TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDK40LT

Juko Pappe 135179

TÖÖLAUA RAKENDUSE KASUTAJALIIDESE MIGREERIMINE VEEBI KASUTADES LIQUID WEB APPLICATION LÄHENEMIST

Bakalaureusetöö

Juhendaja: Deniss Kumlander

Tehnikateaduste doktor
Vanemteadur

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Juko Pappe

22.05.2016

Annotatsioon

Bakalaureusetöö keskendub tarkvara nimega „*UNIT4 Financial Performance Management*“ osade „*Import*“ ja „*Input – Financial*, *Other* & *Intercompany data*“ ümber disanimisele *Windows* vormilt veebi vormile. Antud töö eesmärk on produtseerida uued *wireframe*-id, mille alusel saaks uued kasutajaliidese vaated ehitada. Uus veebilehe disain peaks olema selline, et see vastaks “*Liquid Web Application*” loogika disainimist mõjutavale printsiibile, mille kohaselt veebileht peaks olema kohanduv erinevatele seadmetele. Veebilehe erinevatele seadmetele kohanevaks tegemiseks kasutatakse *Responisve Web Design* printsiipe.

Töö tähtsamad tulemused on *wireframe*-id, mis on ehitatud varasema tarkvara alusel. Disainitavast tarkvarast on koostatud ka mudelid, mis näitavad ära tarkvara põhilise funktsionaalsuse.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 60 leheküljel, 41 peatükki, 73 joonist, 0 tabelit.

Abstract

Migrating desktop application user interface into the web using Liquid Web Application approach

This Bachelor's thesis focuses on redesigning software called “UNIT4 Financial Performance Management” from a desktop based application to a web based application using the liquid web application approach. Parts of the software that are being redesigned for migrating are “Import” and “Input – Financial, Other & Intercompany data.”

This Bachelor's thesis produces new wireframes based on the functionality of the software that is being redesigned. The new wireframe views are built the way that they are adaptable to different devices. Different wireframe views are produced for devices like phones, tablets and computers. The Bachelor's thesis also produces mudels based on the “Unit 4 Consolidation & Cash” software parts “Import” and “Input – Financial, Other & Intercompany data.” These models are built to better redesign the software.

Different web desing principles are used to redesign the software. Different princibles like the so called three click rule or some of the Gestalt princibles of design. While redesigning principles are used that will support Responsive Web Design for the purpose of realizing design that would support liquid web application technology.

The princibles that are used in redesigning are described in the theory chapter. The theory chapter also contains general descriptions of tools used such as wireframes. The third chapter contains some of the produced wireframes with descriptions and explanations why the wireframe views are the way they are.

The thesis is in Estonian and contains 60 pages of text, 41 chapters, 73 figures, 0 tables.

Lühendite ja mõistete sõnastik

App	Aplikatsioon
ASP.NET	Raamistik veebilehtede ehitamiseks kasutades *HTML*-i, *CSS*-i ja *JavaScript*-i
Bootstrap	*HTML*-i, JavaScript-i ja *CSS*-i teekide raamistik
Browser	Ülemaailmsest veebist informatsiooni kättesaamise ja informatsiooni üleslaadimise tarkvararakendus
CSS	*Cascading Style Sheets,* kirjeldab, kuidas *HTML* elemendid paistavad, asetsevad veebilehel, paberil või mõnes teises meedia vormis
Dropdown list	Veebilehe element, mis sisaldab endas nimekirja tegevustest, nimedest jne, mida saab valida
Enterprise rakendus	Firma *Enterprise* rakendus
Enterprise resource planning systems	Ettevõtte *Enterprise* ressursside planeerimi süsteem
FDI	*Financial Data Input,* disainitava tarkvara „UNIT4 *Financial Performance Management“* osa *Financial Data Input* nimetus
Feature	Tunnusjoon, funktsioon
FPM	*Financial Performance Management,* tarkvara „UNIT4 *Financial Performance Management“* nimetuse osa
Font	Kirjastiil
Fovea	Silma võrkkesta keskel asuv ala
Frame	Raam,kaader
Grid	Joondatud kastikeste, võrgustiku kujul asetus
HTML	*HyperText Markup Language,* hüpertekst keel veebilehtede ehitamiseks
JavaScript	Veebi ja *HTML*-i programmeerimis keel
Liquid Web Application	Vedelad, kohanduvad veebi aplikatsioonid
Media queries	Meedia päring
Modeling language	Modelleerimis keel
Model-driver architecture	Mudeli põhine arhidektuur
Model-driver development	Mudeli põhine arendus
Model-driver engineering	Mudeli põhine tehnika/konstrueerimine/projekteerimine

n-ö	Nii-öelda
Pikslid	Pildipunktid ehk pildi väikseimad kahemõõtmelised osad
Previous	Eelmine
Responsive	Reageeriv
Runtime	Jooksmis-, tööaeg
UI	*User Interface*, kasutajaliides
UX	*User Experience,* kasutaja kogemus
Web Design	Veebi disain
XML	*Extensible Markup Language,* metakeel, üldotstarbeline märgistuskeel eelkõige veebipõhistes rakendustes

Sisukord

Jooniste loetelu

1 Sissejuhatus

Antud bakalaureusetöö keskendub tarkvara nimega "*UNIT4 Financial Performance Management*" osade "*Import*" ja "*Input - Financial, Other & Intercompany data*" ümber disainimisele *Windows* vormilt veebi vormile ja produtseerida vastavad *wireframe*-id. Disainimisel tuleb luua uued vormid, nii et veebileht oleks kohanduv erinevatele seadmetele.

1.1 Taust ja probleem

Autori huvi tekkis disaini valdkonna vastu läbides Tallinna Tehnikaülikoolis erinevaid programmeerimis kursuseid. Samuti kasvatas huvi praktikal, arendades rakendust, kus oli vaja teha funktsionaalsust kasutajale arusaadavaks ja mugavaks kasutada. Lisaks tuleb disainimis printsiipide teadmine ja oskus neid kasutada abiks töötades infotehnoloogia valdkonnas.

Kuna desktopi rakendused on saadaval ainult firmades, kus vastav tehnoloogia on muretsetud, siis võib tekkida probleem, et klientidel ei ole lihtne rakendusele ligi pääseda. Veebi lahenduse korral, aga ei pea klient enda seadmesse (näiteks arvutisse) rakendust installeerima ja see on mugavam lahendus. Desktopi põhised rakendused on ka üldiselt piiratud lokaalselt ühte arvutisse või operatsiooni süsteemi. Tänapäeval, aga tuleb pidevalt kasutusele uusi telefone ja tahvelarvuteid ja me oleme liikumas maailma, kus inimestel on igapäevaselt kasutuses mitmeid arvuti/telefon tüüpi seadmeid. Seega ei pruugi olla enam piisav, kui rakendus on kasutajale liiga raskesti kättesaadav. Seega ehitades veebilehti ei ole enam piisav, kui need on sobivad ainult ühele seadmele ja seega peab ka disain olema ehitatud sobima erinevatele seadmetele.

See töö on kasulik tarkvara "*UNIT4 Financial Performance Management*" kasutajatele ja omanikele, neil on võimalus kasutada ehitatud *wireframe* ja koostatud mudeleid tarkvara vastavate osade funktsionaalsuse kohta, et ehitada enda tarkvara veebilehe vaateid. See töö on ka kasulik praktiliste näidetena õppematerjalina.

1.2 Ülesande püstitus

Antud töö eesmärk on ehitada ümber tarkvara "*UNIT4 Financial Performance Management*" osade "*Import*" ja "*Input - Financial, Other & Intercompany data*" disain *Windows* vormilt veebi vormile. Töö peaks produtseerima uued *wireframe*-id, mille alusel saaks uued vaated ehitada. Uus veebilehe disain peaks olema selline, et see vastaks "*Liquid Web Application*" loogika disainimist mõjutavale printsiibile, mille kohaselt veebileht peaks olema kohanduv erinevatele seadmetele. Bakalaureusetöö produtseerib ka mudelid (vooskeemid), mis põhinevad antud tarkvara ümberdisainitavate osade funktsionaalsusel.

1.3 Metoodika

Töös rakendatakse disainimisel mudeli põhist ehitust. Konstrueeritakse mudelid põhinedes tarkvara funktsionaalsusel ja varasemal disainil, mida rakenduses on vaja teha. Lisaks sellele rakendatakse veebidisainerite antud disainimis printsiipe *wireframe*-de ehitamisel. Printsiibid nagu näiteks Gestalti läheduse ja sarnasuse printsiibid või, et kasulik on lähtuda nii-öelda kolme klõpsu reeglist. *Wireframede* ehitamisel peab lähtuma loodud mudelitest, kust on näha, mis funktsionaalsust peab mingi vaade täitma.

"*Liquid Web Application*" loogikat aitab arendada "*Responsive Web Design.*" *"Liquid Web Application*" - selle põhiline sisu on selles, et kasutajal on võimalik töö käigus, rakendusega, vahetada seadet (näiteks arvutilt telefonile) ja jätkata sama kohapealt, kus ta töö pooleli jäi. Disain peab olema tehtud nii, et selline seadme vahetus oleks kasutajale võimalikult mugav. Kasutades "*Responsive Web Design*" printsiipe tuleb uus kasutajaliidese disain teha selline, mis suudab toetada eelnevalt välja toodud seadmete vahetamist.

1.4 Ülevaade tööst

Teises peatükis, kus on kirjanduse ülevaade tuuakse välja erinevad disainimis printsiibid, mida rakendatakse kolmandas peatükis, kus on uuringu osa. Lisaks on seal ka mõned üldised peatükid, mis seletavad lahti mõned üldised mõisted nagu näiteks *wireframe*.

Kolmanda peatükkis on *wireframe*-id koos seletustega, miks need on vastavalt tehtud. Iga kasutatud printsiibiga koos on ka viide teooria osa vastavasse peatükki, kus vastava teema kohta on rohkem kirjutatud.

Kolmanda peatükki esimene pool on suunatud valitud tarkvara "*Import*" osa disainimisele. Kolmanda peatükki teine pool aga "*Input - Financial, Other & Intercompany data*" osa ümber disainimisele. Kolmandas peatükis on ka seletused, kuidas on saavutatud *"Liquid Web Application"* disain läbi "*Responisve Web Design*-i."

Neljandas peatükis on esitatud bakalaureusetöö tulemused. Lisaks on neljandas peatükis ka soovitused bakalaureusetöö edasiseks arendamiseks.

