Rakendusarhitektuurid I
Tegevuste järjekord ja näide „steps”

1. Plaan. Jaotus „backe-end”-iks ja „front-end”-iks (web-osa).

Jagame rakenduse kaheks osaks:
-1) veebiosa (pakett „web” + JSP-leht „products.jsp”).Veebirakenduse spetsiifiline osa.
Tegeleb :

- HTTP-pöördumistega
- sündmuste identifitseerimisega
- veebivormidest andmete sisselugemisega
- veebivormide andmete formaadikontrolliga
- „Model”-li väljakutsumisega („command”-klasside execute()-meetodid)
- „Model”-st saadud andmete lugemisega request-objekti
- Vaate (products.jsp) ettekerimisega

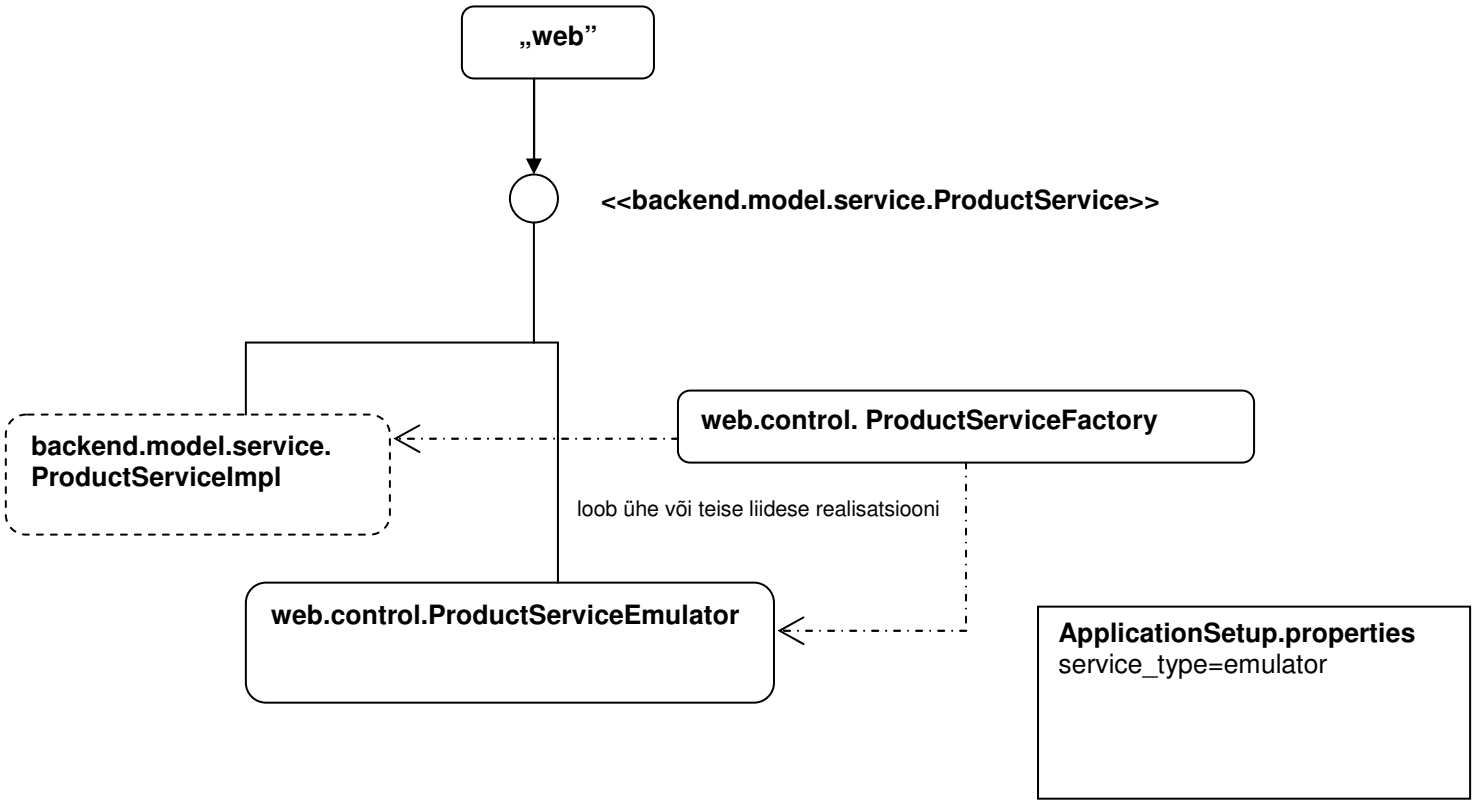
2) backend (pakett „backend”)
Tegeleb:

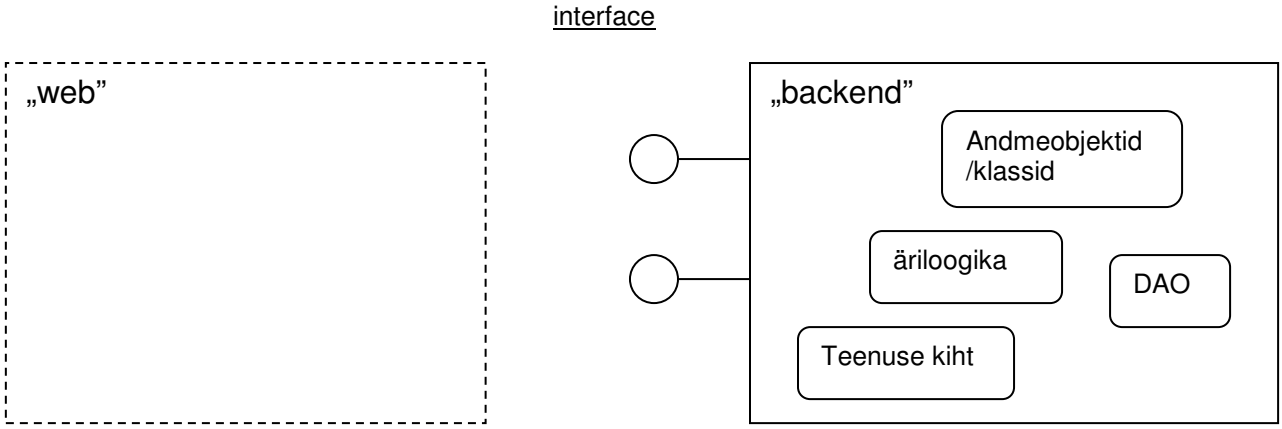
- äriloogika meetodite realiseerimisega. Bakcend-i („Modeli”) loogika on väljakutsujale kättesaadav läbi teenuskihi („service layer”) klasside meetodite (pakett **backend.model.service**).
- andmebaasiga suhtlemisega (pakett **dao**).

