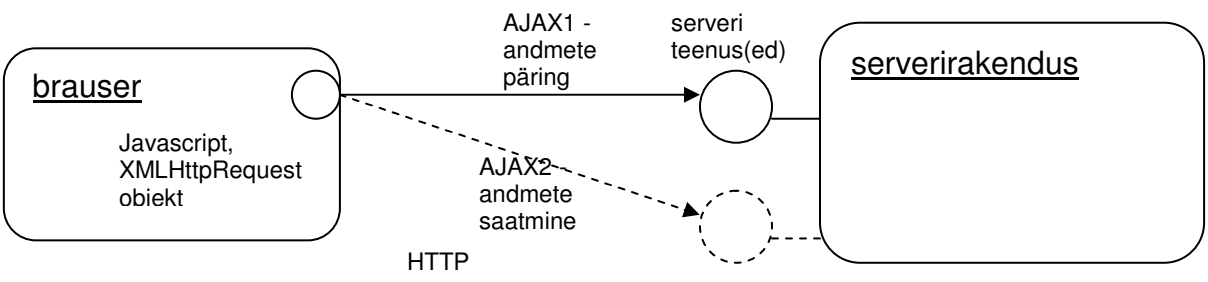
Põhimõtteliselt võib sellist komponenti nimetatda kliendirakendust teenindavaks **serveriteenuseks**.

Server saadab päringule vastuse , sellest vastusest võetakse andmed ja nende alusel

a) luuakse uus alamvorm veebilehel ja sinna alamvormi kirjutatakse serveril saadud andmed
b) kasutatakse mingit olemasolevat alamvormi mis on veebilehel juba olema. Sinna alamvormi laaditakse andmed kasutades Javascripti ja DOM-i.

Näites on kasutatud varianti b) - on tehtud nii et esimest korda (peale veebilehe laadumist) auto nimel klõpsates näidatakse kasutajale veebilehele juba olemasolevat (aga tema eest algselt peidetud – vaadake näite veebilehe HTML „source“-i) alamvormi kuhu laetakse serveril saadud andmed.

3. **Teine AJAX-i väljakutse ja andmete kontroll serveril**. Alamvormil näidatavad andmed (vähemalt paar andmevälja) peavad olema kasutaja poolt muudevad. Alamvormil peab olema andmete salvestamise (serverile saatmise) nupp. Kui kasutaja on vajutanud salvestamise nuppu tehakse teine pöördmumine serverile XMLHttpRequest objekti vahendusel – saadetakse serverile kasutaja poolt muudetud andmed. Serveriteenus (servlet, JSP-leht) millele andmed saadetakse võib olla sama kust andmed esimese AJAX-i päringuga küsiti aga võib olla ka eraldi komponent (mis tegeleb ainult salvestuspäringute vastuvõtmisega).



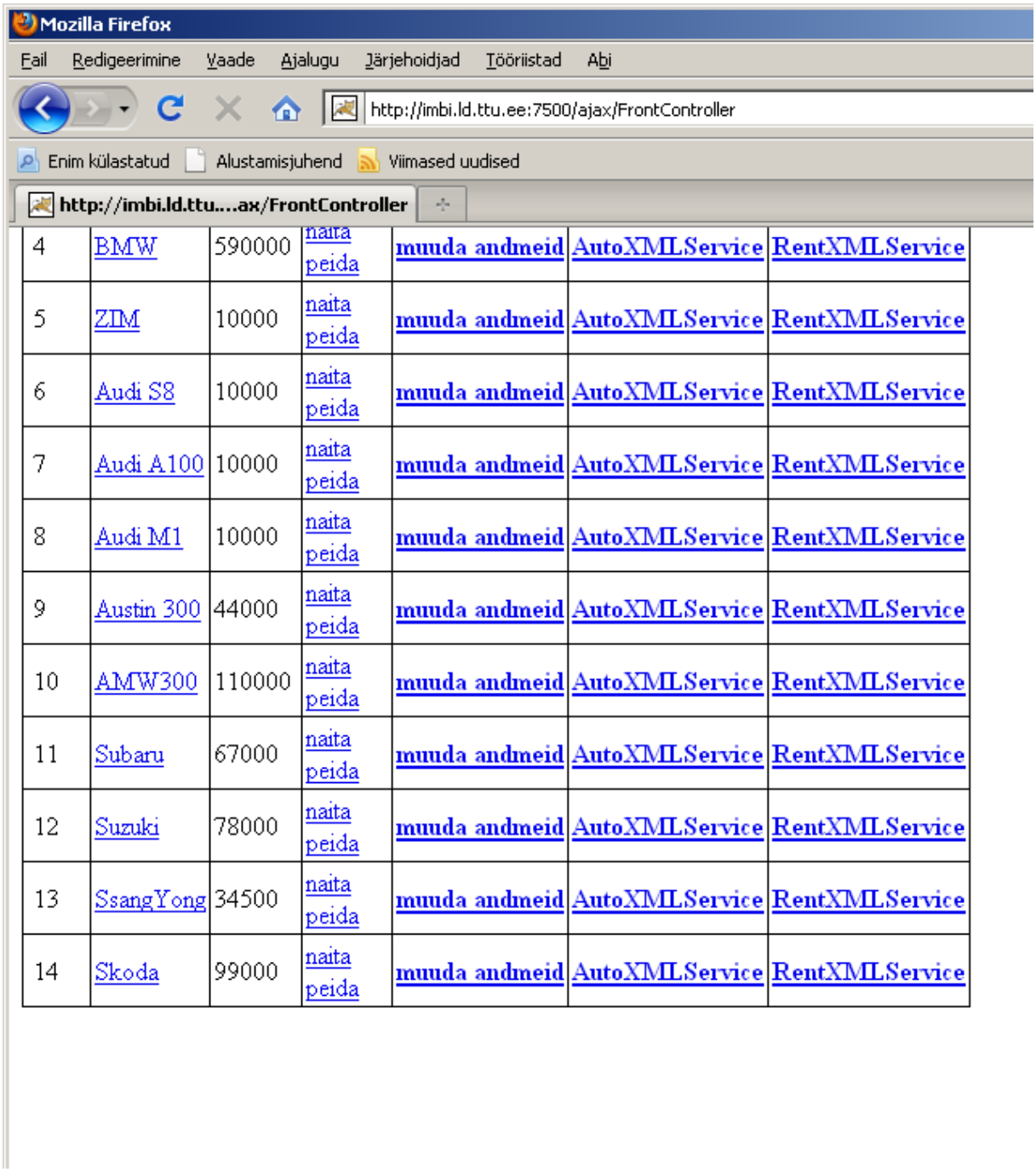
Serveril tuleb nüüd teha saadetud andmete kontroll ja salvestamine. Kindlasti peab olema realiseeritud vähemalt ühe anmdevälja kontroll ja seda nii et kasutajal oleks võimalik sinna anmdevälja midagi valesti sisestada. Näiteks – vormil on mingi numbriväli kuhu on võimalik sisestada ka teksti (s.t. ei ole näiteks Javascriptiga tehtud teksti sisestamine võimatuks). Oluline on et alamvormilt oleks võimalik serverile saata vigaseid andmeid . Kõige lihtsam on lisada alamvormile mingi numbriväli mille sisu siis serveri kontrollitakse – kas on number või mitte?