2 Ülevaade printsiipidest ja kasutatavatest tehnoloogiatest

2.1 Kasutajaliidese tähtsus ja esinevad probleemid

Kasutajaliidest (*UI*) peetakse üheks tähtsamaks rakenduse osaks, kuna see seob lõpp kasutaja funktsionaalsusega. Hästi ehitatud, huvitava funktsionaalsusega rakendus võib halva kasutajaliidese tõttu siiski läbikukkuda. Paljud disainimis printsiibid on sihitud ehitama ühte kasutajaliidest, mis sobiks nii paljudele kasutajatele, kui võimalik. Selline lähenemine aga ei suuda, paljudel juhtudel, kohandada kõiki, erinevatest kasutus keskkondadest, tulevaid vajadusi disaini nii, et kliendi kogemus (*UX*), rakendust kasutades oleks parim. Samas, kui ehitada käsitsi mitmeid kasutajaliideseid (*UI*) erinevate keskkondade jaoks, siis tõuseb arendamise hind ja on raske ette näha kõiki võimalike, vastavast keskkonnast, ette tulevaid muudatusi, mida peab arendamisel arvesse võtma. Kohanevad kasutajaliidesed on lahendus, mis on soovitatavad keskkonna ja sisu muutlikute rakendust jaoks, sest need suudavad kohaneda vastavalt olukorrale. Idee on selles, et keskkond kohaneb kasutajale, mitte vastupidi. Kuigi idee on lihtne siis kohanevate kasutajaliideste tehniline arendamine on suhteliselt keerukas. Osad probleemidest tulenevad tehnoloogia arendamisest, mis toetaks kohanevate kasutajaliideste arendamist, teised, aga kasutajapoolsed, kas inimesed soovivad selliseid rakendusi kasutada. (Akiki jt., 2015, 1–2)

Kohanevad kasutajaliidesed on konteksti teadlikud ehk süsteem on teadlik oma kontekstist ja selle muutusel suudab see teha teatud reeglite järgi vajalike muudatusi kontekstis. Lisaks on kohanevad kasutajaliidesed ka ise konfigureeriv, mis tähendab, et see on suuteline ennast ise automaatselt ümber konfigureerima vastavalt muutustele. Et hoida rakenduse kohanemis reeglid ajakohased siis on vaja süsteemi, mis järgiks selliseid muudatusi. Teine võimalus on jälgida kasutaja tegevust, tagasisidet ja kohandada reegleid vastavalt sellele. Kohanduv kasutajaliides on veel ise-optimeeriv ehk rakendus oskab juhtida süsteemi jõudlust ja ressursside paigutust töötamisel. Seda oskust ülekandes kasutajaliidesele võime öelda, et *UI* suudab näiteks muuta paigutuse

reegleid, pakkuda uut navigatsiooni abi kasutajale või näiteks suudab eemaldada mõne *feature* -i, mida klient ei soovi. (Akiki jt., 2015, 2)

Paljud rakendused kannatavad kasutatavuse all, sest nende *UI* ei arvesta konteksti muutumisega. *Enterpirise* rakendused, nagu enterprise resource planning systems on üks näide sellisest aplikatsioonist. On pakutud välja palju erinevaid lähenemisi kasutajaliideste kasutatavuse tõstmiseks, põhinedes kriteeriumitel, nagu kättesaadavus, kokkulangevad ülesanded, kultuur, kontekst, platvorm jne. (Akiki jt., 2015, 3–4)

2.2 Mudeli põhine lahendus

MDE ehk (*Model-driven engineering*) mudeli põhine projekteerimine või arendamine kombineerib protsessi ja analüüsimist arhitektuuriga, püüdes tõsta aplikatsiooni abstraktsiooni taset. Mudeli põhisel arendamisel on võimalik rakendada erinevatel abstraktsiooni tasemetel kohandusi kasutajaliidesele, mis aitab luua kohanevaid kasutajaliideseid. Erinevad mudeli põhise arenduse vormid on nii-öelda staatiline modelleerimine, genereeriv *runtime* modelleerimine ja interpreteeriv *runtime* modelleerimine. Keskendudes Staatilisele modelleerimisele, toetub see mudelitele, luues kasutajaliidest. Staatilised mudelid on aga muutumatud ja on seega kasulikud ainult arendamise protsessis. (Akiki jt., 2015, 7)

2.3 *Liquid software* ja *Liquid Web Application* tutvustus

Tänapäeva keskmisel U.S.A või Euroopa tarbijal on nii personaalne arvuti, kui ka "nutikas telefon." Pidevalt tuleb kasutusele uuemaid telefone ja tahvleid, me oleme liikumas maailma, kus inimestel on igapäevaselt kasutuses mitmeid arvuti/telefon tüüpi seadmeid. Internet on muutunud seadmetele järjest ligipääsetavamaks ja et pakkuda kasutajatele normaalset elamust, mida nad ootavad on vaja veebis kasutada *Liquid software* loogikat. See tehnoloogia aitab ehitada veebilehti, mis suudavad sujuvalt migreeruda ühelt seadmelt teisele, jälgides ka kasutaja tähelepanu ja konteksti, millega seoses klient teenust/veebilehte kasutab. Rakendades *Liquid software* printsiipe veebi aplikatsioonile saavutamegi rakenduse, mida võib nimetada *Liquid Web Application*. (Mikkonen jt., 2015, 134–135)

Liquid software, selle põhiline sisu seisneb probleemi vaba elamuse pakkumises, veebilehe vaatlemises mitmete erinevate seadmetega. See seisneb loogikal, et aplikatsiooni saab vaadelda erinevatel seadmetel, nii et aplikatsioon kohaneb vastavalt seadmele. Lisaks peaks aplikatsiooni olema võimalik kasutata ühe kasutaja pool erinevatel seadmetel samaaegselt. Vaated võivad vastavalt seadmele erineda. Aplikatsiooni saavad kasutada erinevad kasutajad nende enda seadmel, kas samaaegselt või kordamööda. Sellise loogika realiseerimiseks peab kasutama andmete sünkroniseerimist, et info muutumisel lehel jõuaks muutus kõikidele erinevatele seadmetele. (Mikkonen jt., 2015, 135)

Liquid Web Application printsiibil ehitatud kasutajaliides peab toetama erinevatel seadmetel rakenduse kasutamist. Samuti peab selline kasutajaliides olema ehitatud nii, et oleks toetatud samaagne rakenduse kasutamine. Üks võimalik stsenaarium on näiteks vaadates samaaegselt ühte sama rakenduse vaadet on andmete sisestamine sünkroonis, nii et ühel seadmel tehtud muutus oleks näha teises seadmes vaatavale kasutajale. Veel üks näide oleks telefoni vaate kasutamine, et juhtida suuremat arvuti ekraani vaadet. Sellised lahendused on lihtsamad tsentraliseeritud ja serveri põhistes rakendustes. (Mikkonen jt., 2015, 138-139)

Põhiline tehnoloogia, mida kasutatakse, et muuta veebilehed erinevate seadmete jaoks sobivaks on "*Responsive web design." Responsive* disain oma olemuselt põhineb paindliku asetuse ehitamisel, mida oleks lihtne muuta vastavalt seadmele. Funktsionaalsuse arendamisel tähendab *responsive* disaini kasutamine, et osa funktsionaalsust toimetatakse serveri pool. Arendaja kohapealt tähendab *responsive* disaini kasutamine, et kasulik on kasutada valmis vahendeid nagu näiteks *bootstrap,* mis toetavad *responsive* disaini. (Mikkonen jt., 2015, 140)

2.4 Näited *Liquid* tehnoloogiate toimimisest

Vahest on kasutajaliidese taastamiseks, viimati oldud olekusse, piisav, kui anda edasi hüperlink. Näiteks video juhul võib link sisaldad video punkti, kust mängimist jätkata ja ka heli punkti, kust jätkata. Keerulisemate veebilehtede korral võib lingist väheks jääda, sest informatsiooni, mis olekut kirjeldab võib olla liiga palju. Google docs on üks näide aplikatsioonist, mida saavad mitmed inimesed samaaegselt kasutada, samuti ka eraldi.

Google docs-is sünkroonitakse dokumendis tehtud muudatused (andmed), aga tööriista menüü muudatused mitte ehk kui seadet vahetada siis teksti muudatused on alles. Samas me ei soovi piirduda ainult näiteks pilveteenustega nagu Google docs vaid soovime rakendada *liquid* tehnoloogiat ka mobiili aplikatsioonidele nagu näiteks kontaktide raamat telefonis. Paljud telefoni aplikatsioonidest, mis pole internetiga ühenduses tehakse varsti arvatavasti samuti ümber veebiteenuseteks, aga need sisaldavad siiski palju veebist eraldatud lokaalset tegevust. Lokaalne tegevus seadmes tähendab, et esineb seadme oleku, info sünkroniseerimist teiste seadmetega või siis pilve teenuse ja seadme vahelist sünkroniseerimist ehk esineb erinevaid vorme *liquid* tehnoloogiast. Kasulik sünkroniseerimis vorm seadete vahel võib olla lähedus põhine sünkroniseerimine. Üks näite stsenaarium *liquid* tehnoloogiast on kui alustada e-maili sisestamist telefonis, aga siis minna ümber arvutisse ja lõpetada sisestamine seal. *Liquid* tehnoloogiat võiks kasutatada veebis, mis ei ole piiratud kindlasse operatsiooni süsteemi ehk piiratud kindlasse seadmesse, selle omaduste järgi ja kus r*esponsive* disaini läbi on aplikatsioonid ehitatud kohanema erinevatele seadmetele. (Mikkonen jt., 2015, 135–137)

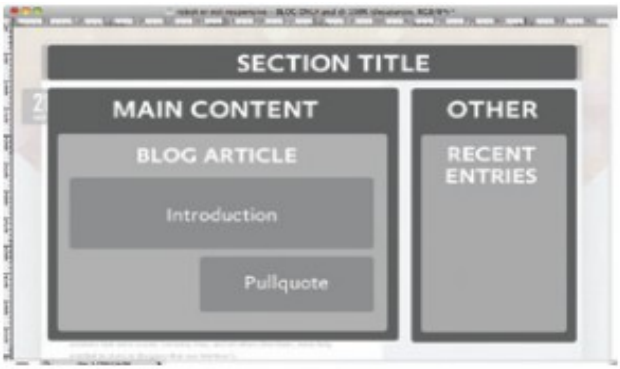
2.5 Responsive disain

Arhitektuuri ala *responsive* disainerid loovad ruume, maju mis muudavad oma ilmet, keskkonda võttes arvesse seda läbivaid, seal olevaid inimesi. Selle asemel, et inimesed peaksid kohanema ruumiga kohaneb ruum inimestega. Näiteks on olemas niinimetatud tark klaas, mis muudab oma läbipaistvust vastavalt inimeste tihedusele ruumis, pakkudes lisa privaatsust. Arhitektuur, mis on loodud arhitekti poolt, tehes loovaid otsuseid, võib seista sajandeid. Selliseid võtteid võiks kasutada ka veebis. (Marcotte 2011, 7–9)

Veebi disaineritel on vaja saada üle veebi paindlikkuse probleemidest. Selle asemel, et luua igale seadmele, brauserile erinevat veebilehte peaks looma ühe veebilehe, mis suudab kohaneda vastavalt keskkonnale, kus seda kasutatakse. Selleks peaks kasutama R*esponsive Web Design*. Selle põhilised printsiibid on paindlik *grid*-põhine asetus, paindlikut pildid ja paindlik meedia ja kolmandaks *Media queries* CSS3-lt. (Marcotte 2011, 7–9)

2.6 Responsive disaini põhine asetus

Rääkides erinevatest kaunitest kunstidest, kas muusika või näiteks kirjandus, siis võib öelda, et igaüks neist on vastukaja neile eelnevast kunstivoolust. 20. sajandi keskosa Modernistlikud artistid tihti vaatlesid oma eelkäijate, 19. sajandi Romantistliku perioodi artistide töid. Antud modernistid vaatlesid küll 19. sajandi romantiste põlgusega, sest romantistide ornamentika tundus neile liigne ja ebavajalik. 20. sajandi keskosa modernistid väitsid, et selline liigne ornamentika segas teose suhtlemist vaatlejaga. Modernistide reaktsioonid olid erinevad, aga graafilised disainerid nagu Jan Tschichold, Emil Ruder ja Josef Müller-Brockmann lõid süsteemi, mis koosnes ridade ja veergude loogikast. Kunsti teose sisu osad sisestati vastavatesse veergudesse ja ridadesse. Sama loogika on ka kohandatud kaasaegsesse veebidisaini, kuna aga veebilehe suurus on muudetav siis tuleb seda arvestada, kui me soovime teha kohenevat/muutuvat veebilehte. (Marcotte 2011, 13–15)



Joonis 1. Lehekülje sisu inventuur (Marcotte 2011, 24).