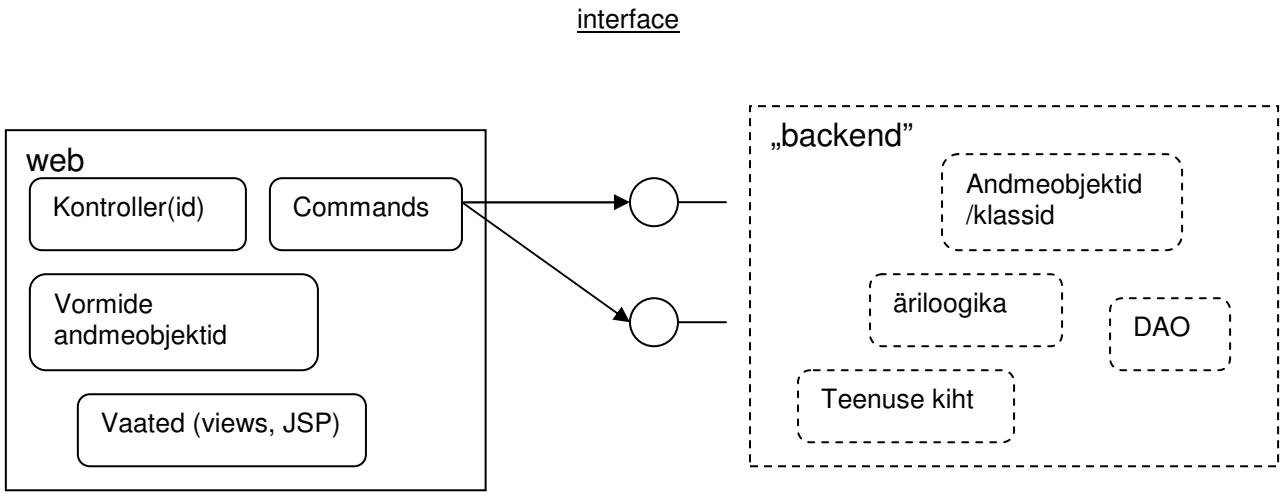
Toimimise põhimõte on selline et teeme teineteisest sõltumatult valmis „backend”-i ja „web” osa. Selleks et „web”-osa oleks võimalik „backend”-ist sõltumatult teha tuleb „web”-i osas „backend”-i emuleerida – seda teevad „Modeli” teenuskihti emuleerivad klassid **web.control. ProductCatalogTreeEmulator** ja **web.control.ProductServiceEmulator** mis realiseerivad liideseid **backend.model.service.ProductCatalogTree** ja **backend.model.service.ProductService**.

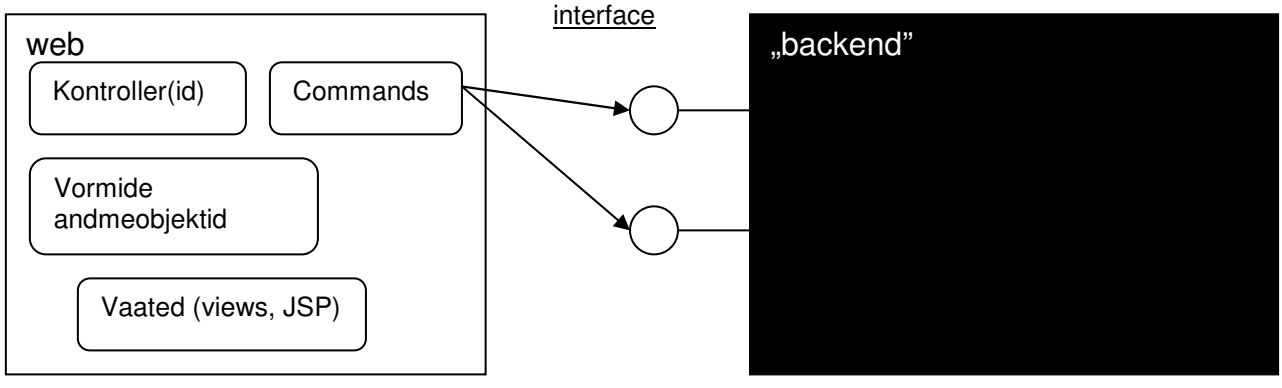
Konfiguratsioonifailis **ApplicationSetup.properties** saab vahetada parameetri service_type väärtust millega määratakse rakenduse töö ajal kas „backendina” võetakse rakenduses töölne reaalne backend-i osa või emulaator, selle faili sisu loeb teenuse-objekttide tegemise vabrik **web.control. ProductServiceFactory**.

service_type=real_backend_service
või
service_type=emulator


„web”
<<backend.model.service.ProductService>>
backend.model.service.
ProductServiceImpl
web.control. ProductServiceFactory
loob ühe või teise liidese realisatsiooni
web.control.ProductServiceEmulator
ApplicationSetup.properties
service_type=emulator

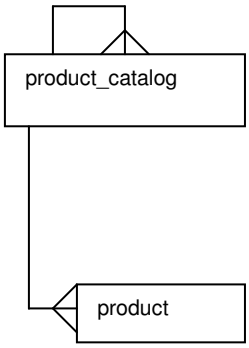

interface
„web”
„backend”
Andmeobjektid
/klassid
äriloogika
DAO
Teenuse kiht


interface
web
Kontroller(id)
Commands
Vormide
andmeobjektid
Vaated (views, JSP)
„backend”
Andmeobjektid
/klassid
äriloogika
DAO
Teenuse kiht

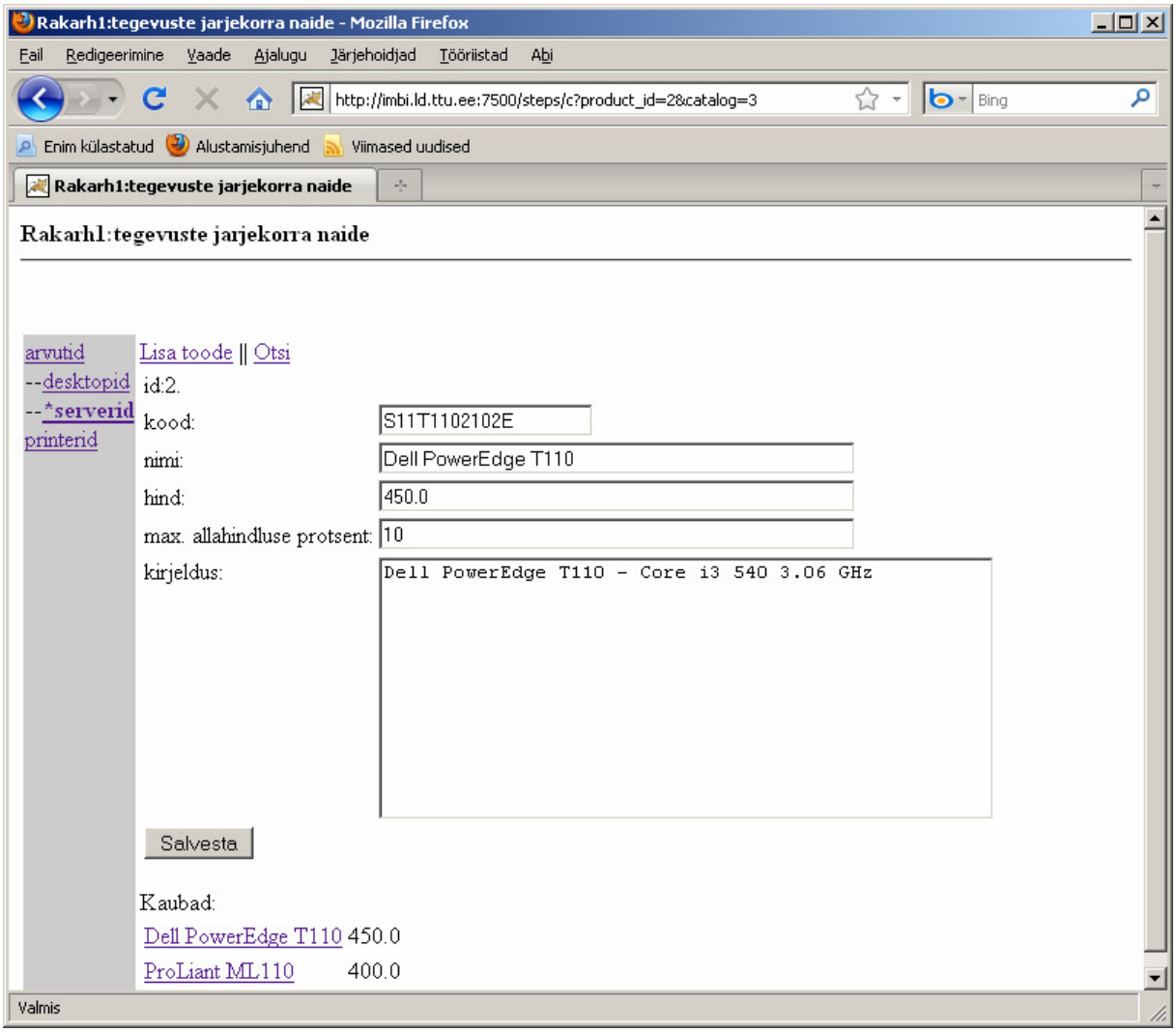

interface
web
Kontroller(id)
Commands
Vormide
andmeobjektid
Vaated (views, JSP)
„backend”