Serveril tehtud kontroll annab salvestus-päringule vastuseks mingi tulemuse (piisab kahest valikust – „andmed õiged“ või „andmed ei ole korrektsed“), seda tulemust kasutatakse kasutajale salvestustulemuste kohta info kuvamisel – jällegi Javascripti+DOM-i abil.

Vaata näidet aadressilt:

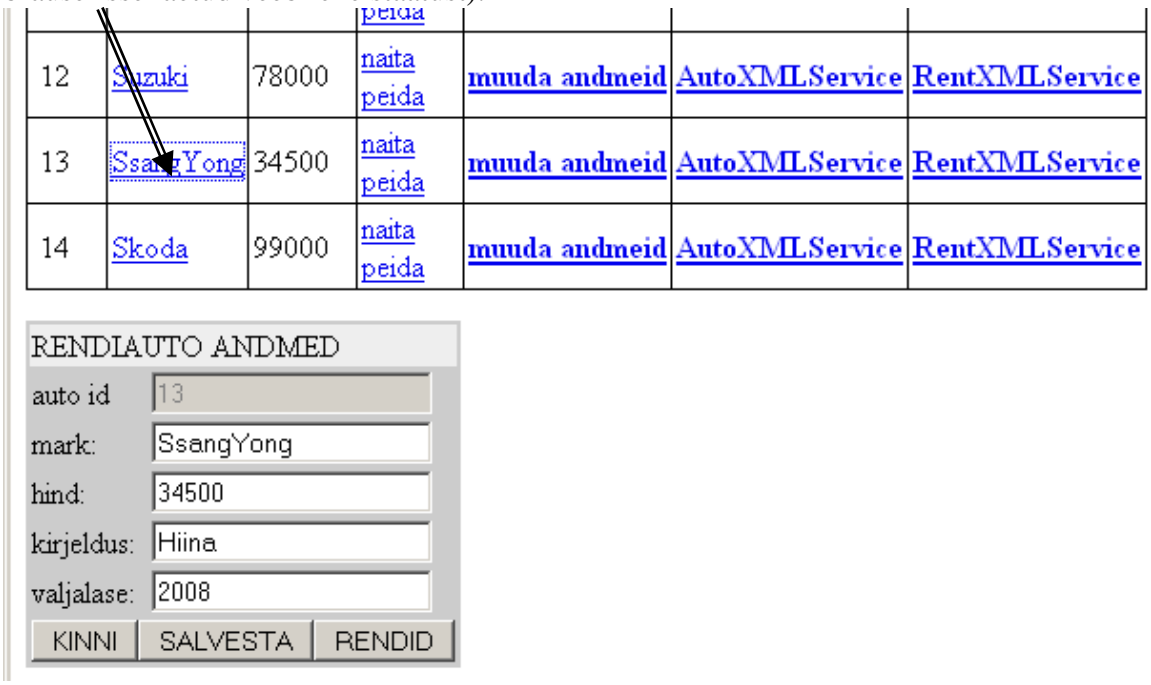
http://imbi.ld.ttu.ee:7500/ajax/FrontController

1.Veebileht algolekus:

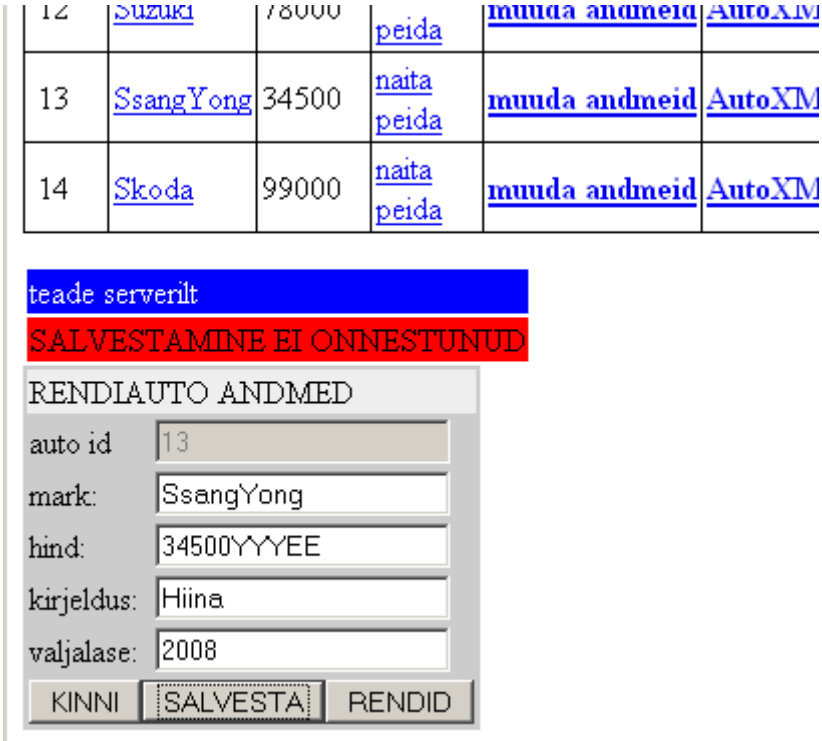


4	BMW	590000	naita peida	muuda andmeid	AutoXMLService	RentXMLService
5	ZIM	10000	naita peida	muuda andmeid	AutoXMLService	RentXMLService
6	Audi S8	10000	naita peida	muuda andmeid	AutoXMLService	RentXMLService
7	Audi A100	10000	naita peida	muuda andmeid	AutoXMLService	RentXMLService
8	Audi M1	10000	naita peida	muuda andmeid	AutoXMLService	RentXMLService
9	Austin 300	44000	naita peida	muuda andmeid	AutoXMLService	RentXMLService
10	AMW300	110000	naita peida	muuda andmeid	AutoXMLService	RentXMLService
11	Subaru	67000	naita peida	muuda andmeid	AutoXMLService	RentXMLService
12	Suzuki	78000	naita peida	muuda andmeid	AutoXMLService	RentXMLService
13	SsangYong	34500	naita peida	muuda andmeid	AutoXMLService	RentXMLService
14	Skoda	99000	naita peida	muuda andmeid	AutoXMLService	RentXMLService