Luues paindlikku *grid*-põhist asetust, tuleks lehe osad, div klasside abil, struktureerida. Abiks on kui on olemas veebilehe sisu inventuur (Joonis.1). *Grid*-põhine asetus on lehe skelett nagu näiteks joonisel 2. (Marcotte 2011, 23, 25)

```
<div id=”page”>
	<div class=”title”>
		<!-- kood -->
	</div><!-- div title lõppeb>
	<div class=”main”>
		<!-- kood -->
	</div><!-- div main lõppeb>
	<div class=”other”>
		<!-- kood -->
	</div><!-- div other lõppeb>
</div> <!-- div page lõppeb>
```

Joonis 2: *Grid* põhise asetuse näide (Marcotte 2011, 23, 25).

Siin on div veebileht, mis hoiab endas kõiki lehekülje elemente, siis veel title osa, kus on lehe pealkiri jne. Nüüd on vaja muuta lehe osad nagu *title*, *main* ja *other* vastavateks ridadeks ja veergudeks. (Marcotte 2011, 23, 25)

Luues oma *CSS*-i, kui me soovime teha kohanevat asetust, mis vastavalt brauseri akna suurusele muudab oma asetust, peaksime me kasutama pikslite asemel suhtelisi suurusi. Pikslitel ei ole midagi viga, aga luues *responsive* veebilehte on parem kasutada näiteks protsente. Üks viis protsentide määramiseks on katse eksitus meetod, aga teaduslikum meetod on kasutada valemit objekt / kontekst = tulemus mis tuleb korrutada arvuga sada, et saada tulemus. Objekti all on valemis mõeldud objekti pikslite arvu, mis algselt määratud ja konteksti all teda ümbritseva div-i suurus. Näiteks *title* div-i jaoks oleks kontekst *page* div-i pikslite arv. Kasutades protsente, mis määravad veebilehe osade suuruse saame me kiire ja kohaneva veebilehe, mis muutub vastavalt brauseri akna suurusele. (Marcotte 2011, 28–33)

2.7 Media queries

Ükski disain kohanduv või staatiline ei muuda ennast väga hästi suurustel, millele see polnud algselt disainitud. *Responsive* disainil piirab väiksematel resolutsioonidel meie *grid*-põhine asetus lehe sisu, suurematel resolutsioonidel aga jääb liialt palju tühja ruumi. Ehk siis, kui me jõuame suurustele, millele loodud disain pole mõeldud muutub *grid* takistuseks lehe sisule. Õnneks on *W3C* välja arendanud ühe lahenduse sellele probleemile. (Marcotte 2011, 70–71)

Kõigepealt on *CSS2*-e spetsifikatsioonist tulenevad *media types,* mis on kategooriad koostatud erinevatest brauseritest ja seadmetest. Kategooriatesse kuuluvad näiteks *all, braille, embossed, handheld, screen* ja neid on veel. Tuntumad on *screen* ja *print*, aga kõik on loodud, et saaks disainida paremini kindla seadme jaoks. *Media type* tunneb ära, mis seadmega on tegu ja vastavalt sellele laeb *CSS* disaini faili. Näiteks koodi jupp joonisel 3. (Marcotte 2011, 72–74)

```
<link rel=”stylesheet” href=”disain.css” media=”all”/>
```

Joonis 3: *Media type* näide (Marcotte 2011, 72).

Või siis *CSS* disaini failis lisada spetsifikatsioon (Joonis 4). (Marcotte 2011, 72–74)

```
@media screen {
	body {
		font-size: 100%;
	}
}
@media print {
	body {
		font-size: 15pt;
	}
}
```

Joonis 4: *@Media* koodi näide *CSS* failis (Marcotte 2011, 73).

Media type aga ei lahenda kõiki probleeme üksinda, kuna laialt on levima hakanud väike ekraan seaded. Kuna varajastel telefonidel polnud veel korralikke brausereid siis disaineri ei kasutanud algul *handheld* spetsifikatsiooni ja kui need hiljem tulid ei olnud eriti *CSS* disaini faile, mis kujundaksid *handheld* valikule. Selle tulemusena paljud disainerid hakkasid lihtsalt kasutama *screen* valikut. Lisaks sellele probleemile võib mõni *media type* olla ka liiga üldine, sest näiteks telefone on ju erinevaid, kõigile ei pruugi sama disain sobida. (Marcotte 2011, 72–74)

Probleemi püüdis *W3C* lahendada tehnoloogiaga *media queries*, see tunneb ära lisaks seadme tüübile ka selle füüsilise karakteristika. *Media query*-l on 2 osa, see tunneb ära meedia tüübi ja vastava füüsilise karakteristika (joonis 5). (Marcotte 2011, 74–75)

```
@media screen and (min-width: 1024px){
	body {
		font-size: 100%;
	}
}
```

Joonis 5: *Media query* näide (Marcotte 2011, 74).

Füüsilise karakteristika osa on …. (min-width: 1024px). *Min-with* on vastava karakteristika nimi ja 1024px on suurus, mille korral antud päring aktiveerub (meedia tüüp peab aktiveerumiseks olema *screen*). Antud lahendus hoiab disaini koodi ühes failis, aga saab viidata ka teistele disaini failidele, näiteks eeltoodud (joonis 6) *link* lahendus. (Marcotte 2011, 74–75)

```
<link rel=”stylesheet” href=”wide.css” media=”screen and
(min-width: 1024px)”/>
```

Joonis 6: *@Media* lingi näide (Marcotte 2011, 75).

Võis siis:

```
@import url(“wide.css”) screen and (min-width: 1024px);
```

Joonis 7: *Media query @import* näide (Marcotte 2011, 75).

Muidugi on palju teisi kriteeriume, mida saab kontrollida, lisaks pikkusele (*height*) ja laiusele (width). (Marcotte 2011, 74–75)

Media query aitab meil parandada vigu, mis tulevad sisse seadmete vahetamise või lihtsalt brauseri suuruse muutmise tõttu, lehe asetuse muutusest. *Media query* abil saame parandada visuaalseid vigu, näiteks optimeerides lehe sisu vastavalt seadmele või muutes veebilehe elementide resolutsioone. *Media query* aitab meil muuta lehe tundlikumaks ja paremini kohanevaks vastavalt seadmele, mis lehte kasutab. Paindliku *grid*-põhise asetusega ja *media query*-ga saame luua *responsive* põhimõtetega veebilehti. (Marcotte 2011, 79)

2.8 „Telefon ennem“

Vahest eeldatakse, et kuna suure ekraaniga kasutajale saab anda rohkem informatsiooni, kuna on rohkem ruumi. Telefoni kasutajale, aga tuleks anda ainult vajalik informatsioon, sest ruumi lihtsalt ei ole. Selline lähenemine ei ole, aga parim, kui ekraanil on rohkem ruumi ei ole vaja see täis toppida elemente, mida kasutaja ei vaja. Erinevate suurustega ekraanide kasutajatel peaks olema kõigil õigus fokuseeritud sisuga veebilehtedele, mitte ainult telefoni kasutajatel. (Marcotte 2011, 111)

Disainer Luke Wroblewski on välja pakkunud disainimis filosoofia, mille kohaselt peaks kõigepealt disainima telefonile ja seejärel teistele seadmetele. “Telefon ennem” on *responsive* disainile kasulik printsiip. Kuna aina rohkem seadmeid ja brausereid kasutavad meie veebilehti ja telefoni põhine veebi liiklus on märgatavalt kasvanud, ei peaks telefoni disain olema järelmõte. (Marcotte 2011, 111–112)

2.9 Taju põhine disain

Kuna meie taju ümbritseva suhtes on mõjutatud, meie kogemustes, hetke olukorrast ja meie eesmärkidest, siis seda saab võtta arvesse, kui me disainime veebilehte. Kui rääkida meie kogemuste mõjust meie tajule, siis kui näiteks võtta meie kodu toad, õued, naabruskond jne. Siis kui me liigume sellises keskkonnas on meil teatud eelarvamused, mida me kusagil näeme. Aga samas on meil ka eelarvamused keskkondade jaoks nagu näiteks köök, me eeldame, et seal on teatud esemed nagu kraanikauss, külmik jne. Sellised eelarvamused, aga tekitavad vahest efekti, et me arvame, et me nägime, kusagil midagi, mida seal tegelikult polnud, sest meie kujutluses see ese, objekt peaks sinna ümbruskonda kuuluma. Sama moodi on ka rakenduse, veebilehe kasutamisel, kas arvutis või telefonis. On elemendid, mis me arvame, et peaksid kuuluma veebilehele või desktopi rakendusele, üldine menüü, midagi, mille järgi arusaada, mis lehega on tegu jne. Kuna meil on kujutis või *frame* mõttes, kus midagi lehel võiks olla siis vahest me klõpsame nuppudel ilma neid lugemata, sest me arvame alateadlikult, mis mingi nupp teha võiks. Näiteks, kui erinevatel lehtedel on sarnased nupud, mida mööda kasutaja järjest liigub, siis kui mõnel lehel nuppude asetus muutub, siis paljud kasutajad ei pruugi seda märgata ja võivad vale nuppu klõpsata. (Johnson 2010, 1, 3–4)

Kui rääkida hetke olukorra mõjudest siis võttes näiteks lugemise, võib väite, et üldine mõjutab spetsiifilist. Näiteks lause mõjutab, kuidas me näeme üksikuid sõnu ja üksik sõna mõjutab, kuidas me näeme selle sõna mõnda tähte. Muidugi tähed moodustavad sõnad ja sõnad laused, nii et see toimib ka teistpidi. Meie nägemine ei loo alati teaduslikult korrektset arusaama objektist, läbi erinevate optiliste illusioonide abil on võimalik mõttelt ja arvuliselt sarnaseid, sama suuri kujutisi esitada nagu need oleksid erineva suuruse või mõttega. Muidugi, kuidas me midagi näeme on mõjutatud peale nägemismeele näiteks ka kuulmise ja muude meelte kaudu. Näiteks kõhurääkija efekt,

me näeme liikuvat nuku suud ja kuuleme häält ja kuna nuku suu on see, mida me märkame, siis me viime kaks elementi kokku. (Johnson 2010, 6–8)

Rääkides eesmärgi põhisest tajust siis, kui me otsime mingit kindlat informatsiooni veebilehelt või tahame täita spetsiifilist ülesannet siis me keskendume nendele objektidele/elementidele veebilehel, mis on vastava ülesandega seotud. Sellisel juhul me võime täielikult eirata elemente, mis pole meie eesmärgiga seotud. Näiteks, kui me oleme, millestki väga huvitatud siis me suudame keskenduda ainult sellele ja ülejäänud müra tõrjub meie aju eemale. Kuna täiskasvanud on rohkem eesmärkidele suunatud, kui lapsed, siis nad märkavad ka asju, mis pole nende eesmärgiga seotud vähem. Lapsed, kelle tähelepanu hajub kergemalt, märkavad kõrvalisi asju, aga rohkem ja lapsed pole ka nii eesmärkidele keskendunud, kui täiskasvanud. (Johnson 2010, 9–10)

Neid taju elemente saab kasutada disainis mitut moodi. Näiteks on kasulik esita informatsiooni selgelt ja ühemõtteliselt või kasutada üldisi standardeid, kui muud võimalust pole. Teiseks elementide paigutamisel tuleb olla järiekindel. Erinevatel lehtedel olevad samad elemendid peaksid samasugused välja nägema, nii on kasutajal neid kergem märgata. Oluline põhimõtte on ka teada, mis ülesandeid kasutajad tahavad lehel täita ja disainer peaks igas iteratsiooni osas vajaliku informatsiooni kasutajale nähtavaks/arusaadavaks tegema. Muidugi saab neid taju põhimõtteid ka muud moodi kasutada. (Johnson 2010, 12)