I osa: „backend”

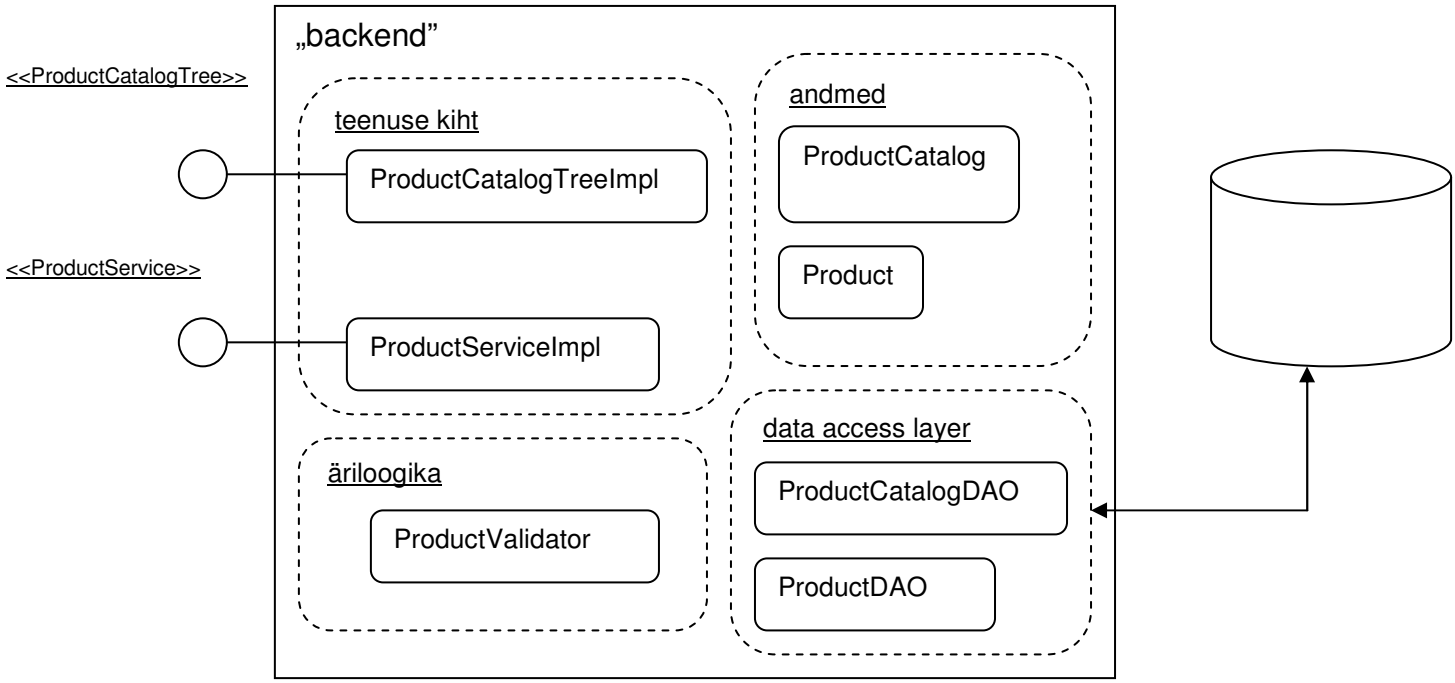
2. Tutvumine funktsionaalsuse ja andmemudeliga (andmebaasiskeemiga).



Kasulik on ka mingi ligikaudne ettekujutus rakenduse kasutajaliidesest. Aga pole vaja täpset vaadet, mingisugune ettekujutus.Mis nagunii tekib.


Rakarh1:tegevuste jarjekorra naide - Mozilla Firefox
Fail Redigeerimine Vaade Ajalugu Järjehoidjad Tööriistad Abi
http://imbi.ld.ttu.ee:7500/steps/c?product_id=2&catalog=3
Bing
Enim külastatud Alustamisjuhend Viimased uudised
Rakarh1:tegevuste jarjekorra naide
Rakarh1:tegevuste jarjekorra naide
arvutid
--desktopid
--*serverid
printerid
Lisa toode || Otsi
id:2.
kood: S11T1102102E
nimi: Dell PowerEdge T110
hind: 450.0
max. allahindluse protsent: 10
kirjeldus: Dell PowerEdge T110 - Core i3 540 3.06 GHz
Salvesta
Kaubad:
Dell PowerEdge T110 450.0
ProLiant ML110 400.0
Valmis

3. „back-end”-i disaini paikapanek.