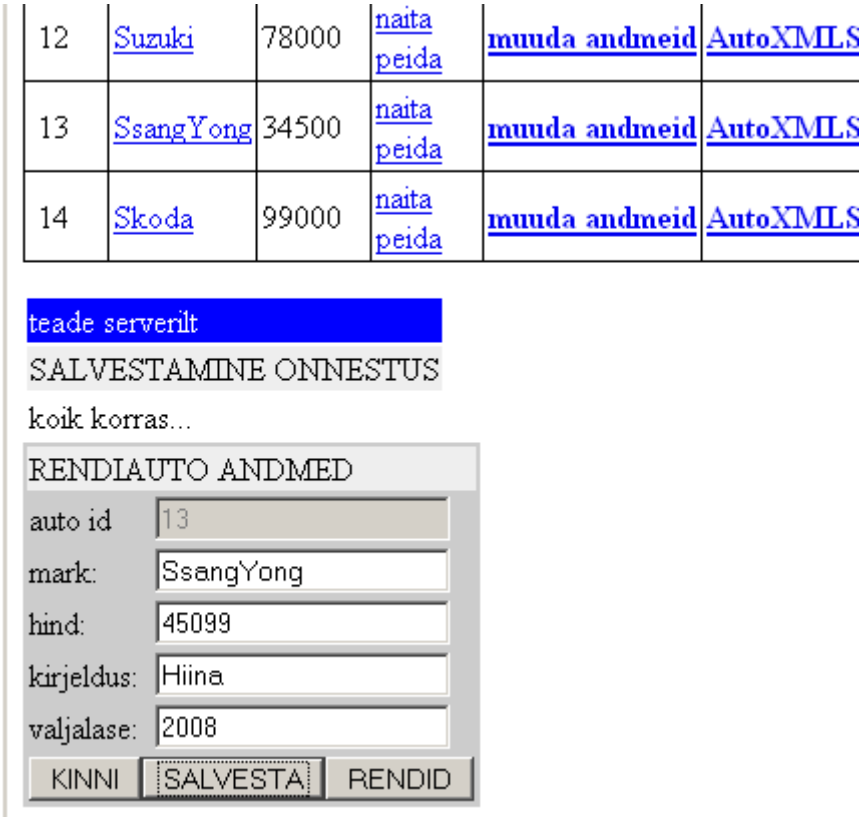
2.Veebileht peale vajutamist auto nime peal (veebilehe ümberlaadimist ei toimu!muudetakse brauserisse laetud veebilehe staatust):



3. Veebileht valede andmete korral:



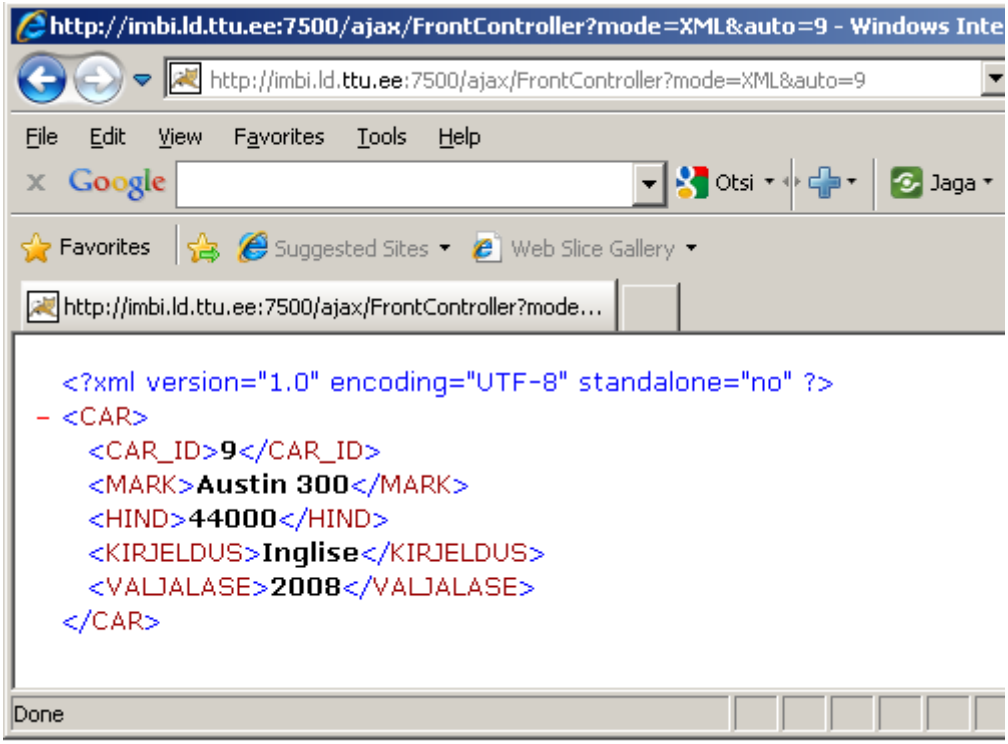
4. Veebileht siis kui andmed olid õiged:



Lisatingimused:

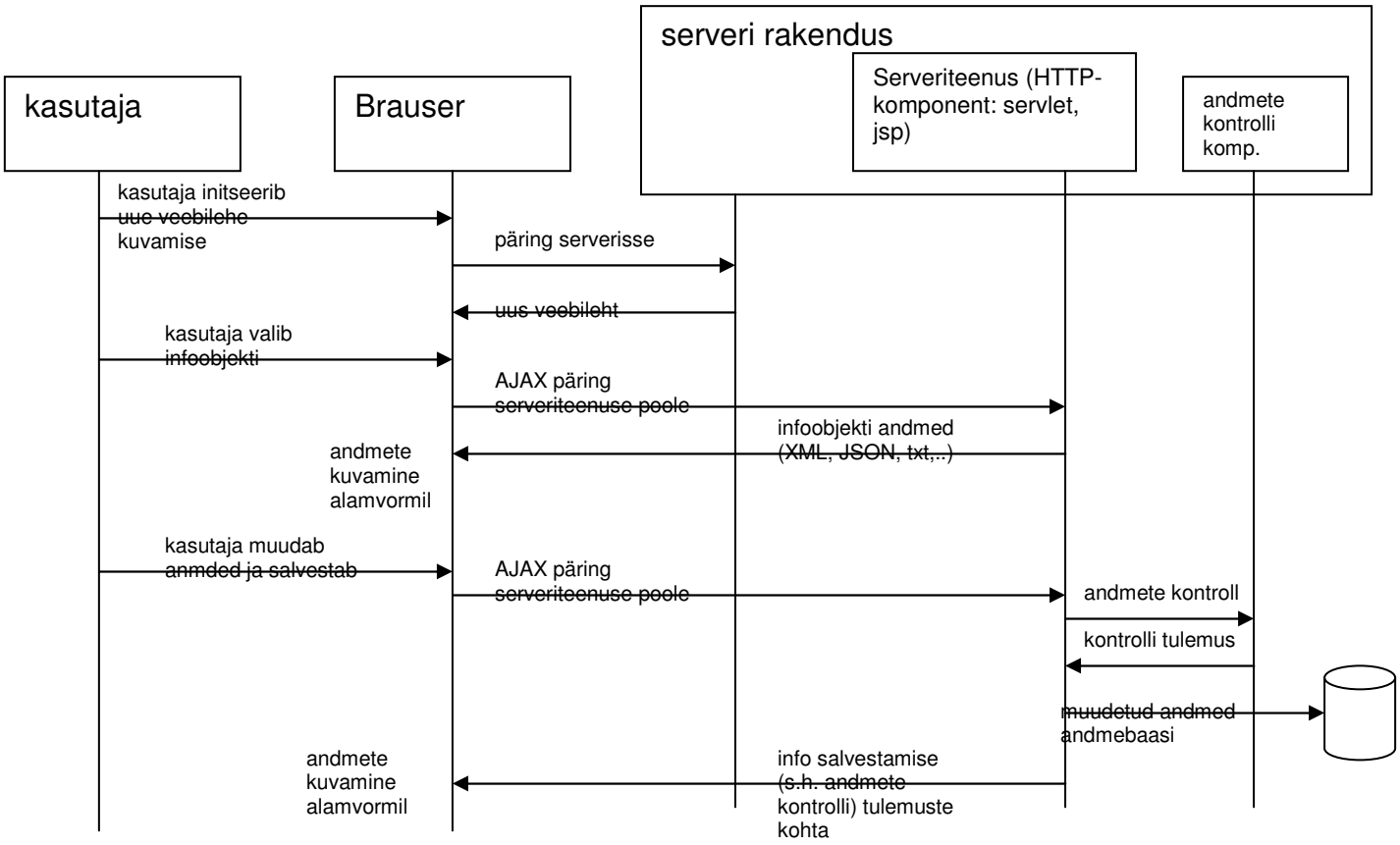
- võib vabalt kasutada AJAX raamistike ja komponente (JQuery ja teised)
- andmete formaat mida AJAX-i väljakutsetes kasutatakse on vaba (XML, tekst, JSON,..)
- rakendusega ühine turvakontekst. Kui AJAX-i väljakutseid kasutatakse sellisel veebilehel millele juurdepääsuks on vaja sisse logida siis ei tohi autentimata pöörumistele vastata ka AJAX-i pöördumisi teenindav serveri komponent (teenus). Näiteks – näites on serveriteenuse URL-iks mis väljastab brauserile auto andmeid järgmine aadress:

http://imbi.ld.ttu.ee:7500/ajax/FrontController?mode=XML&auto=9



Sellel URL-il asuv teenus ei tohi andmeid saata brauserisse kui kasutaja on veebirakendusse sisse logimata (teeb näiteks brauseri lahti ja „kleebib“ aadressiribale teenuse aadressi.)

Jadadiagramm AJAX-i alamülesande kohta.



11. JUniti kasutamine ühe komponendi (klassi) testimiseks.Kahe vabalt valitud rakenduse klassi testimine.

nõue 11.13. Kirjutage üks käsurealt käivitatav test kasutades **JUnit** testimis-raamistikku.

Tutvuge JUnit nimelise testimise raamistikuga - http://junit.org/
http://www.linux.ie/articles/tutorials/junit.php

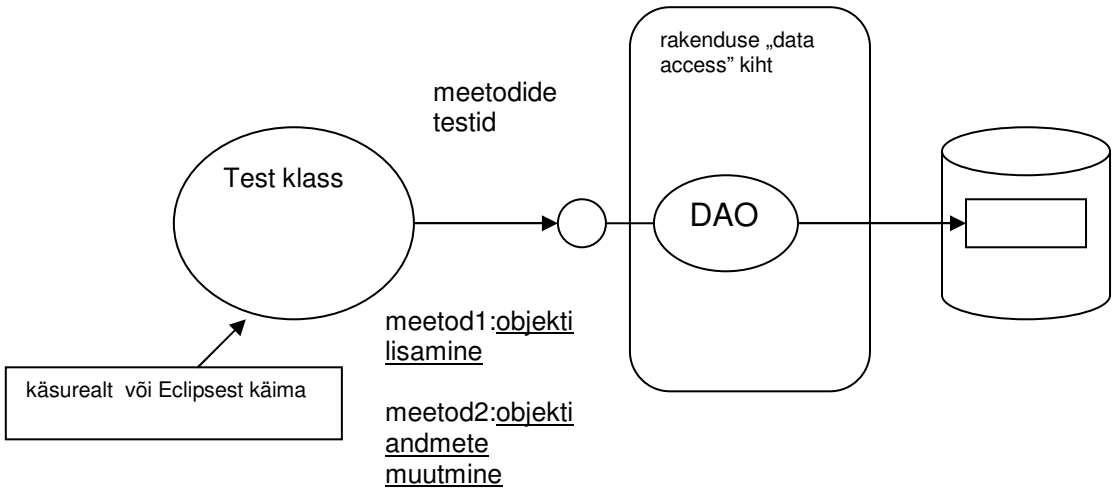
NB! Kui Te tahate kasutada mingit muud Java testimise raamistikku siis võite seda ka teha. Junit on lihtsalt üks levinumaid ja sellepärast temast ka siisn juttu tehakse.

Valige välja suvalised klassi oma rakendusest ja kirjutage nendele klassidele testklass („testcase") mis testib vähemalt kahte antud klassi meetodit (kui klassil on kaks või rohkem meetodit, kui on ainult üks meetod siis saate testida muidugi ka ainult ühte meetodit).

Veenduge et teie JUnit-i testid töötavad ka praktikatööde serveris imbi.ld.ttu.ee , eksamil peaks olema võimalik neid selles serveris käivitada (v.a. erijuhud kus rakendus ei ole imbi.ld.ttu.ee serveris või rakenduse tehnoloogia ei võimalda JUniti kasutamist).