2.10 Gestalti printsiibid

20 sajandi algus poolel viisid grupp saksa psühholooge läbi uurimise inimese nägemistaju kohta. Üks põhilisi avastusi, mis nad tegid oli, et inimeste nägemine on suunatud struktuuride otsimisele, mitte eraldi seisvate nurkade või üksikute joonte märkamisele. See on nimetatud “Gestalti nägemistaju printisiibid”. Tänapäeva teooriad inimese nägemistaju kohta keskenduvad põhiliselt silma neurofüsioloogial, aga uued avastused siiski toetavad Gestalti printsiipe. Gestalti printsiibid on siia maani kehtivad, kuigi need on pigem nagu *frameworks*, millega nägemistaju kirjeldada, mitte vundamentaalsed tõestused. (Johnson 2010, 11, 13)

Kui elemendid on ekraanil halvasti struktureeritud ja visuaalselt pole grupid eristatavad siis on kasutajal raskem õppida tarkvara kasutama. Gestalti läheduse printsiip väidab, et elementide vaheline lähedus mõjutab, kuidas me neid grupeerime ekraanil. Objektid/elemendid, mis on teiste elementide suhtes üksteisele lähemal tunduvad grupeeritud. Disainerid kasutavad tihti vahejooni või panevad elemendid ühtsesse konteineri, et need oleksid grupeeritud, aga Gestalti printsiibi järgi ei ole piirjooni vaja, kui elemendid on üksteisele visuaalselt lähedal, võrreldes teiste elementidega. (Johnson 2010, 14–16)

Gestalti sarnasuse printsiibi järgi objektid, mis näevad samasugused välja, aga on teistest erinevad tunduvad meile visuaalselt olevat gruppides. Ühendades Gestaldi sarnasuse printsiibi Gestaldi läheduse printsiibiga on võimalik jõuda hea tulemuseni, elementide gruppeerimisel. Kui luua grupp samasuurustega tähtedest, mis asuvad üksteisest võrdsetel kaugustel, näiteks neljas reas ja veerus ja kui värvida nendest üks rida teist värvi, siis see üks rida tundub ülejäänutest erinevalt grupeeritud. (Johnson 2010, 16)

2.11 Struktuurne veebileht

Kui kasutajad navigeerivad läbi veebilehe siis nad ei loe igat sõna vaid lasevad silmadega üle veebilehe, otsides neile vajalikku informatsiooni. Struktuurselt veebilehelt on aga lihtsam leida informatsiooni, sest meie nägemistaju on suunatud struktuuri leidmisele. Samuti kiirendab struktureerimine informatsiooni leidmist ja sellest aru saamist. Struktureerimine aitab vältida ka elementide kordust, näiteks otsingu tulemuste kuvamisel. Veebilehe struktureerimine ja erinevate graafilise disaini printsiipide, näiteks Gestalti, kasutamine aitab meil efektiivselt anda kasutajale edasi vajalikku informatsiooni. (Johnson 2010, 25–26)

Struktureerimis näide on isegi, midagi sellist väikest nagu telefoni number või kuupäev. Kas kirjutada number järjest pika reana või jagada osadeks. Veebilehtedel saab jagada numbreid andes igale osale numbrist oma tekstisisestus väljas. Samal ajal, kui veebilehe loomiseks kasutatav tehnoloogia lubab, saab kasutada ka tühikuid numbri osade eraldamiseks ühes väljas. Kuupäevadega aitab paremini, kui kasutada andme-spetsiifilist sisestus välja. Näiteks kasutades *dropdown listi*, kust kasutaja saab valida

ainult kindlad väärtused. Nii on andmed struktureeritud ja selline lähenemine aitab ka vältida andmesisestus vigasid. (Johnson 2010, 28–29)

Visuaalse hierarhia, aitab kasutajal leida informatsiooni, mis on talle vajalik ja seda teha kiiresti. Nähtavalt arusaadava hierarhia loomine on oluline nii vormides, mida kasutaja peab täitma või seadete määramise vaadetes, samamoodi nagu lehtedel, kust informatsiooni lihtsalt loetakse. Hierarhia peaks olema loodud nii, et see looks informatsioonist kindlad visuaalselt märgatavad osad/sektsioonid. Kui on vajalik luua ka alamsektsioone siis, need peaksid olema visuaalselt eristatavad põhisektsioonidest ja põhisektsioonid peavad olema esmatähtsalt esitatud. Igal sektsioonil peaks olema täpne nimi, midagi sellist, millest on kiiresti arusaada, mis lõiguga/sektsiooniga on tegu. (Johnson 2010, 30–31)

2.12 Kohanemisvõimeline asetus erinevatel seadmetele

Kasutajad vaatlevad *app*-i otsides sealt informatsiooni, millest nad hoolivad, seega on tähtis, et *app*-i põhifunktsioonile jääks fookus kõikides keskkondades. Tähtis on ka vältida suuri muudatusi *app*-i struktuuris, nii tunnevad, uues keskkonnas, kasutajad ära siiski tuttavad töö mustrid. Kui *app* ei vaheta seadmel näiteks telefoni asetust muutes ekraani olekut, ekraan püsib, kas ainult pikki või küliti pidi, siis pole kasutajale vaja öelda, et ta peaks telefoni olekut muutma. Lisades elemente, mis kasutajale annavad teada, et ta peaks seadme positsiooni muutma, lisavad lihtsalt visuaalselt ebavajalikku segadust, kasutaja saab aru, et peab positsiooni muuta, kui ekraan ei reageeri. Toetama peaks, ekraani ainult näiteks püsti pidi püsimisel, siiski ka seda, et kui seade tagurpidi pöörata siis ekraan sellele kohaneb. Mängu korral, kus seadme positsioon on kui kasutaja sisend peaks, seade muutma ekraani positsiooni vastavalt *app*-i funktsionaalsusele. Näiteks mängu korral, nii kaua, kui otsene mäng pole veel alanud, kasutaja on mängu menüüs, vahetab seade ekraani olekut vastavalt telefoni positsioonile Mängu alates, aga muudab ekraani olekut vastavalt mängu funktsionaalsusele. (iOS Developer Library)

Asetuse kasutamises suhtlemises kasutajaga saab näidata, mis elemendid on tähtsamad ja millised on omavahel seotud. Näiteks elemendid, mis on tähtsamad on parem paigutada ekraani ülemisse vasakusse äärde, vähem olulised alla paremale. Samuti

elemendid, mis on suuremad jäävad silma, kui tähtsamad võrreldes väiksemate elementidega. Suuremaid elemente on näiteks telefonide korral ka lihtsam klõpsata. Telefonis on mõistlik anda nuppudele 44 x 44 pikslise suuruse. Hea on kasutada ka elementide joondamist ja grupeerimist. Grupeeritud elementidest on kasutajal parem läbi kerida, sest tal on lihtsam informatsioonile keskenduda. Hea tava on ka vältida elementide kujutamist väga erinevalt, millel on sarnased ülesanded. Parem on kui need elemendid on sarnased. Peab olema kindel, et kasutaja saab *app*-i peamist sisu lugeda selle normaal suuruses ilma vajaduseta seda suurendada. Samas peab olema *app* valmis kohanema, kui kasutaja tahab seadetes muuta teksti suurust. (iOS Developer Library)

2.13 Kolme sammu/klõpsu reegel

Kolme klõpsu reegel on eksisteerinud veebi algus aegadest alates. Reegel põhineb loogikal, et kui kasutaja ei leia, mida ta vajab kolme klõpsu järel ta arvatavasti lahkub lehelt ja mida rohkem kasutajad peavad klõpsima seda väiksem on nende huvi veebilehel pakutava vastu. Kui me põhineme kolme klõpsu reeglil ehitades oma veebilehte, tuleb leht suurema tõenäosusega loogilise struktuuriga. Üks näite soovitus kolme klõpsu reeglist on, et tuleks panna globaalne navigatsioon. (Porter 2003)

Üldiselt on lepitud arusaamisega, et veebi kasutajad on fokuseeritud kiirusele. Samas võib tunduda, et kui tavalises lehes on sadu elemente siis kolme klõpsu reegel on võimatu saavutada, kuid kui alustada planeerimist varajases arenduse punktis. Pannes paika, mida kasutaja lehelt tahab ja kuidas on see saavutatav. Kolme klõpsu reegli jaoks tuleks veebileht jagada navigatsiooni osas “Viie reegli” järgi. (Zeldman 2001, 98)

Viie reegel on kirjeldatud järgmises peatükkis.

Kolme klõpsu reegli alla, läheb ka see, kui kasutaja pärast kolme sammu jõuab punkti, kus ta teab, et ta on õige koha leidnud, kuigi tal võib minna veel üks klõps, et otsese infoni jõuda. Kolme klõpsu reegel on pigem soovituslik, mitte reegel, mida ei tohi kunagi murda. (Zeldman 2001, 98)

Kolme klõpsu reegli testimisel leiti, et osad kasutajad jätsid oma tegevused pooleli pärast kolme klõpsu või pärast kahe, kolme lehe külastamist, samal ajal aga paljud

jätkasid oma tegevust kuni 12 klõpsuni või külastasid kuni 25 lehte. Uuringust aga ei leitud loogikat, mis oleks näidanud seotust klõpsude arvu ja vajaliku info leidmise vahel. Samamoodi võrreldi ka kasutaja rahulolu klõpsude arvu kohapealt. Kasutajad väitsid, et neid häirib, kui info leidmine võtab liiga kaua aega. Seega kasutajatel tekib frustratsioon, kui nad ei leia vajalikku informatsiooni, aga see ei tähenda tingimata, et klõpsude arv on see, mis põhjustab ärritumist. (Porter 2003)

Kolme klõpsu reegel on keskendunud kliendile ja aitab disaineritel luua veebilehti, mis põhinevad kasutajale vajaliku info andmisel. Seega ei saa väita, et see oleks halb reegel, selle järgimine võib aidata luua paremaid veebilehti. Kuna kasutajaid huvitab neile vajaliku tegevuse lõpetamine ja kui kolme klõpsu reegel aitab seda saavutada, siis on kasulik seda meeles pidada. (Porter 2003)

2.14 Viie reegel

Niinimetatud viie reegel seisneb kasutajale, veebilehe navigatsioonis, võimalikult selgete valikute andmises. Nimelt paljude valikutega, mitme tasemelised menüüd ajavad kasutajaid segadusse. Parem on anda kasutajale kolm kuni viis valikut, nii on neil lihtsam valida, kuhu minna. Mitme erineva funktsionaalsusega veebilehtedes, aga võivad olla mitmekihilised navigatsiooni menüüd. Mõned disainerid ja arhitektid jagavad eraldi navigatsiooni menüüdesse ülesande põhised ja eesmärgi põhised lingid. Eesmärgi põhised on sellised, kus kasutaja soovib veebilehel liikuda, et leida infot. Ülesande põhine liikumine toimub kasutaja poolt siis, kui ta soovib midagi kindlat teha, näiteks kandideerida tööle. Mängides läbi erinevad stsenaariumid, mis on seotud veebilehe navigatsiooniga on Viie reeglit ja Kolme klõpsu reeglit lihtsam realiseerida. (Zeldman 2001, 99–100)

2.15 Perifeerne nägemine ja disain

Meie nägemine on palju teravam silma keskosas, kui äärtes, kust tuleb nägemise perifeerne osa. Silma keskosas on väike osa nimega *fovea*, mis on umbes 1% võrkkestast, aga sellelt tulevale infole koondub enam vähem pool meie aju visuaalse pildi moodustavast alast. Liikudes *fovea* alalt välja halveneb nägemine koheselt,

liikudes meie nägemise ääre alale on olukord sedavõrd hullem. Põhjus, miks me üldse näeme ümbritsevat, peale ala, mis on otse meie ees on see, et meie silm liigub sekundis umbes kolm korda ilma, et me ise seda märkaks. Nii moodi *fovea* saab info ja meie aju täidab tühjad kohad. Nii et kuigi meie perifeerne nägemine on nõrk me saame siiski kujutise, mis on meie ümber. (Johnson 2010, 11, 65–66)