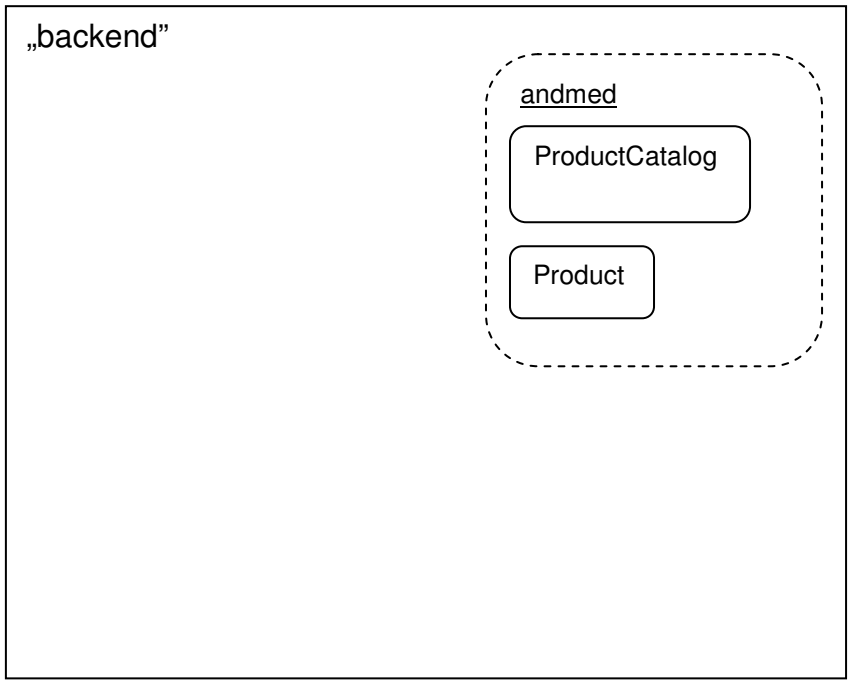


4. Rakenduse operatsioonide väljaselgitamine. Mida rakendus peab tegema andmetega.Operatsioonide andmetegevused paika.

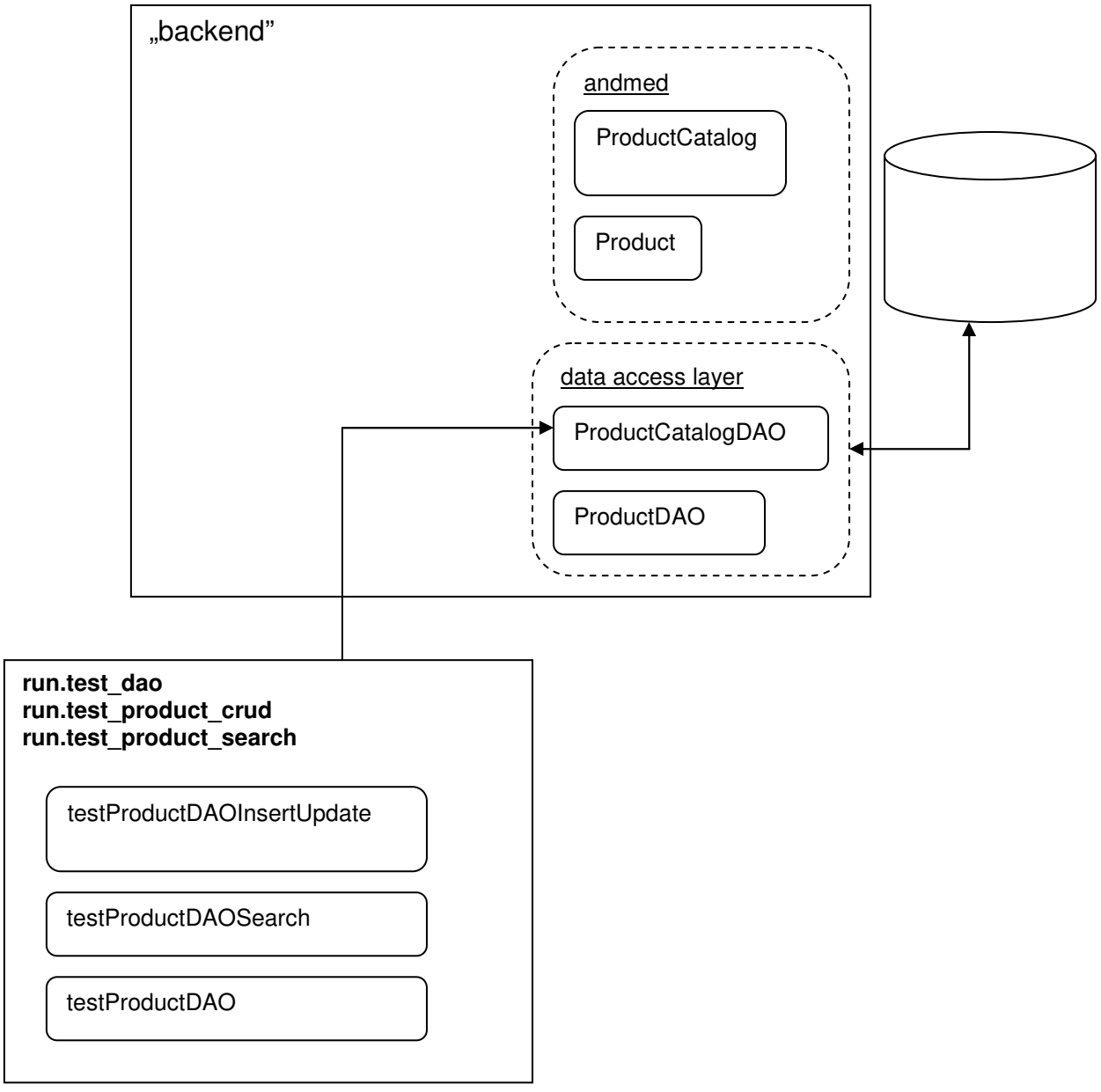
* kataloogipuu näitamine nii et üks haru on valitud
* toote lisamine
* valitud kataloogis toodete näitamine
* toodete otsing
* toote andmete muutmine

Selgeks teha kuidas on need operatsioonid seotud andmetabelitega.

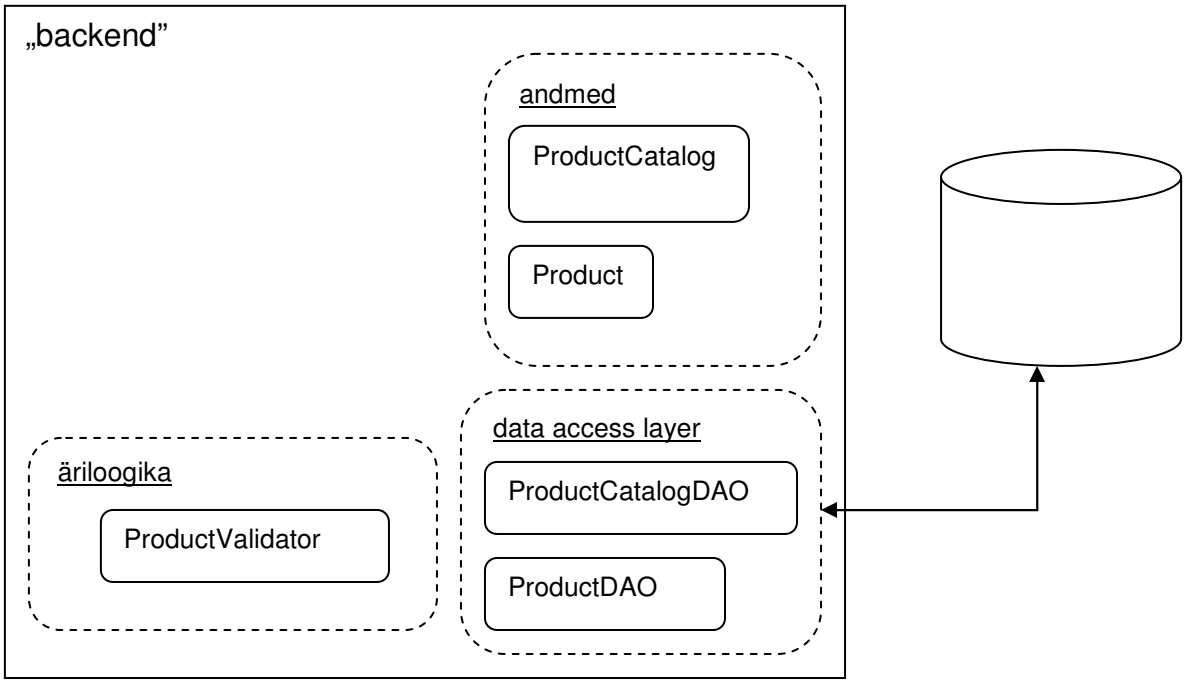
5. Andmeklasside loomine.



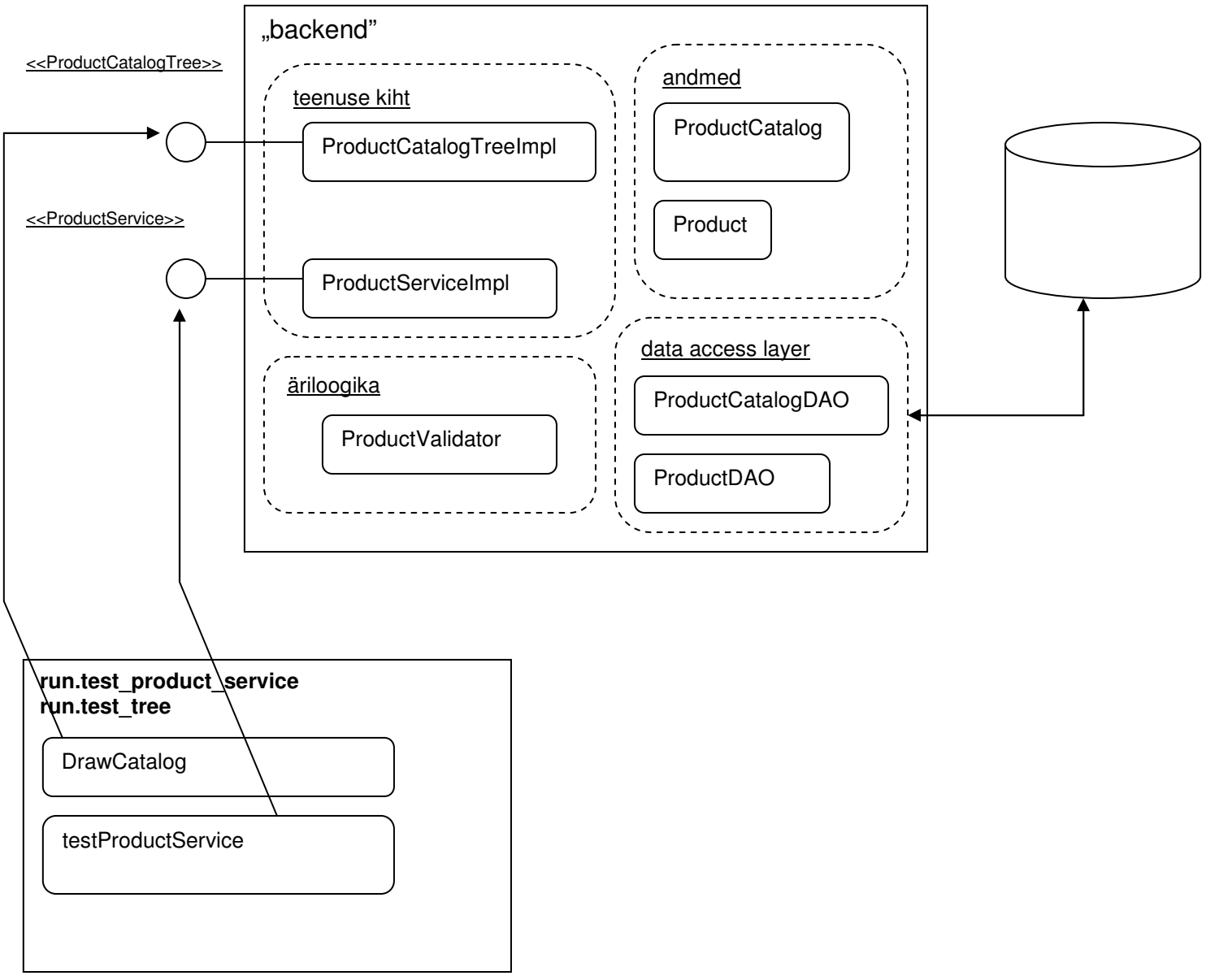
6. DAO-klasside loomine ja testimine.