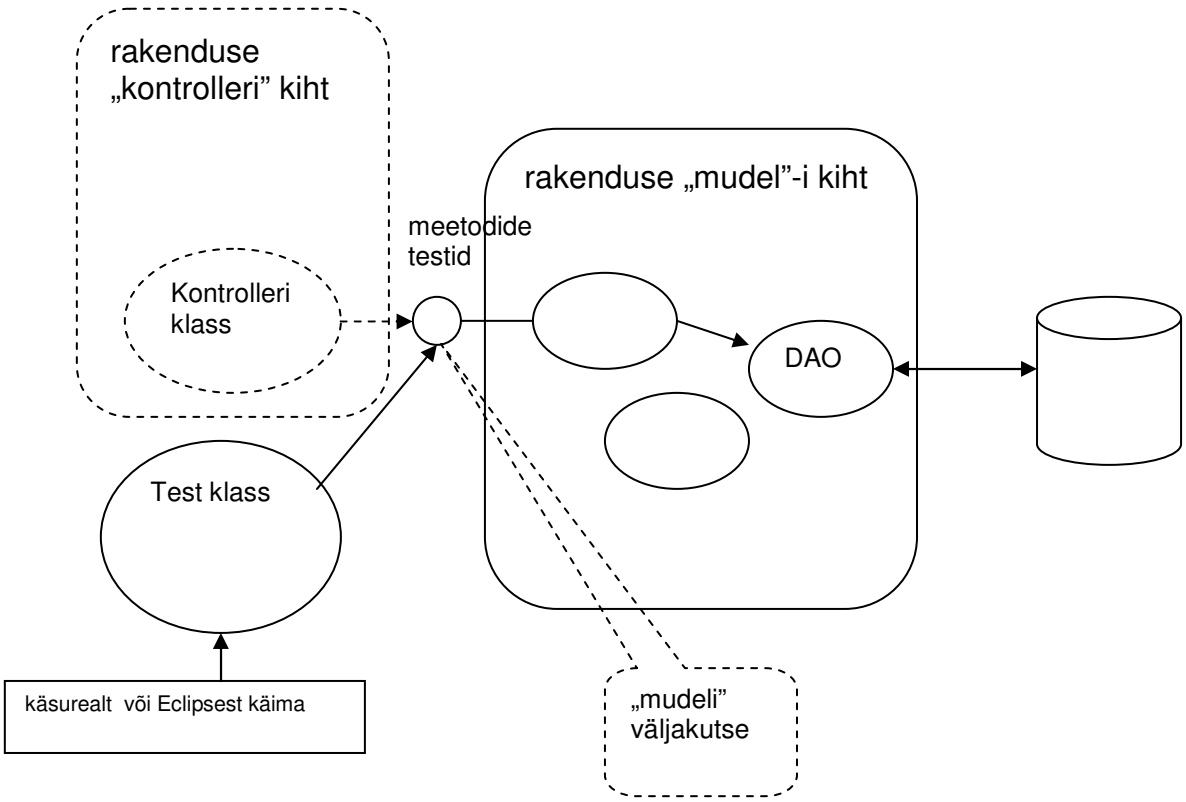
Kes teeb rakendust mitte-Java platvormil – katsuge leida JUnit-ile analoogne vahend mis sobib teie platvormil kasutamiseks ja kirjutage testid

Kõiki rakenduse klasse käsurealt testida ei saa (õigemini – see on keeruline). Näiteks servlet-klassid peavad töötama rakendusserveris (Tomcat-is), neid käsurealt käivitada on keeruline. Kõige mõistlikum ja reaalselt ka kõige kasulikum (vajalikum) on testida näiteks DAO-klasse.



Joonis 1. DAO integratsioontest

Sellist testi nimetatakse integratsioonitestiks – siin me ei testi ainult DAO toimimist vaid DAO toimimist koos andmebaasi serveriga.



Joonis 2. „mudeli" väljakutse, integratsioonitest

Teine hea komponent (klass) testimiseks on see komponent mis asub „mudeli" kihis ja mida kutsutakse välja rakenduse kontrolleri kihi poolt (mida see tähendab see saab selgeks siis kui oleme üle vaadanud MVC arhitektuuri olemuse). Väga lihtsa arhitektuuriga rakenduse puhul võib selleks „mudeli" komponendiks olla ka DAO komponent , sellisel juhul on skeem sama mis joonisel 1. ("DAO integratsioonitest"). Reeglina kontrolleri kihist siiski DAO-sid otse välja ei kutsuta, loengutes ja harjutustes räägime ka sellest miks nii ei tehta.

Põhimõtteliselt joonise 2 puhul asendatakse kontrolleri klass (mis tegelikult kutsub „mudeli" komponentide mingeid meetodeid välja) testklassiga.

Nüüd on tegemist ka integratsioonitestiga aga sellest testist osa võttev komponentide jada on pikem ja testi käigus testitav rakenduse osa on suurem, testi käigus toimuv

komponentidevaheline interaktsioon keerulisem. Põhimõtteliselt testite joonise 2. puhul oma rakenduse „back-end" osa , olles asendanud „front-end" osa testiga.

12. Veasituatsioonide logimine rakenduses.Logifailide soovitava asukoht.

nõue 12.14. Logifaili olemasolu

Veasituatsioonide tekkimisel (kõige konkreetsemaks näiteks andmebaasiserveri „maasolek" – seega ei ole siin mõeldud kasutaja poolt sisestatud vigaste andmete situatsiooni) rakenduses tuleks need veasituatsiooni kirjutada logifaili. Logifailis peaks sisalduma võimalikult detailne info vea tekkimise asukohast, ajast ja vea olemusest). Logimiseks võite kasutada **log4j** teeki (seda kasutavad ka harjutustundide näited) või mingit muud lahendust.

uuendus 18.03.2013 !

Logifaili asukoht võiks olla selline et seda faili sisu oleks võimalik veebibrauseris näha. Samas ei ole hea kui see logifail on teie rakenduse veebikataloogisTomcat-i serveril, nagu näiteks:

http://imbi.ld.ttu.ee:7500/t644231/log.txt

ehk serveri failisüsteemist vaadates

/usr/local/jakarta/webapps/t644231/log.txt

Asi on selles et seda logifaili kirjutatakse Tomcat-i serveri poolt ja rakendust üles pannud kasutajal ei pruugi olla võimalik rakendust serverilt maha võtta failiõiguste tõttu – logifaili ei saa kustutada. Sellepärast on soovitus määrata oma rakenduste logide kataloog Apache HTTP veebserveri kataloogi:

/usr/local/apache2/htdocs/tomcat_webapp_logs

Näiteks rakenduse **JSP_servlet_bean_maven** logifaili asukoha määramine failis log4j.properties:

log4j.appender.A2.File=**/usr/local/apache2/htdocs/tomcat_webapp_logs**/JSP_servlet_bean_maven/log.txt

Tomcat-i rakenduste veebi-logide asukoha juurkataloog Apache serveril

Teie rakenduse alamkataloog.Seda pole vaja ise teha, Tomcat teeb siis kui hakkab uut logifaili kirjutama.