Üks probleemides veebilehtede kasutajatel on vea teadete märkamine, kui see pole vähemalt 2 sentimeetri ulatuses klõpsu kohast, ei pruugi kasutaja seda märgata. Näiteks võttes sisse- flogimise. Vea tekkides logimisel, kui seda viga ei kuvata piisavalt lähedal klõpsamise kohale, võib kasutajal tekkida arusaamatus. Arusaamatus võib tekkida just siis, kui veateade jääb meie perifeersesse nägemise alasse. Sellisel juhul ei saa kasutaja aru, mida ta edasi peab tegema. Alles hilisemal lehe ülevaatamisel, märkab kasutaja vea teadet. (Johnson 2010, 11, 54)

Kuigi meie perifeerne nägemine on mõneti ebausaldusväärne elementide leidmisel, mis pole nägemise keskel, siis see siiski mängib mõningat rolli, kui kasutaja midagi lehelt otsib. Näiteks, kui me otsime midagi elementide hulgast näiteks loendist siis kõik meie nägemis meeled aitavad meid selles. Samas aga otsingu kiirus oleneb ka loendis olevate elementide välimusest (värv, suurus jne). Kui kõik elemendid otsitavas grupis on sarnased siis on suhteliselt raske, kindlat objekti leida, aga erinevate objektidega on see lihtsam. Elementi saab teistest eristada selle värvi muutes, elemendi teistest suuremaks tehes, või näiteks tähe teksti tugevuse suurendades. Näiteks punast värvi kasutatakse enamasti probleemide esitamisena. Sellist loogikat kutsutakse vahest ka "*pop*" - välja hüppav, kasutatakse tähtsa informatsiooni välja näitamiseks. (Johnson 2010, 62–64)

Hea tehnika, mis aitab kindlaks teha, kuhu kasutajad kõige rohkem tähelepanu osutavad on nii-öelda soojus kaardid. Soojus kaardid moodustatakse vastavalt kasutajate klõpsamistele tulnud informatsioonist. Näiteks ettevõte *North Face* uuris oma veebipoodi, kuna neil oli probleem, et inimesed ei suundunud maksma toimingute lõpetamisel. Kasutades soojus kaartide lähenemist kasutavat uuringut avastati, et paljud kasutajad märkasid ühte reklaam nuppu rohkem, kui maksmisele suunavat nuppu. Muutes lehe vastavate nuppude disain suurenes ettevõtte kasum. (Hergul)

Soojus kaartide lähenemise järgi on võimalik teada veebilehe populaarsemaid osasid. On võimalik näha soojus kaardil veebilehe osasid, kuhu kasutajad kõige rohkem

klõpsavad. Saab hinnata ka kasutaja huvi erinevate lehe osade vastu selle järgi, kuhu kursor pidama jääb. Lisaks hiire kursori liikumise ja klõsudele saab katsetada ja vaadata, kui palju lehel alla keritakse. Seega, kui kasutajad ei jõua informatsioonini, mis on neile vajalik tuleks seda liigutada ülesse poole või muuta lehe asetust. Soojus kaartide filtreerimine vastavalt kuupäevale või andmete allikale on samuti võimalik. Jälgides mingi kindla kriteeriumiga kasutajate tegevust oma lehel, kes lehega nii-öelda hästi sobindusid, saab teha mugandusi oma lehele, et nendele kasutajatele paremini oma veebilehte sobitada. Kasulik võib olla sellisel juhul uurida soojus kaarte. Statistika, kui palju inimesed klõpsivad, kuhu nad klõpsivad ja kuidas lehel liiguvad on samuti kasulik info, mida silmas pidada. Näiteks kui statistikast on näha, et palju klõpse on kohtades, kus pole lehe peasisu või üldse kasutajad klõpsivad liiga palju siis võib olla vajalik muuta oma disaini, sest kasutajad ei pruugi lehest aru saada. (Hergul)

2.16 Kasutajaliides ja lugemine

Paljud meie industriaalse ühiskonna igapäeva tegevused nõuavad lugemist. Enamus meie kaasaegse ühiskonna lapsi õpetatakse lugema varajases eas ja juba eakamate inimeste jaoks on lugemine automaatne tegevus. Lugemist peetakse meie ühiskonna harituse pärast, vahest sama normaalseks nagu rääkimist. See väide aga pole tõsi, lugema ja kirjutama õppisid väike valitud hulk inimesi alles mõni tuhat aastat enne Kristust ja üle maailma normaalseks sai see alles mõned sajandid tagasi. Lisaks sellele õpivad lapsed rääkima ilma mingisuguse otsese harjutamiseta, nad lihtsalt omandavad selle nende ümber olevast keskkonnast. Lugemine on aga oskus, mida peab harjutama ja õppima nagu näiteks klaveri mängu. Seega, loomulikult, oleme me suunatud inimkeelest aru saama, aga lugemine on õpitud oskus. (Johnson 2010, 11, 33)

Pikkade proosa tekstide lugemise nõudmine kasutajalt võib neid tunduvalt aeglustada või nad väldivad nende lugemist üldse (Research-Based Web Design ..., 39).

On kahte moodi lugemist või õigemini mustri äratundmis protsessi üks on tunnusjoonte-põhine teine on konteksti-põhine. Funktsiooni-põhises tunneb nägemis süsteem ennam ära üksikud tunnusjooned ja siis moodustab nendest terviku. Näiteks aju tunneb ära teatud sümboleid, kui tähti teisi, kui numbreid jne. Kui on tegemist tähtedega siis osad tähe grupid tunneb ära, kui sõnad ja sõna grupid, kui laused. Funktsiooni-põhist

lugemist võib kutsuda ka konteksti vabaks lugemiseks, sest ajul on võime ära tunda jooni, kurve jne alates sünnist. Kuigi näiteks tähti peab õppima, siis alguses on tegemist siiski mitte-automaatse tegevusega, aga harjutades muutub see ajapikku automaatseks. (Johnson 2010, 11, 35)

Konteksti-põhine lugemine toimib funktsiooni-põhise lugemisega sarnaselt aga vastupidi. Lõigust leiab kasutaja laused ja lausest sõnad ehk visuaalne süsteem märkab kõigepealt kõrgema leveli mustrid ja mõtleb välja või arvab, mis on madalama taseme komponendid. Kuna aga enamus fraase ei ilmu tekstides sama pidevalt nagu tähed sõnades siis on väike tõenäosus, et konteksti-põhine lugemine muutub automaatseks. (Johnson 2010, 35)

Automaatset lugemist saab häirida, kasutades sõnu, mida kasutaja ei pruugi automaatselt teada, vaid peab keskenduma ja nende tähendusele mõtlema. Sama moodi, kui veebilehel on kasutatud tundmatuid *font*-e, millest on raske aru saada, siis tekivad automaatse lugemisega probleemid. Veel mõjutab lugemist teksti suuruse muutmine, liiga väikest teksti on raske lugeda ja uuringud on näidanud, et ka suurte tähtedena, kirjutatud teksti loetakse mõningal määral aeglasemini. Liiga kirev või segane taust segab samuti kasutajate lugemist ja nad peavad rohkem keskenduma, et tekstist aru saada. Kasutajad ei saa automaatselt lugeda ja peavad keskenduma rohkem ka siis, kui teksti värv ei sobi kokku taustaga näiteks on liiga kirev. (Johnson 2010, 11, 39–41)

2.17 Wireframe

Wireframe on kasutajaliidese, kujundamise esimene samm. See on lihtne üksiku veebilehe kujutis, mis sisaldab kõiki veebilehe elemente ja nende omavahelist seondumist. *Wireframe* sisaldab endas palju selgitusi, suunates kasutaja muude arhitektuuriliste mudelite või muude disaini kujutavate, kirjeldavate dokumentide juurde. Lisaks võib see sisaldada selgitusi mõnede elementide funktsioonide kohta, kuidas vastav element töötab. Selgitusi sisaldab elementide kohta siis, kui need ei ole arusaadavad lihtsalt visuaalse vaatlemise korral. Erinevad *wireframe*-id võivad sisaldada erineva tasemega spetsiifilisust. Võib kirjeldada ka spetsiifiliste navigatsiooni elementide kindlat koosseisu. Lihtsamate projektide jaoks on piisav ühest *wireframe*-ist, aga keerulisemate projektide jaoks on vaja rohkem. *Wireframe* tuleks teha nii palju, et

saaks kujutada ära põhilised lehe esitlused, elementide paiknemise jagades need üldistesse klassidesse. Ei ole vaja luua igale vaatele oma *wireframe*. (Garett 2011, 128–129)

2.18 Windows ja *Web* vormid

Vormid on osa ekraanist, mis on kasutuses, et kuvada kasutajale informatsiooni ja et suhelda vastastikku kasutajaga. Kõige lihtsam viis kuvada vormi kasutajaliidesena on lisada sellele elemendid, mille abil saab kontrollida vormi. Vormi elemendid, millega kasutaja suhtleb saab programmeerida vastama sündmustele ehk tegevustele, mida kasutaja vormiga teeb. *Windows* vormiga saab suhelda kasutajaga ja tema arvutiga lokaalselt. *Windows* vormi ehitades, lisades sinna elemente ja programmeerides neid elemente, saab suhelda kasutajaga ja nii teada, mis tegevusi on vaja vastavas arvutis teha. (Introduction to Windows Forms)

Web-i vormis lehekülg on kasutajaliides veebi aplikatsioonile. *Web*-i vormis lehti saab ehitada kasutades näiteks *HTML*-i, *XML*-i ja näiteks *JavaScript*-i. Üks *web*-i vorm omadusi on näiteks, et see on sobiv erinevate *browser*-ite ja telefoni seadmetega. Alternatiivselt on vahest vaja sobitada oma *Web*-i vormi lehekülge, et see sobiks mõnedele teistest erinevatele *browser*-itele. (Introduction to Web ….)

Kasutajaliidese disain on *Web*-i vormi lehekülgedel jaotatud visuaalseks ja koodi osaks. Visuaalne osa on ehitatud kasutades koos näiteks *HTML*-i ja *ASP.NET*-i või siis kasutada neid eraldi. Visuaalse vormi taga olev kood suhtleb visuaalse kasutajale nähtava osaga. *Web*-i vormides on võimalik luua dünaamilise sisuga veebilehti, keerulise asetusega. Veebi aplikatsioonides on serveri osa tihti eraldi teises arvutis või operatsiooni süsteemis, kui arvuti, kus klient töötab. (Introduction to Web ….)

2.19 Vooskeemid

Vooskeeme kasutatakse, et kirjeldada protsesse, selle samme ja otsuseid, mis seal peab tegema. Vooskeemides on mitmeid erinevaid sümboleid on näiteks ümarate äärtega sümbolid ja ristküliku kujulised sümbolid. Vooskeemi sümbolitel, mis on erinevate

kujudega on kõigil oma tähendused, et anda skeemidele arusaadavust. Protsesside sammude vahelist liikumist kirjeldatakse näiteks nooltega. Ümarate äärtega sümboleid kasutatakse, et näidata protsesside algus ja lõpp-punkte, vahesammude näitamiseks kasutatakse ristkülikuid, sisend või väljund kuvatakse rööpkülikuga jne. Vooskeemide loomiseks kasutatakse tänapäeval tavaliselt erinevaid diagrammide loomise tarkvarasid. Vooskeeme saab kasutada näiteks projekti plaanimiseks või programmi ja süsteemi disainiks või töö voo dokumenteerimiseks. (Flowcharts)

3 „*UNIT4 Financial Performance Management*“ tarkvara *UI* ümber disainimine

Selle peatüki tulemused on *wireframed*, mis töö pidi produtseerima. Siin on näidatud ka, kuidas vastavaid printsiipe peatükist 2 on kasutatud.