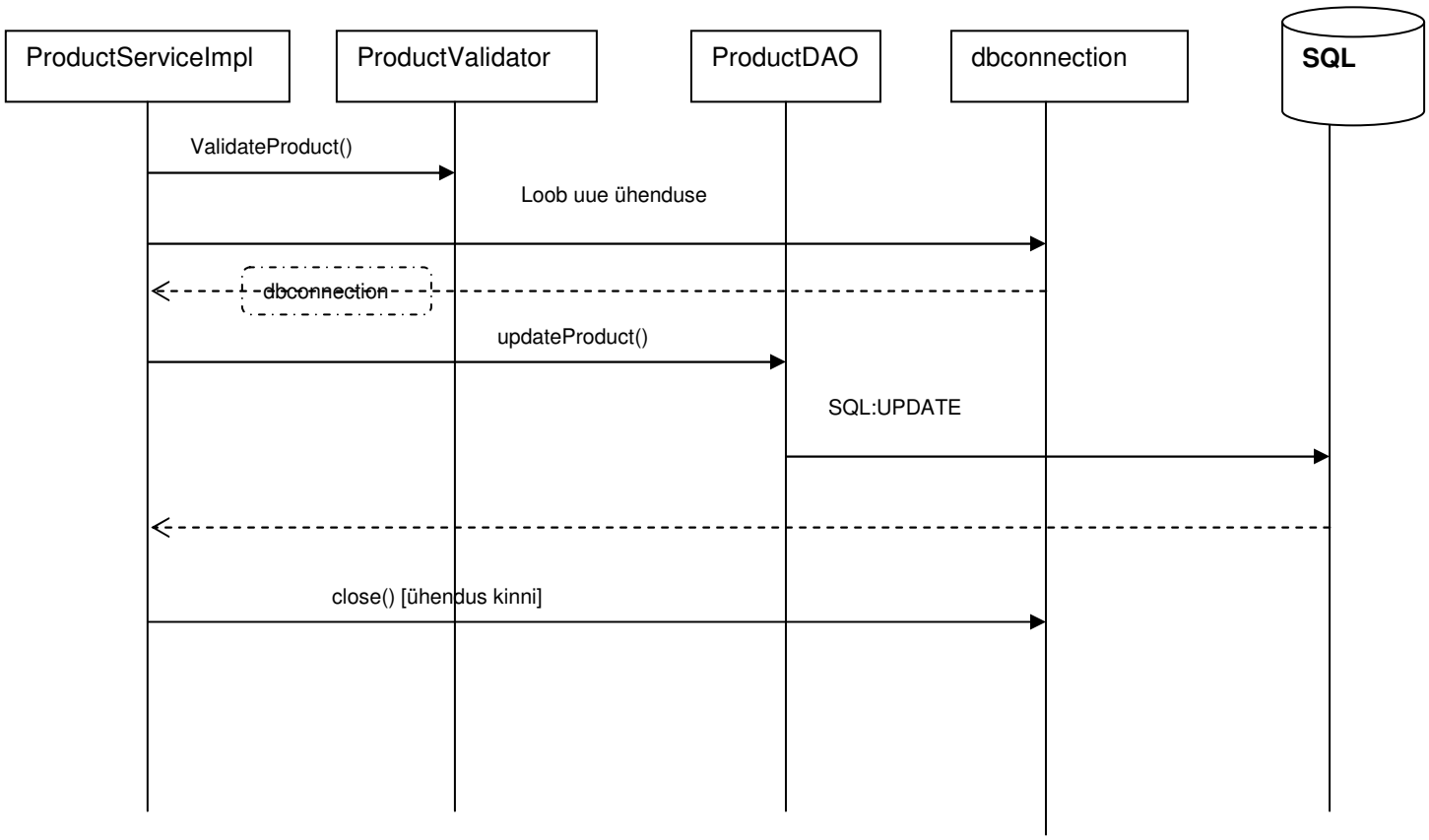


7. Äriloogika klasside loomine (validaatorid jms.)



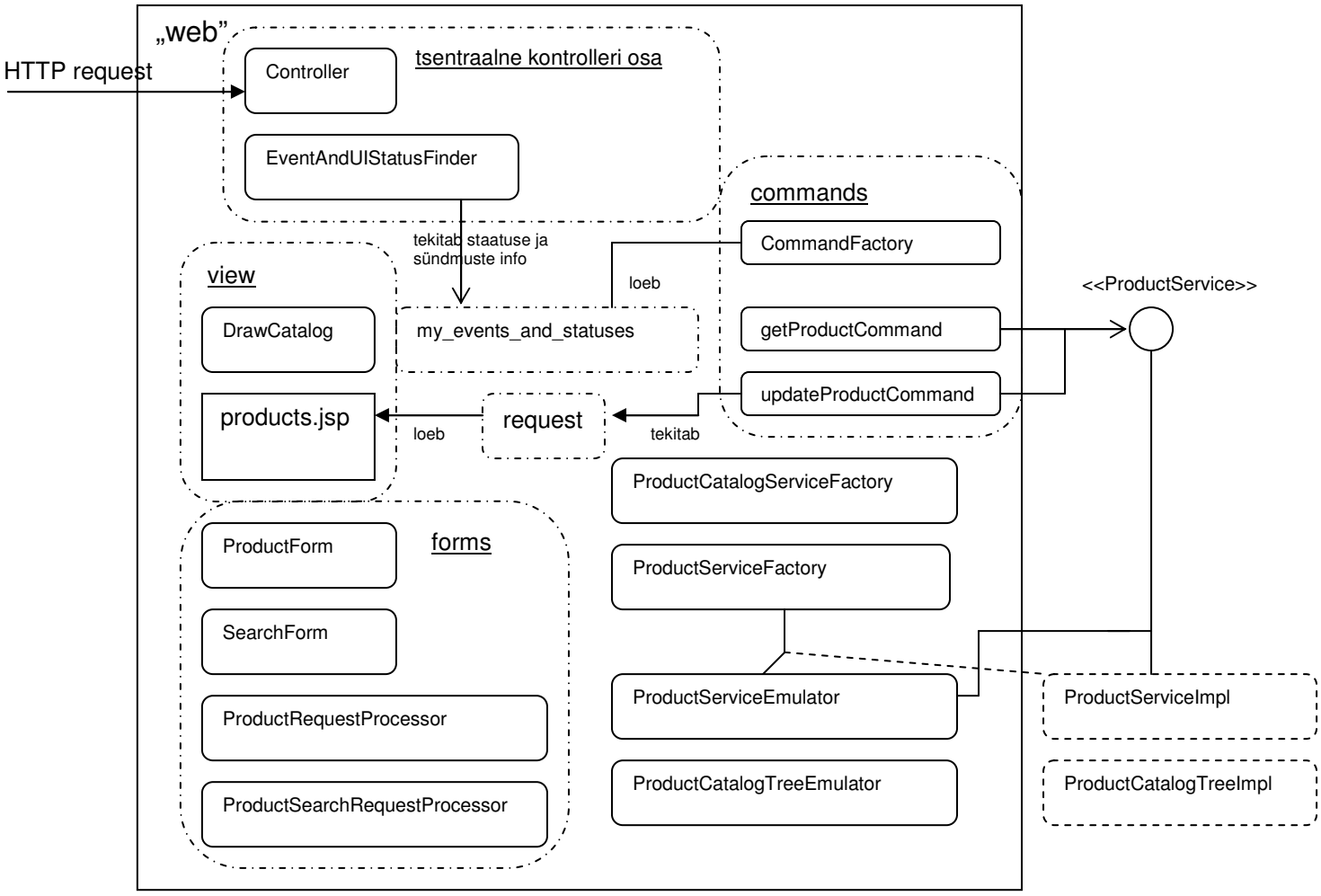
8. Mudeli teenuste kihi klasside loomine ja testimine. Mudeli teenustekihi interface-id ja realisatsioonid.