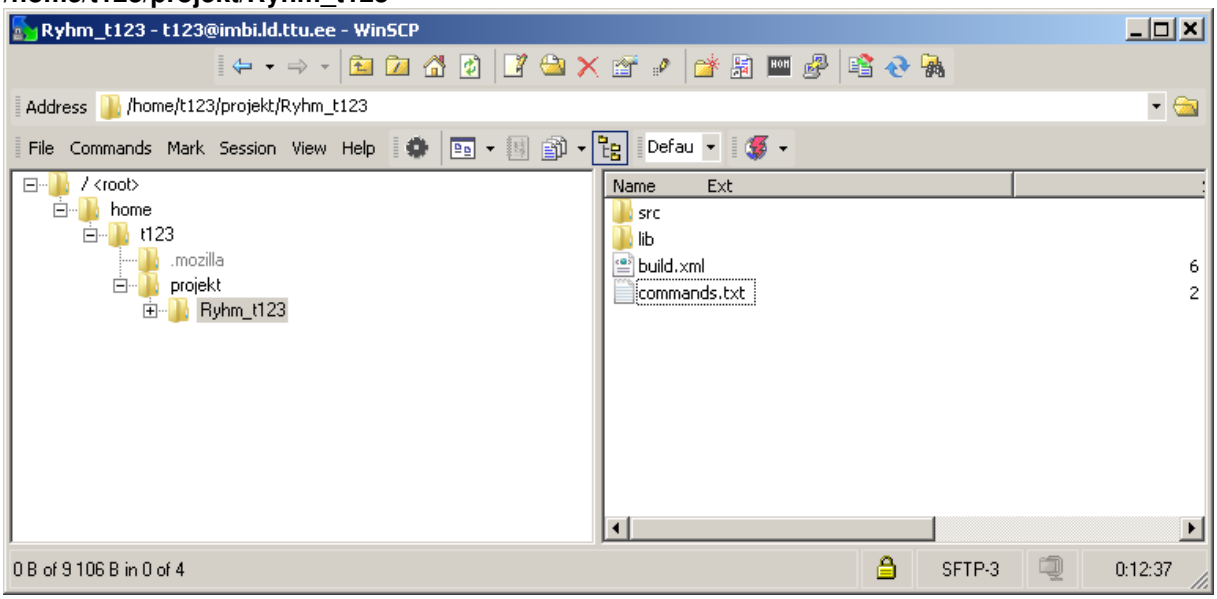
13. Rakenduse kompileerimine ja installeerimine käsurealt TTÜ serveris.

nõue 13.15. Kaitsmise ajal peab rakenduse lähtekood peab olema kopeeritud TTÜ serverisse tudengi kodukataloogi (vabal valikul imbi.ld.ttu.ee või hektor4.ttu.ee) ja peab olema võimalik seda rakendust linuxi käsureal uuesti kompileerida ja serverile installeerida.

Võite kasutada selleks näiteks Apache Ant-i või Maven-it kasutades ja kohandades näidetes toodud build.xml või pom.xml faile vastavaks oma rakendusele.

Näiteks nii:
Töö on kopeeritud imbi.ld.ttu.ee serverisse kasutaja t123 kodukataloogi.