3.1 Disainitava tarkvara üldine kirjeldus

“*UNIT4 Financial Performance Management*” on finantstarkvara, mida kasutatakse äriliste finants andmete konsolideerimiseks, raha kasutuse planeerimiseks, finants andmete salvestamiseks jne. Konsolideerimise osad on hooldus, andmete sisestamine või importimine, andmete konsolideerimine ja raportitega töötamine. Ärilise raha kasutamise protsessi osad on hooldus, andmete sisestamine ja importimine ja raportitega töötamine. Andmete sisestamise ja importimisel töötatakse finants andmetega nagu kulud, tulud jne erinevates valdkondades. Raportitega töötades kohandatakse andmeid ja töödeldakse need kasulikuks informatsiooniks.

3.2 *Liquid Web Application* disaini saavutamine

Teooria osas 2.3 on tutvustatud *Liquid software* põhimõtteid ja tutvustatud, kuidas neid rakendada *Liquid Web* aplikatsioonile. Räägitud sellest, et põhiline vahend, mida kasutada *Liquid* printsiipidega aplikatsiooni saavutamiseks veebis on "*Responsive Web Design"*. Nimelt *Liquid Web* aplikatsioon peab toetama erinevate seadmete vahelist liikumist ja andmete kuvamist. "*Responsive Web Design*," üldine kirjeldus teooria osas 2.5, aitab toetada erinevatele seadmetele sobivaid kasutajaliideseid.

Wireframe-ide ehitamisel olen kasutanud erinevaid printsiipe, mis on toodud välja töö peatükis 2. Erinevaid printsiipe on kasutatud nii, et need toetaksid responsive disaini, mille abil saavutata *Liquid Web* aplikatsioon. Kasutatud on ka mõndasid üldiseid disaini printsiipe, parema kasutaja kogemuse andmiseks, teooria osas 2.1 on mainitud

kvaliteetse/hea kasutajaliidese vajalikkusest. Erinevate *wireframe*-ide juurde peatükis 3 on lisatud seletused, miks midagi on disainitud just nii.

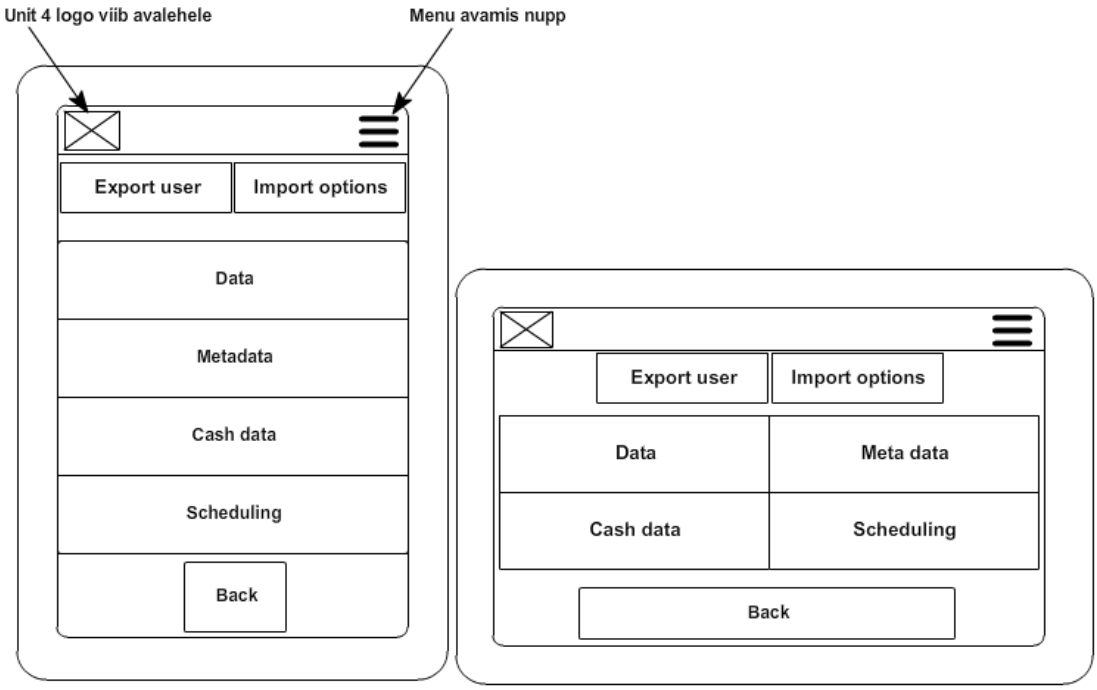
Erinevate vaadete *wireframe*-de loomisel ehitasin kõigepealt telefoni vaate. Selline lahenemine paremini realiseerida *responisve* disaini, lähenemine "Telefon ennem" on kirjeldatud teooria punktis 2.8. Selline lähenemine aitab ka kõige vajalikumad elemendid kindlaks teha, et ei kuvataks kasutajale ebavajalikku infot.

Responsive disaini jaoks on kõikide vaadete elemendid asetatud *grid*-i põhiselt, kirjeldus teooria osas 2.6. Selle järgi paigutasin elemendid, nii et need oleks võimalik koodis asetada vastavatesse koodi osadesse, teooria osas 2.6 on näide, kuidas koodi div klassides elemente hoitakse. Elemendid on paigutatud selleks nii, et need asuksid vastavates ridades ja veergudes.

Üks üldisi disaini printsiipe on näoteks teooria osas 2.14 kirjeldatud viie reegel. Antud printsiibi pea mõte on kasutajale, veebilehe navigatsioonis, võimalikult selgete valikute andmine. Selle paremaks realiseerimiseks on kasulik läbi mängida erinevad navigatsiooniga seonduvad stsenaariumid. Erinevate stsenaariumite läbimängimiseks on koostatud vooskeemide joonised lisades, mis kirjeldavad tarkvara funktsionaalsust, vooskeemidest on kirjeldatud peatükis 2.19.

3.3 Impordi osa

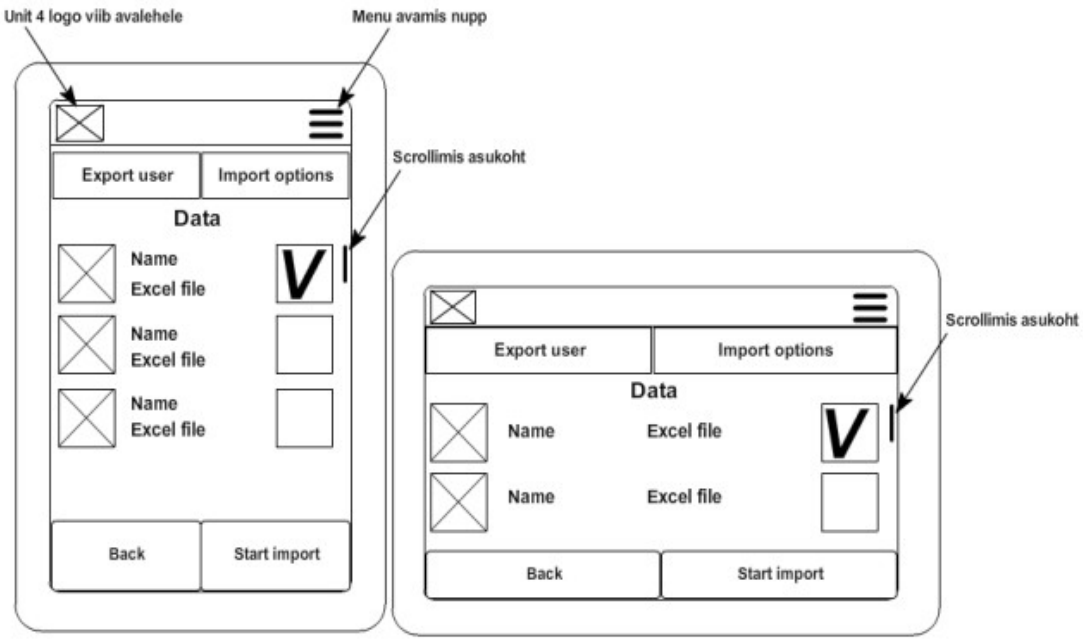
3.3.1 Impordi esileht



Joonis 8: Impordi avaleht telefonile

Peatükis 2.12 “Kohanemisvõimeline asetus erinevatel seadmetel” tõin välja, et suuremad elemendid jäävad kasutajale paremini silma ja elementide gruppeerimine aitab kasutajal informatsiooni lihtsamini *app*-ist kätte saada. Samuti tõin teoori osas 2.12 välja, et suuremad elemendid tunduvad tähtsamad, selle põhimõtte järgi (joonisel 8) on nupud *Data*, *Meta data*, *Cash data* ja *Scheduling* suuremad, kui lehe teised nupud. Nende nuppude grupeerimiseks kasutasin Gestaldi läheduse printsiipi kirjeldatud Peatükis 2.10 “Gestaldi printsiibid.” Gestaldi läheduse printsiip väidab, et objektide vaheline kaugus mõjutab, kuidas vaataja vastavaid objekte grupeerib. Objektid, mis asuvad üksteisele lähemal, aga teistest elementidest kaugemal, tunduvad olevat grupeeritud.

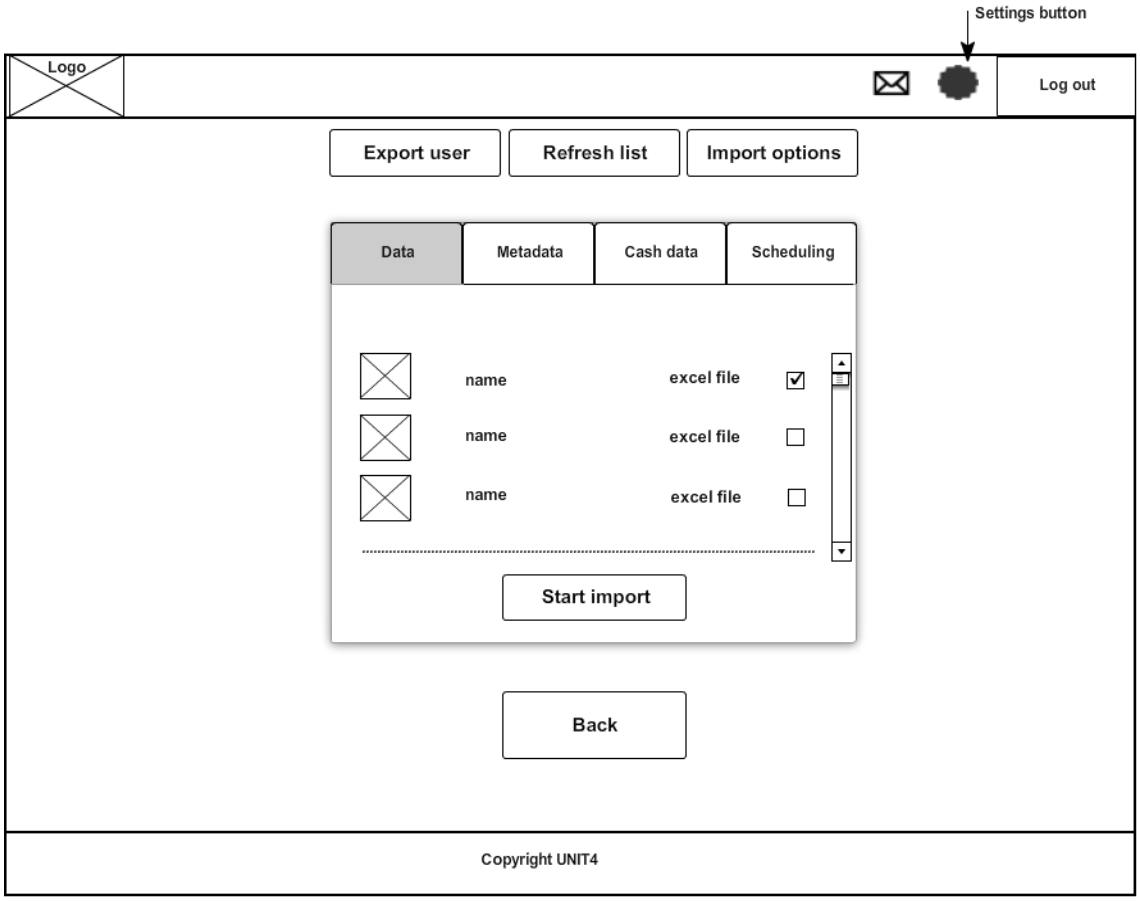
Nii joonisel 8, 9 kui 10 on elemendid asetatud r*esponsive* disaini jaoks *grid*-põhiselt ridadesse ja veergudesse. *Responsive* disain toetab "*Liquid Web Application*" printsiibil disaini. Peatükis osas 2.6 on seletatud *grid* põhisest asetusest ja "*Liquid Web Application*" peatükis 2.3.