ProductServiceImpl
ProductValidator
ProductDAO
dbconnection
SQL
ValidateProduct()
Loob uue ühenduse
dbconnection
updateProduct()
SQL:UPDATE
close() [ühendus kinni]

II osa : „web”

9. Veebiosa komponendid ja tegevusplaan.



tsentraalne kontrolleri osa – „web”-osa juhtivad komponendid , käivituvad alati, iga HTTP-requesti korral. Aga kirjutame kõige viimasena.

commands – käsuobjektid konkreetsete Modeli osadega suhtlemiseks. Igale (enamasti ka kasutajaliidesel selgelt eristatavale) tegevusele vastab oma „command”.

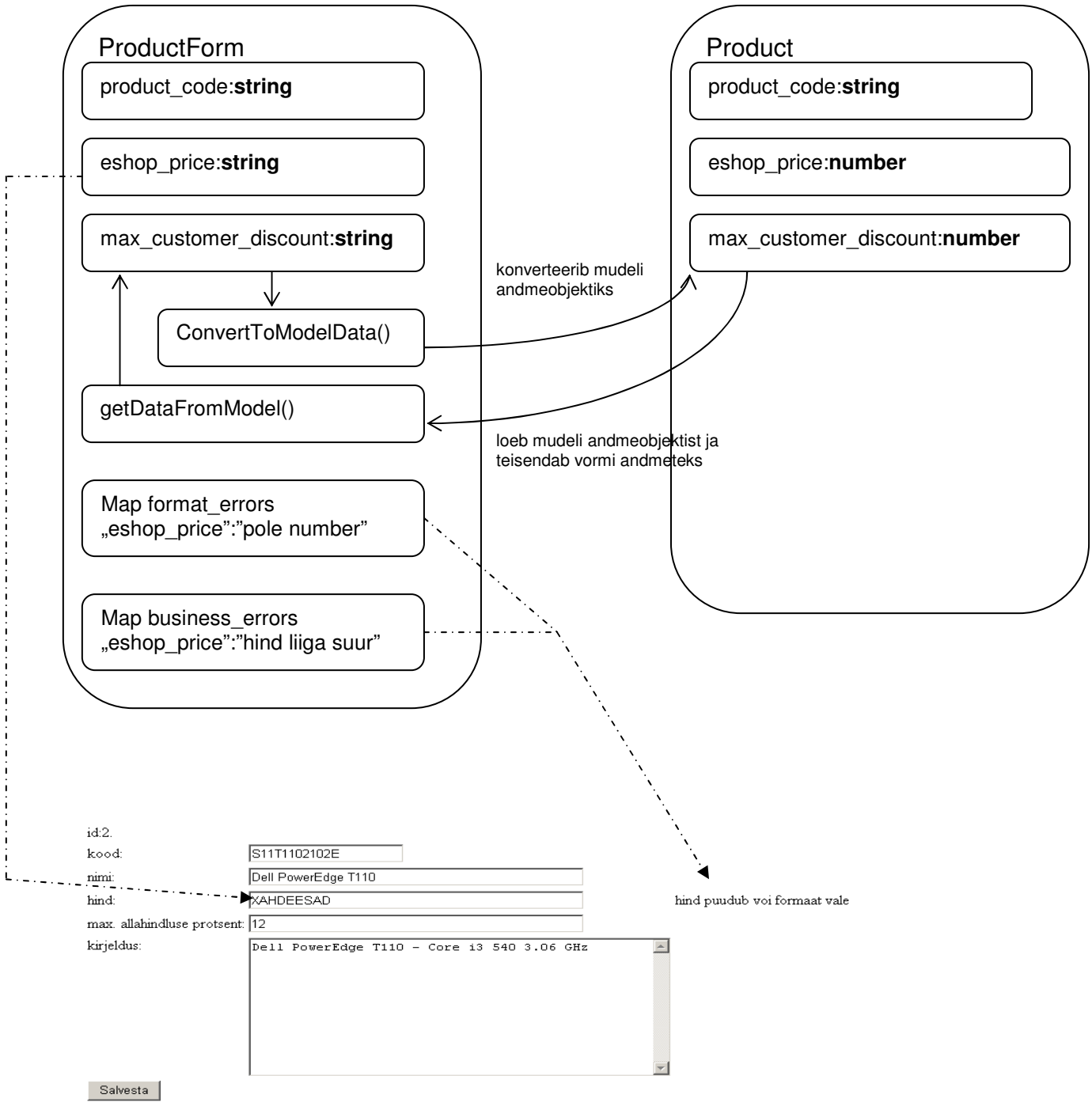
forms – andmeobjektid HTML-vormide jaoks. HTML-vormide kuvamise loogika kasutab form-tüüpi andmeobjekte.

view – andmete kuvamise loogika. Andmed võtab request-ist, vormi andmed on request-is form-tüüpi andmeobjektides.Võib sisaldada lisaks veebilehtedel eraldi klasse teatud ekraanivormi elementide „joonistamiseks”, antud näites kataloogipuu joonistamse klass **DrawCatalog**.

teenuste factory – lülitab „web”-i osa vastavalt konfiguratsiooniparameetrile reaalse „backendi” külge või emulaatori külge.

emulator-klassid – võimaldavad „web” osa teha ilma reaalse „backend”-ita. Emuleerivad (*mocking*) rakenduse Modeli-osa, täpsemalt selle Modeli osa teenuste kihti.

10. Vormi andmete töötlemine. Vormide andmeobjektid.



Vaatame ülemist kasutajaliidese pilti (mis näitab praegu situatsiooni kus kasutaja sisestas andmeid valesti) ja mõtleme mis me kasutajaliidese töötluse funktsionaalsuselt tahame:

Tahame et:

- ekraanivormide väljadel oleks võimalik hoida andmetüübist sõltumatult suvalist teksti (selles näites ei ole kasutatud Javascripti et näiteks mitte lasta ebaõiges formaadis andmeid sisestada). Kasutaja töö tuleb alles hoida, isegi kui ta formaadi/andmetüübi mõttes meie „Modeli" andmetüüpidega ei sobi.