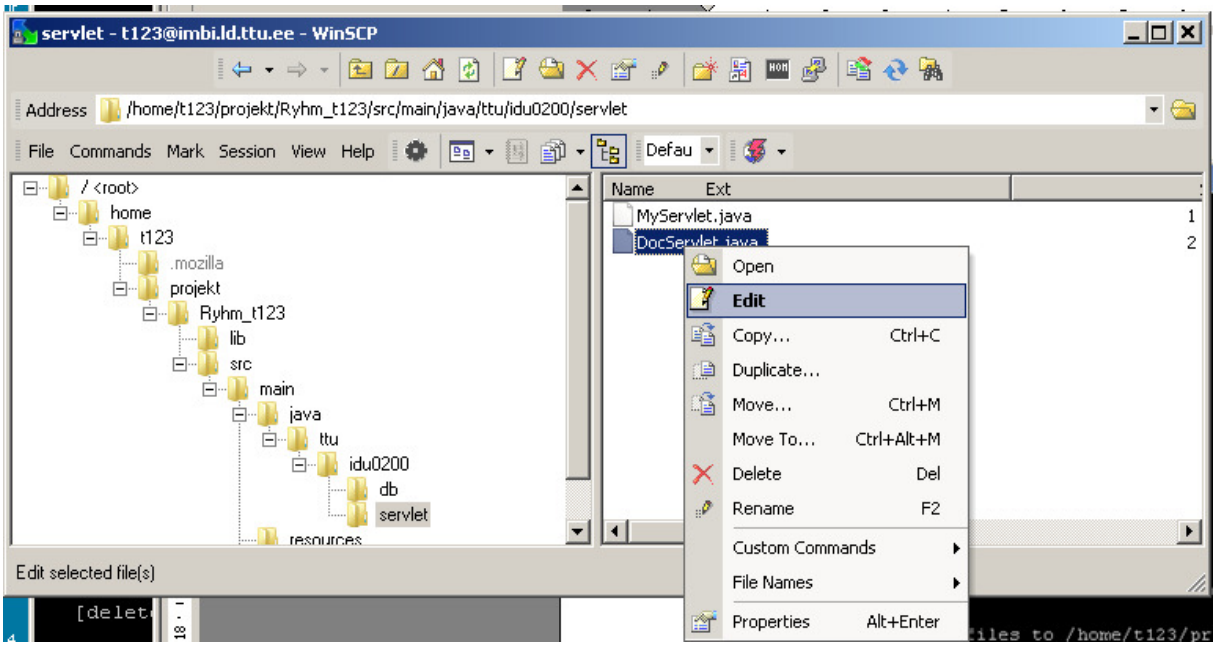
/home/t123/projekt/Ryhm_t123



cd /home/t123/projekt/Ryhm_t123
ant deploy_war_local

```
t123@imbi:~/projekt/Ryhm_t123
[t123@imbi Ryhm_t123]$ cd /home/t123/projekt/Ryhm_t123
[t123@imbi Ryhm_t123]$ ant deploy_war_local
Buildfile: /home/t123/projekt/Ryhm_t123/build.xml

init:

clean:
   [delete] Deleting directory /home/t123/projekt/Ryhm_t123/WEB-INF/classes
   [delete] Deleting directory /home/t123/projekt/Ryhm_t123/WEB-INF

prepare:
    [mkdir] Created dir: /home/t123/projekt/Ryhm_t123/WEB-INF/classes

compile:
    [javac] /home/t123/projekt/Ryhm_t123/build.xml:78: warning: 'includeantrunti
me' was not set, defaulting to build.sysclasspath=last; set to false for repeata
ble builds
    [javac] Compiling 4 source files to /home/t123/projekt/Ryhm_t123/WEB-INF/cla
sses
     [copy] Copying 5 files to /home/t123/projekt/Ryhm_t123/WEB-INF/classes

war:
      [war] Building war: /home/t123/projekt/Ryhm_t123/Ryhm_t123.war

deploy_war_local:
     [copy] Copying 2 files to /usr/local/jakarta/webapps
   [delete] Deleting directory /home/t123/projekt/Ryhm_t123/WEB-INF/classes
   [delete] Deleting directory /home/t123/projekt/Ryhm_t123/WEB-INF

BUILD SUCCESSFUL
Total time: 2 seconds
[t123@imbi Ryhm_t123]$
```

Eksamil töö kaitsmise ajal võib tekkida ka vajadus lähtekoodi muuta (kuigi ilmselt teeme seda harva ja ka siis on muudatused väikesed), siis saame koodi muuta faile otse üle ssh lahti tehes (näiteks WinSCP-ga).



Kui rühmas on mitu inimest siis peab rakendus olema kopeeritud ainult ühe rühma liikme kodukataloogi.

14. Rakenduse lähtekoodi kommentaarides klasside ja meetodite autori kirjapanemine.

nõue 14.16. Kirjutage lähtekoodi nende klasside kommentaaridesse autori nimi.
Kui olete mingit klassi teinud koos nii et autorsust on raske eristada siis kirjutage autoreid mitu.

Kui ühes klassis on meetoditel erinevad autorid siis kirjutage autor-i kohta kommentaar vastava meetodi algusse.

```
package idu0020.web;
/**
 * @author Martin Jalakas
 * @version 1.0  12/03/2013
 * IDU0020
 */
```

.

Kui olete märgitud klassi või meetodi autoriks siis on eksamil küllaltki loogiline oodata et peaksite oskama seda klassi või meetodit kommenteerida ilma väga pika järele-mõtlemise ajata

15. Dokumentatsioonist

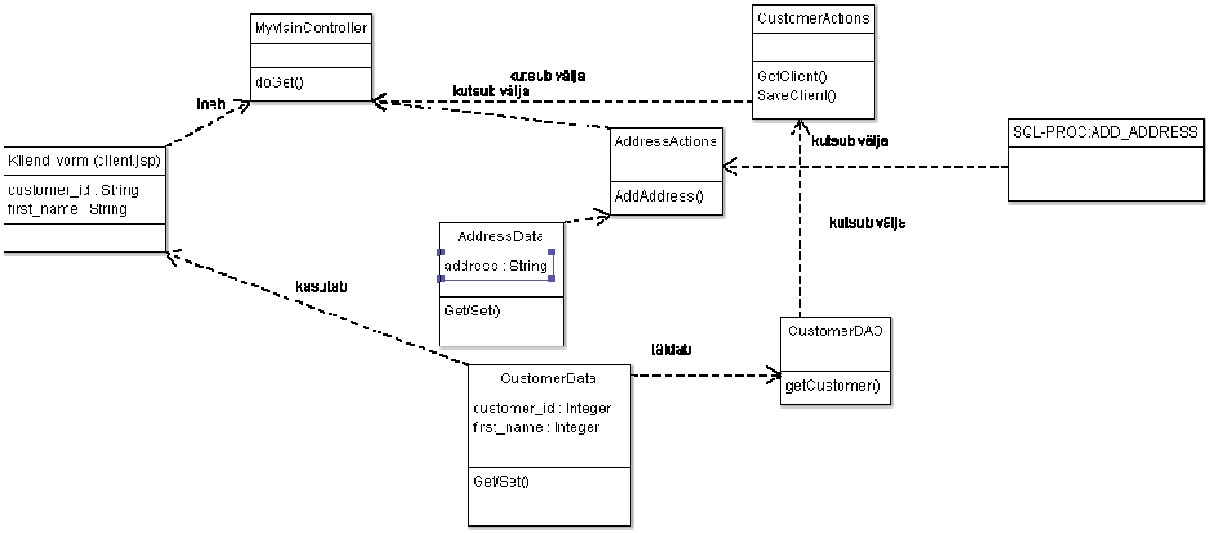
Mis oleks vaja esitada kirjalikult (see tähendab failis, mitte väljatrükituna).

nõue 15.17. Vabalt valitud vahendiga (ArgoUML, Rational, MS Visio) kandke klassidiagrammidele kõik teie poolt loodud komponendid – klassid, JSP-lehed, XSL-lehed,SQL-protseduurid. Kuigi vormistuseks on klassidiagramm tuleks nendlee diagrammidel kanda kõik komponendid – ka selliseid mis ei ole vormistatud kui klassid (JSP-lehed , SQL-protseduurid vms.)

NB! Selgitusi ei ole vaja juurde kirjutada , piisab ainult diagrammidest – kuna töö kaitsmine eksamil on suuline siis on ainult oluline et teil endal oleks meeles mis otstarvet mingi komponent täidab. Klassidiagrammidele ei ole vaja lisada klasse või komponente mis ei ole teie enda poolt kirjutatud – isegi kui need komponendid on teie komponentidega tihedalt seotud (näiteks raamistike kasutamisel).

Kui komponendid ei mahu ühele klassidiagrammile ja skeem muutub juba liiga ebaülevaatlikuks siis tuleks teha mitu klassididgarmmi mis osaldavad üksteisega tihedamalt seotud komponente. Kui klassidiagramme on mitu siis võib selguse huvides komponente/klasse nendel diagrammidel ka dubleerida.