Joonis 9: Impordi esilehe telefoni tabel

Peatükis 2.12 rääkisin ka lehe struktuuri muutuste kohta. Nimelt peaks *app*-i põhi funktsioonile jääma fookus kõikidel lehtedel. Kuigi lehe struktuur muutub natuke minnes telefonilt üle arvutisse, kuna arvutis kuvatakse ka tabelit failidega lisaks nuppudele. Telefonis mugavama kasutamise eesmärgil on tabel teisel lehel (joonis 9). Nimelt tõin peatükis 2.12 välja, et suuremaid nuppe on lihtsam kasutada, nii et, kui oleks lisanud telefoni vaatesse ka importimise tabeli siis see oleks muutnud *app*-i kasutamise ebamugavaks.

Vaatel joonisel 9 kasutasin samuti elementide grupeerimiseks Gestalti printsiipe (peatükk 2.10). Linnukeste kastid on 44x44 punkti, et neid oleks lihtsam kasutada, alla selle suuruse võib muutuda ebamugavaks (peatükk 2.12).



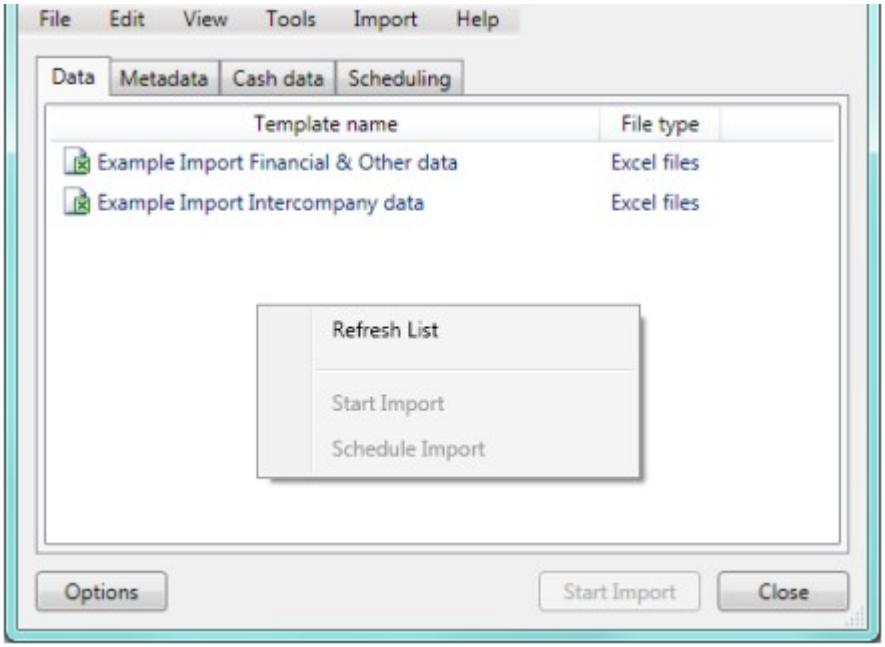
Joonis 10: Impordi esileht arvutile, tahvlile

Vähima sammu/klõpsu reegli järgi, kirjeldatud peatükis 2.13, on tabel, mis telefoni vaates oli erineval lehel (joonis 9), arvuti ja tahvli vaates (joonis 10) samal lehel, et kasutaja ei peaks lisa sammu tegema. Samuti, vähima võimaliku sammu/klõpsu põhimõte järgi, tõin ka *File* menüüst välja ja muutsin nuppudeks tegevused listi uuendamiseks (*Refresh list*), impordile lisa parameetrite valimine (*Import options* varasema disaini peal selle nimi *options*) ja suuremate õigustega kasutaja lisamine (*Export user*). Nii ei pea kasutaja otsima neid nuppe navigatsiooni menüüst, põhimõte mis tuleb, nii öelda, viie reeglist. Viie reegli järgi, kirjeldatud peatükis 2.14, ajavad mahukad navigatsiooni menüüd kasutajad segadusse.

Gestalt-i läheduse printsiibi järgi, kirjeldatud peatükkis 2.10, objektide vaheline kaugus mõjutab, kuidas vaataja vastavaid objekte grupeerib. Objektid, mis asuvad üksteisele lähemal, aga teistest elementidest kaugemal, tunduvad olevat grupeeritud. Gestalti printsiibid väidavad ka, et elemendid, mis on nii grupeeritud ei vaja piirjoontega eristamist, nii öelda kasti aladesse, et grupid oleks paremini märgatavad. Nende

põhimõtete järgi grupeerisin nupud *Refresh list*, *Import options* ja *Export user*. Samuti nupp *Back* asub teistest eraldi ja keskel olev vastava impordi valimise koht on ühte grupi paigutatud.

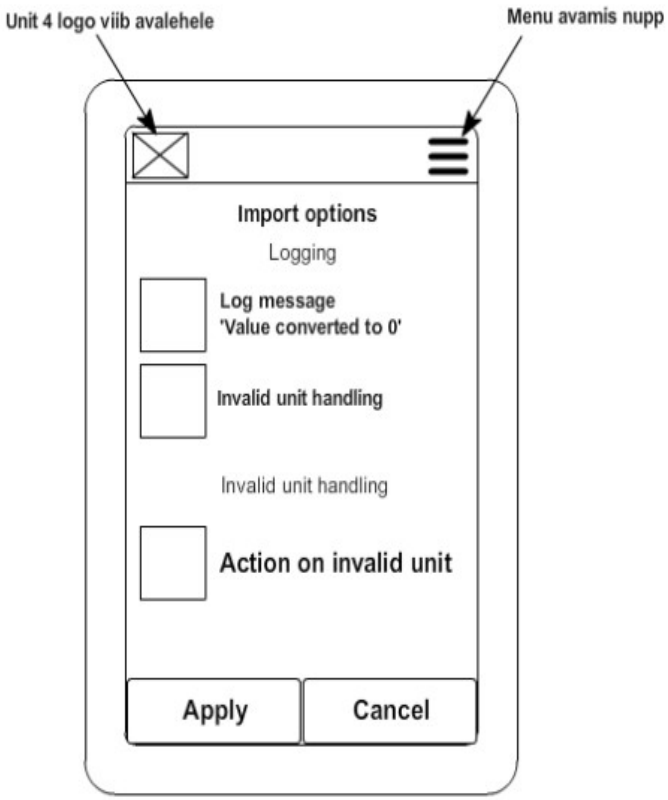
Peatükis 2.15 “Perifeerne nägemine ja disain” tuleb välja, et me märkame elemente paremini, mis on teistest erinevad, näiteks suuremad. Samuti peatükis 2.12 “Kohanemisvõimeline asetus erinevatel seadmetel” tuleb välja, et suuremad elemendid jäävad kasutajale paremini silma. Sellepärast on paljud elemendid tehtud suuremaks näiteks (joonis 10) tabeli valikud *Data*, *Metadata*, *Cash data*, *Scheduling* jne. Impordi nupp on tõstetud keskmisesse tabelisse, et oleks arusaada, et see kuulub sinna gruppi. *Shadow* on lisatud tabelile, et see oleks erinev ülejäänud veebilehe elementidest, sest import on selle lehe pea ülesanne.



Joonis 11: Impordi esilehe disain enne

Impordi esilehe disain *windows* vormil.