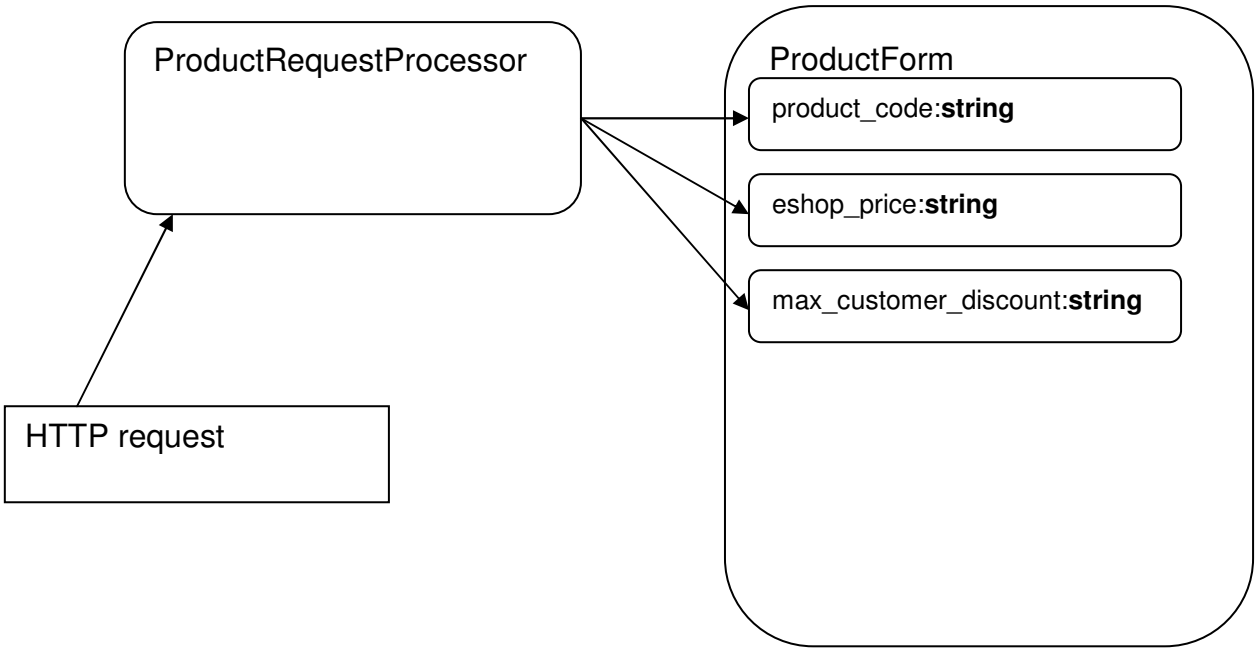
- ekraanivormidel oleks võimalik näidata andmevälja täpsusega „Modeli" poolt tagastatud veateateid ja andmeformaadiga seotud veateateid (seda viimast, formaati kontrollitakse rakenduse „web" osas).

Mõlemate eesmärkide täitmiseks (ja eeldusel et andmeid hoiame rakenduses ikka andmeobjektides, nende andmeobjekytide mingites atribuutides (mitte näiteks stringi-tüüpi massiivides) tuleb sisse tuua täiendavad andmeobjektid – **vormide andmeobjektid** – vaata paketti **web.forms**.

Vormide andmeobjektid on **veebivormi kesksed andmeobjektid** mis hoiavad veebivorm kuvamiseks vajalikku infot, sealhulgas veateadete massiive.

Need **form**-klassid on ka ainukesed klassid kus on andmed ja funktsionaalsus koos – need klassid sisaldavad meetodeid vormi andmete konverteerimiseks Modeli andmebóbjektideks ja Modeli andmetest tagasi vormi andmeteks (**ConvertToModelData()** ja **getDataFromModel()**) . Põhjus – „Modeli"ga suhtleme ikka õiges formaadis andmeobjektide abil (**backend.model.data**) kus andmetüübid on vastavuses korrektse, andmebaasi salvestatavale skeemiga).Ja „Model" loomulikult ka tagastab **backend.model.data**-tüüpi objekte mis tuleb teisendada **form**-tüüpi andmeobjektiks – sest vaates (**products.jsp**) on viited „request"-ist saadud **form**-tüüpi objektidele.

11. Andmete sisselugemine HTTP-pöördumusest vormi andmeobjektidesse. RequestProcessor-id.



Andmete lugemiseks HTTP request-ist teeme eraldi klassid – HTML vormide elementide nimed ja üldisemalt nendest vormidest andmete sisselugemise loogika võib muutuda, nii on parem see et see loogika oleks erladi, form-objektidest väljaspool.

12. Millised sündmused ja kasutajaliidese staatused on rakenduses? EventAndUIStatusFinder.

Kasutajaliidese olekud , olekute ja sündmuste hulka hoitakse staatuse/sündmuse nimeliste elementidena Map-is mis on sünmduste ja staatuste leidmise meetodi **EventAndUIStatusFinder.find()** väljundiks.

Elementide väärtusteks on kas „ui_status" või „event" aga tegelikult pole selles näites sellest vahetegemisest kasu , seda kas tegemist on (ühekordse) sündmuse või kasutajaliidese staatusega siin tegelikult ei eristata – mõlemal juhul on vaja käima lasta mingi Command.

UI staatused ja südmused selles rakenduses:

„show_products_in_catalog"	näita kataloogis olevaid tooteid
„show_product"	näita valitud toodet muutmisvormis
„search_products"	näita otsinguvormi ja (vajadusel) otsi
„new_product_form"	näita uue toote lisamise vormi
„insert_product"	sisesta uus toode andmebaasi
„update_product	muuda toote andmeid

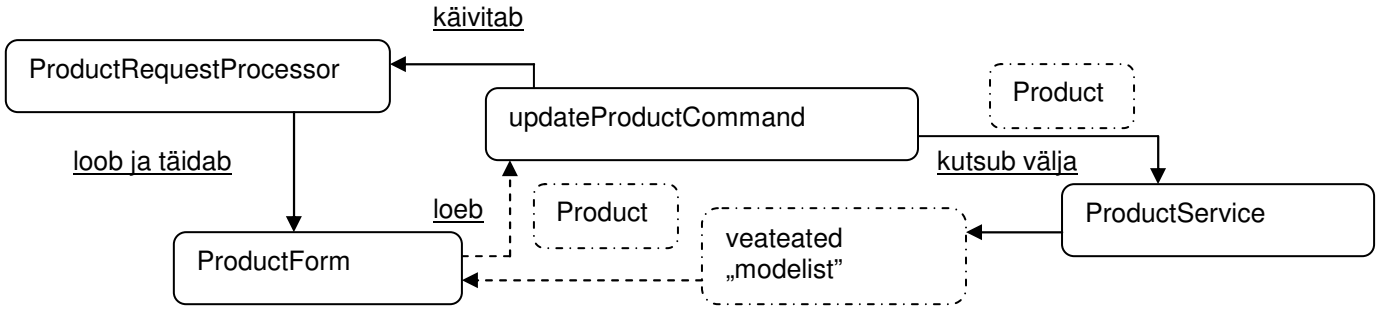
”

Lisaks on üheks staatuseks alati ka kataloogipuu näitamine aga kuna kataloogipuud tuleb selles näites nagunii alati täidata siis ei ole seda siin eraldi sündmuseks tehtud – **CommandFactory** tekitab alati vaikimisi ka kataloogipuu päringu käsu.

Nii et koos kataloogipuu näitamisega oleks meil vajadus 7 **Command**-klassi järele – rakendus teeb 7 erinevat tegevust.