Kasulik on peale märkida ka komponentidevahelised seosed („kes kasutab keda“, „uses“-seos UML-is)

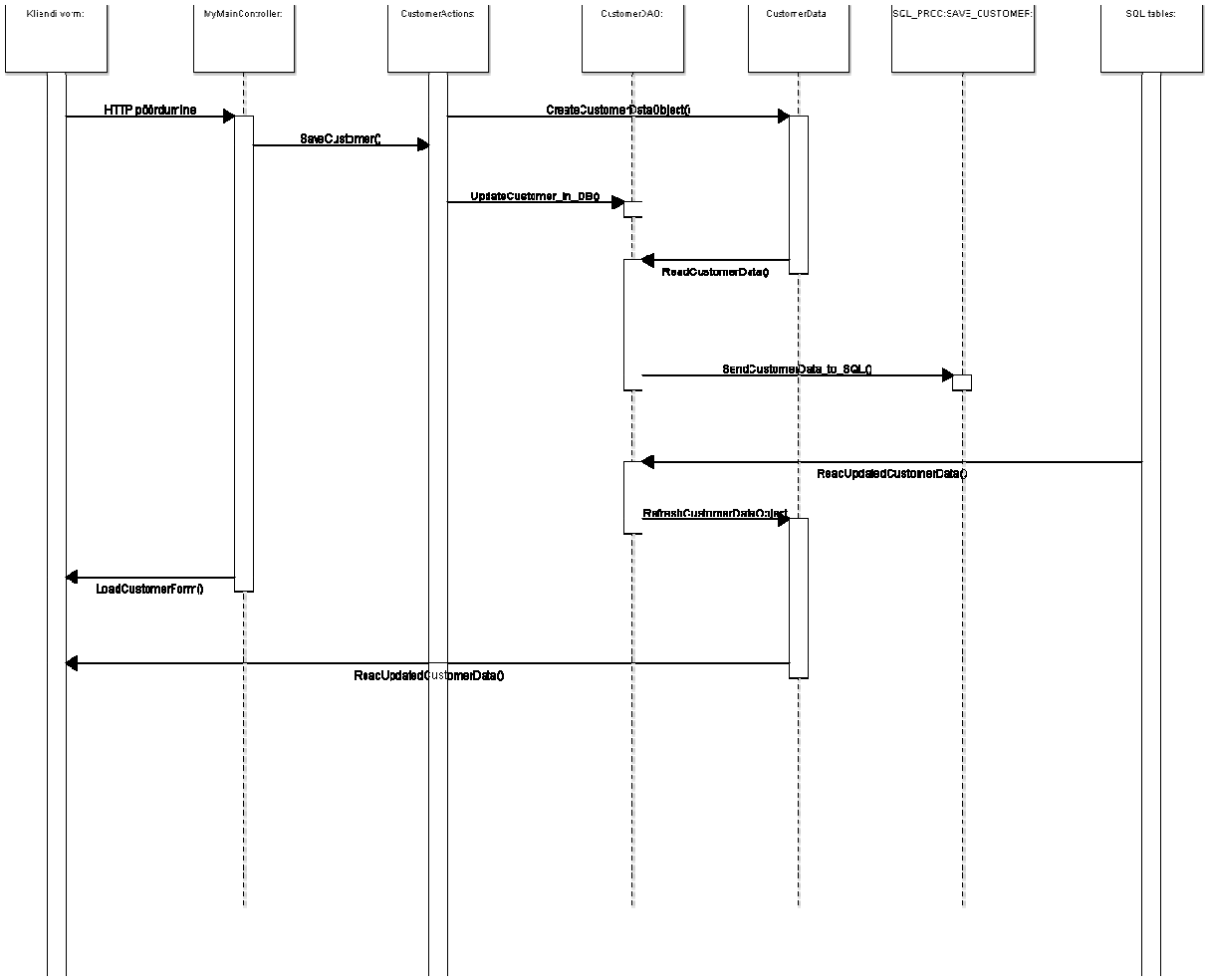


<u>nõue 15.18.</u> Kaks jada- või koostöödiagrammi.

Võtke 2 kõige (soovitavalt kõige keerukamat) terviktegevust ja tehke nende tegevuste kohta jada- või koostöödiagramm (sequence or collaboration diagram) mis kirjeldavad terviktegevust stsenaariumina ja näitavad mis komponendid millises järjekorras võtavad stsenaariumist osa.

Terviktegevuseks on näiteks:

* html-vormis olevate kliendi andmete salvestamine ja salvestustulemuste näitamine uuesti HTML-vormis – vaata allpool olevat näidet.



Selgitusi ei ole vaja skeemidele juurde kirjutada , piisab ainult diagrammidest – kuna töö kaitsmine eksamil on suuline siis on ainult oluline et teil endal oleks meeles mis otstarvet mingi komponent täidab.

nõue 15.19. Pange kirja milliseid disainimustreid te kasutasite oma rakenduses (lisaks MVC, DAO ja Composite View mustritele) ja kus teie rakenduses on neid mustreid realsieerivad osad (klassid). Võib olla teksti vormis, mõne lausega iga mustri kohta.

16. Veebiraamistike (Spring, Struts, jne. jne.) ja Javascripti raamistike kasutamisest.

Kes oskab ja omab kogemust võib vabalt kasutada rakendusraamistikke (**Struts , Velocity, Spring ja teised**) ja muud lisatarkvara mis teeb rakenduse tegemise lihtsamaks. Kui kasutate raamistikke siis teatud disaini-komponente ei ole vaja ise programmeerida – nende komponentide töö teeb ära osaliselt või tervikuna juba raamistiku tarkvara – näiteks punkt 1 (kontroller). Raamistike kasutamise puhul tuleb sellistes punktides rääkida kaitsmisel millised raamistiku komponendid, API-d,

funktsioonid, objektid realiseerivad seda osa funktsionaalsusest mida teie näiteks punktis 1. tegema ei pidanud – seega tutvustate enda kasutatud raamistikku.

NB ! Raamistike kasutamist käsitletakse selles kursuses minimaalselt ja nende kasutamist ei soovita ma algajatele – ülaltoodud tarkvara võiksid selle aine raamides kasutada pigem need kes neid raamistikke (Struts, Spring,..) juba suhteliselt hästi tunnevad – selle aine praktikatöö tegemiseks mingit raamistikku ära õppida enam ei jõua. Ja ei ole vaja ka. Raamistikud panevad rakenduse disaini juba ette suures osas paika ja seetõttu ei ole nende kasutamine meie aine eesmärkidega päris hästi kooskõlas kuigi reaalsetes tarkvaraprojektides on raamistike kasutamine kindlasti mõistlik.

Rakenduse Javascripti-osa tegemisel võib kasutada kõiki Javascripti raamistikke mille kasutamise leiate kasuliku olevat (JQuery, ExtJS, Dojo, Prototype, jne. jne. jne.)