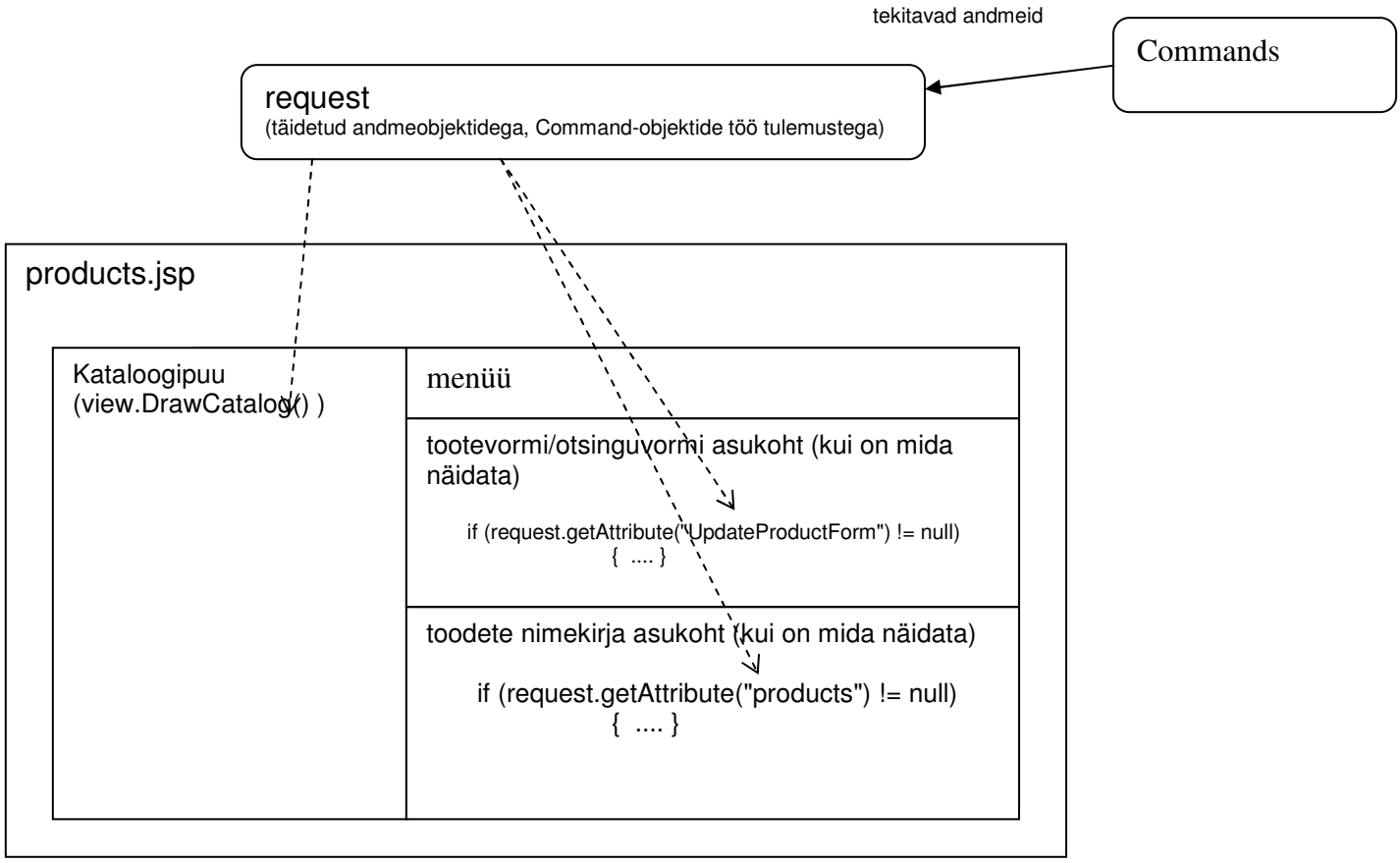
13. Käsuvabrik ja käsuklassid. (Command Factory & Commands). Käskude massiiv kui käsuvabriku töö tulemus.

Käsuvabrik **CommandFactory** võtab sisendiks **EventAndUIStatusFinder**-i poolt leitud staatuste ja sündmuste massiivi ja teeb käskude massiivi mis sisaldab just selliste staatuste/sündmuste teenindmaiseks vajalikke käske. Käskude massiiv tagastatakse **Controllerile**.



Command-klasside **execute()** meetodeid kirjutades läheb meil nüüd vaja HTTP-pöördumisest vormi andmeobjekte tekitavaid „**requestprocessor**”-klasse, vormi andmeobjektide klasse ja **ProductService** liidest. Andmete salvestamisel loetakse **ProductService** väljundist sisse „Modeli” osa poolt tagastatud andmevälja-põhised veateated ja täidetakse nendega **ProductForm**-i veateadete massiiv (kui andmed ei vastanud „Modelis” oleva validaatorite reeglitele)

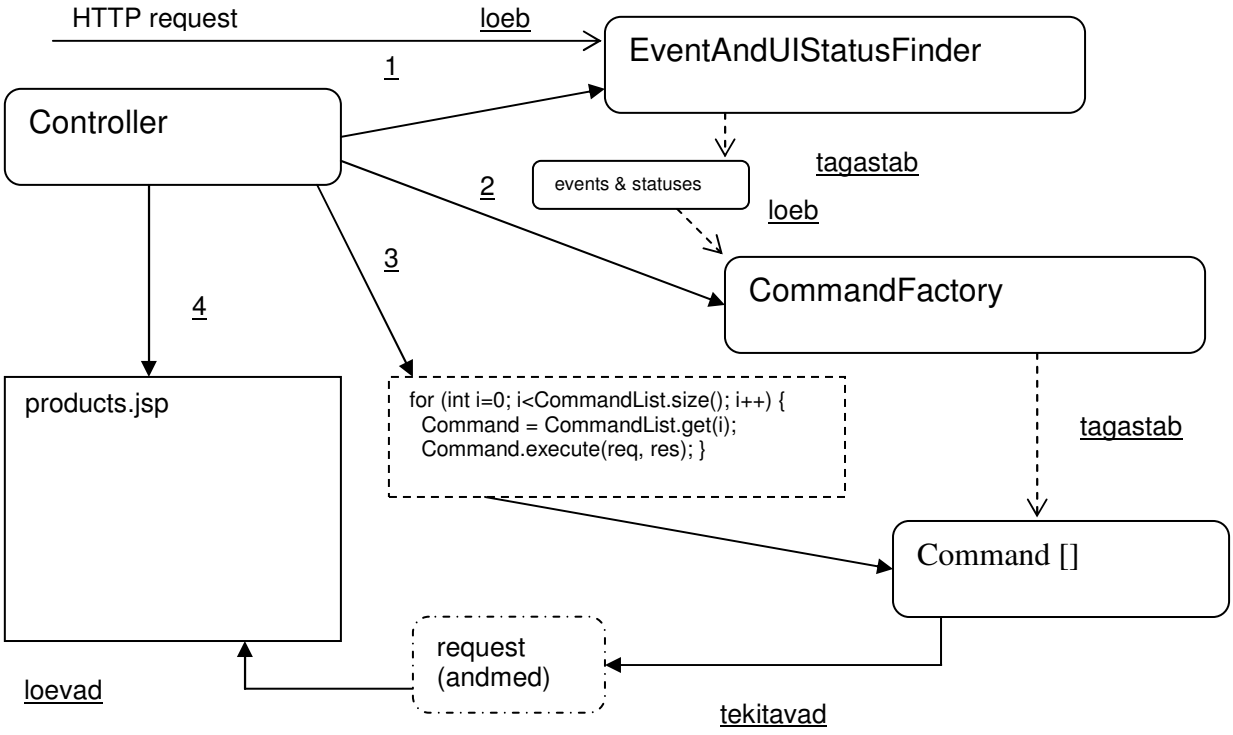
14. Vaade. Veebileht kui komposiitvaate raamistik, template, „layout”.



15. Teeme „servicefactory” klassid mis lülitavad rakenduse vastavalt vajadusele reaalsele backendile või emulaatorile.

Vaata selle materjali kõige esimest skeemi.

16. Paneme kõik kokku. Teeme „Controlleri”.



Kontroller tegutseb järgmises järjekorras:

1. käivitab sündmuste ja staatuste lugeja („leidja”) mis uurib **HTTP pöördumisest** millised on rakenduse kasutajaliidese staatused ja sündmmused mida tuleb serveri poolel töödelda.
2. käivitab käsu-objektide vabriku
3. käivitab kõigi käsu-objektide massiivi **Command []** käskude execute()-meetodid
4. annab tööjärje üle vaate-objektile milleks antud näites on alati **products.jsp**.