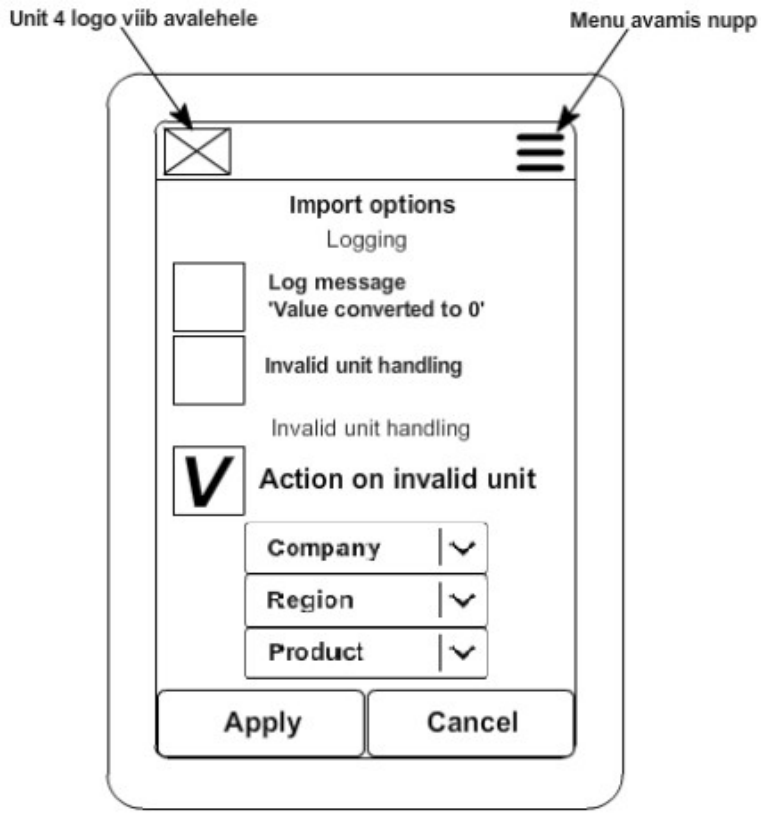
3.3.2 Impordi seaded



Joonis 12: Impordi seaded telefoni vaade

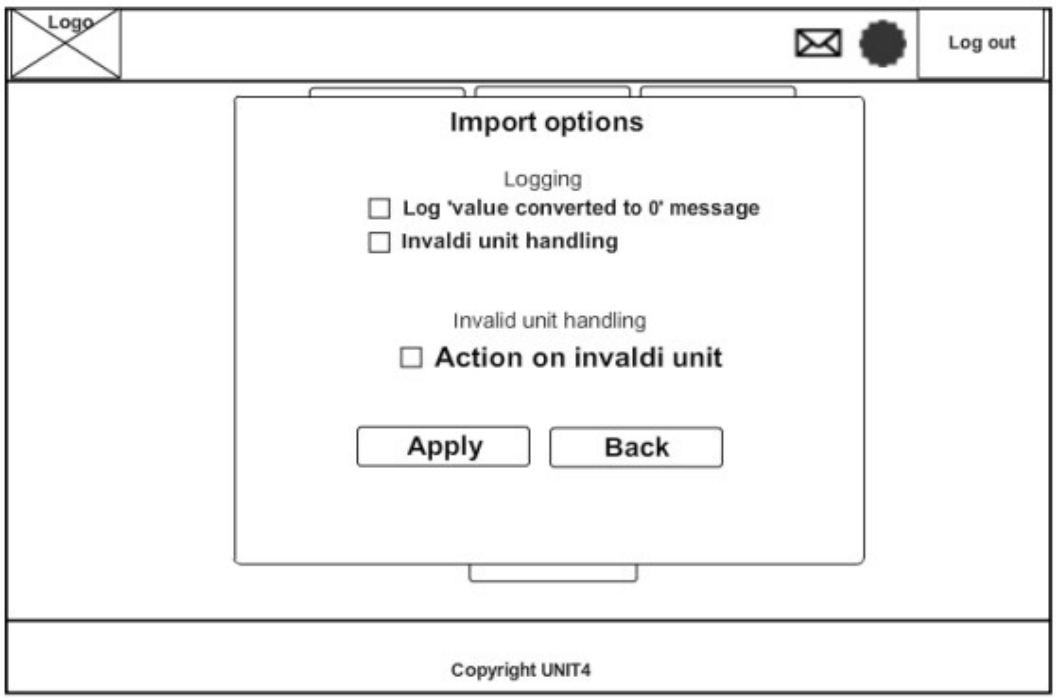
Peatükis 2.12 kasutan, antud vaatel (joonis 12), disaini printsiipi, mille kohaselt telefonis on mõistlik anda nuppudele 44 x 44 punktise suuruse, sest selliseid elemente on lihtsam kasutada. Lisaks teooria osast 2.15 kust selgub, et me näeme elemente, mis on erinevad näiteks suuremad teistest paremini, sellepärast on linnukeste panemis kastid suuremad, kui teksti elemendid. Lisaks veel visuaalse hierarhia paremaks loomiseks on antud nimed erinevatele vaate osadele need on väiksemad tekstid *Logging* ja *Invalid unit handling* ja suurem lehe pealkiri on *Import options,* see on ka suuremalt kujutatud (vastavad printsiibid kirjeldatud peatükis 2.11). Gestalti läheduse printsiipide järgi on elemendid grupeeritud, alumised nupud ja linnukeste kastid (Gestalti printsiibid peatükis 2.10)

Joonistel 12, 13, 14 kui ka 15 on *wireframe*-i elemendid asetatud r*esponsive* disaini jaoks *grid*-põhiselt ridadesse ja veergudesse. Peatükis 2.6 on seletatud *grid* põhisest asetusest. *Responsive* disain toetab "*Liquid Web Application*" printsiibil disaini, "*Liquid Web Application*" on seletatud peatükis 2.3.



Joonis 13: Impordi seaded telefoni vaade

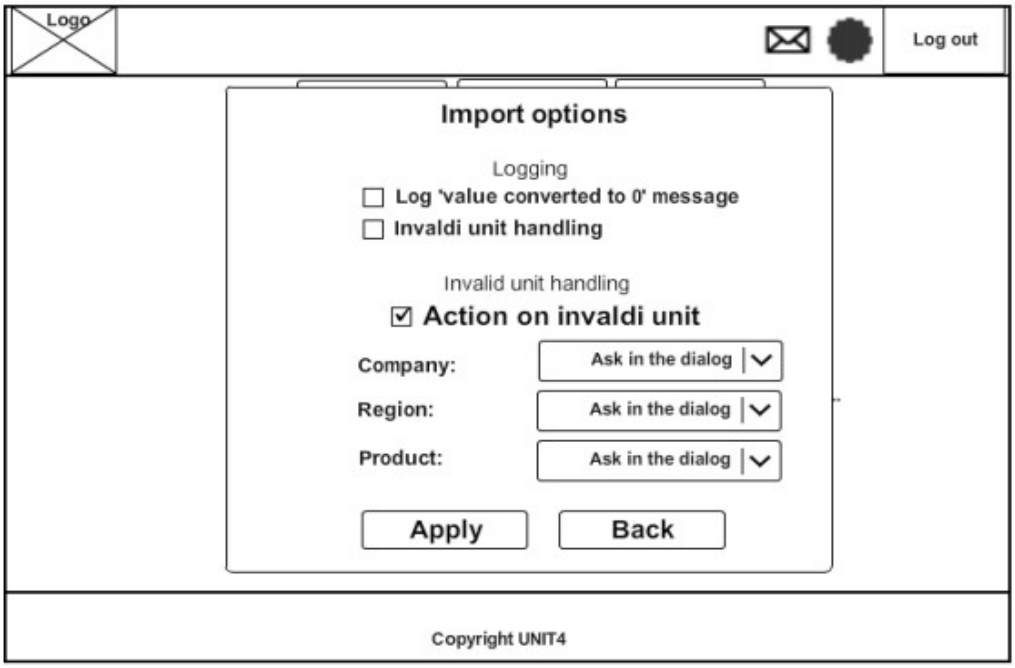
Alumised lisa valikud joonisel kolmteist, mida polnud joonisel kaksteist on nähtavad, kui linnuke on *Action on invalid unit* kastikeses. Muud elemendid muutuvad sellisel juhul väiksemaks, kuna nagu peatükis 2.15 soojus kaartide osas tuleb välja siis kasutajad ei pruugi alati kerida lehte alla ja ei pruugi olulist infot, mis lehe alumises osas asub nii leida. Nii et mahutasin kõik elemendid telefoni ekraanile muutes teised väiksemaks. Vaatel joonis kolmteist ei ole neid elemente kuvatud, et elemendid oleksid suuremad ja vähemalt 44x44 punktise suurusega (teooria osa 2.12). Nupp *Apply* oli varem *OK* (joonis 12) muutsin, et oleks arusaadavam, et rakendatakse muudatusi töö keskkonnale, sest peatükis 2.16 üks asi, mis segab kasutaja töö kiirust, kuna kasutaja ei saa automaatselt lugeda, on sõnade kasutamine, mida kasutaja ei pruugi kohe teada. Antud olukorras ei pruugi nimi *OK* anda täielikku arusaamist, mis juhtub, kui seda vajutada.



Joonis 14: Impordi seadete arvuti tahvli vaade 1

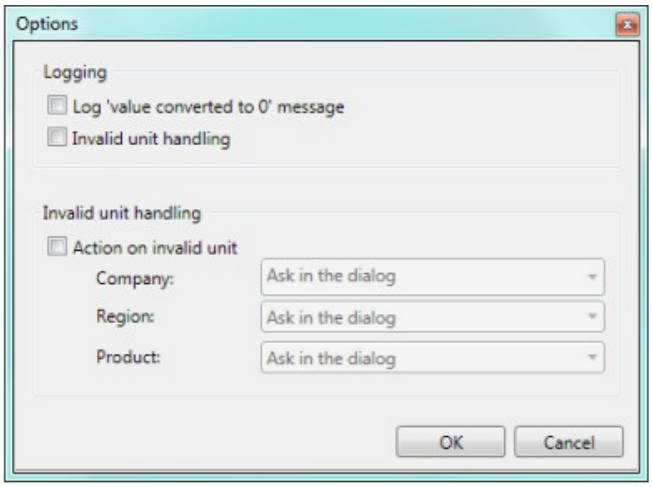
Peatükis 2.11 “Struktuurne veebileht” ütleb, et kasutajatel on mugavam lehte kasutada, kui see on struktureeritud. Struktureerida aitab näiteks igale grupile visuaalse nime panek, see aitab lehest paremini ka arusaada. Kasutades Gestalt-i printsiipe kirjeldatud teooria osas 2.10, nimelt Gestalt-i läheduse printsiipi, mis väidab, et objektide vaheline kaugus mõjutab, kuidas vaataja vastavaid objekte grupeerib. Gestalt-i läheduse printsiipi kasutades ei ole vaja elementide grupeerimiseks neid piirjoontega teistest eraldada (kirjeldus peatükis 2.10), mida on tehtud varasemas *Windows* vormis. Gestalt-i sarnasuse printsiibi (kirjeldus peatükis 2.10) järgi elemendid, mis on sarnased tunduvad olevat samuti ühes grupis.

Gestalt-i läheduse printsiipe kasutasin seadete grupeerimiseks, mis on *Logging* all ja mis on *Invalid unit handling* all. Nimed *Logging* ja *Invalid unit handling* on head grupi nime kirjeldamiseks, mis aitab lehte struktureerida. Lisaks, kuna nendel mõlematel gruppidel on kasutatud erinevaid elemente siis Gestalt-i sarnasuse printsiibi järgi tunduvad need samuti erinevates gruppides. Gestalti sarnasuse printsiibi järgi, kuna valik *Action on invalid unit* on suurem (erineb teistest), kui teised valikud , mis on *Logging* valiku all, tundub see ka erinevas grupis olevana.



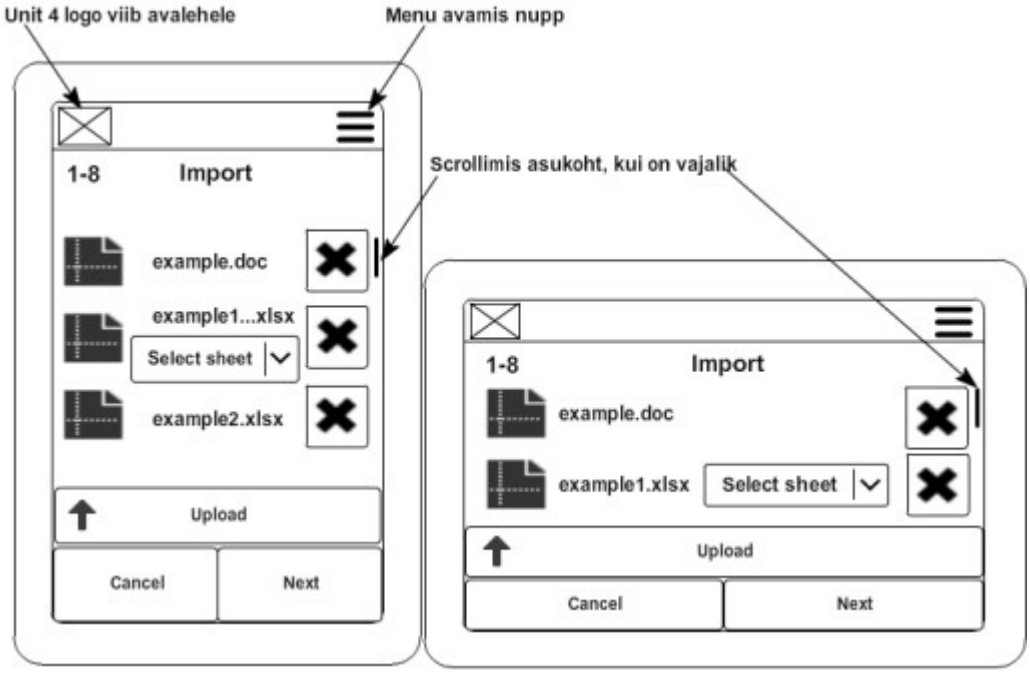
Joonis 15: Impordi seaded arvuti telefoni vaade 2

Peatükis 2.11 on mainitud ka, et kasutajad ei loe igat sõna internetileheküljel vaid lasevad silmadega üle veebilehe, otsides informatsiooni, mis on neile vajalik. Sellepärast kuna valikud, mis on grupis *Action on invalid unit* on vabatahtlikud, neid ei pea ise määrama, siis kuvatakse need ainult sellisel juhul, kui kasutaja tahab neid näha (vastav linnuke lisatud). Peatükis 2.11 üks disaini soovitusi on ka see, et erinevad elemendid jäävad kasutajale lehelt midagi otsides paremini silma. Sellepärast on ka valik *Action on invalid unit* suurem, kui muud valikud, et kasutaja seda märkaks.



Joonis 16: Impordi seadete vaade disain enne

3.3.3 Impordi samm 1

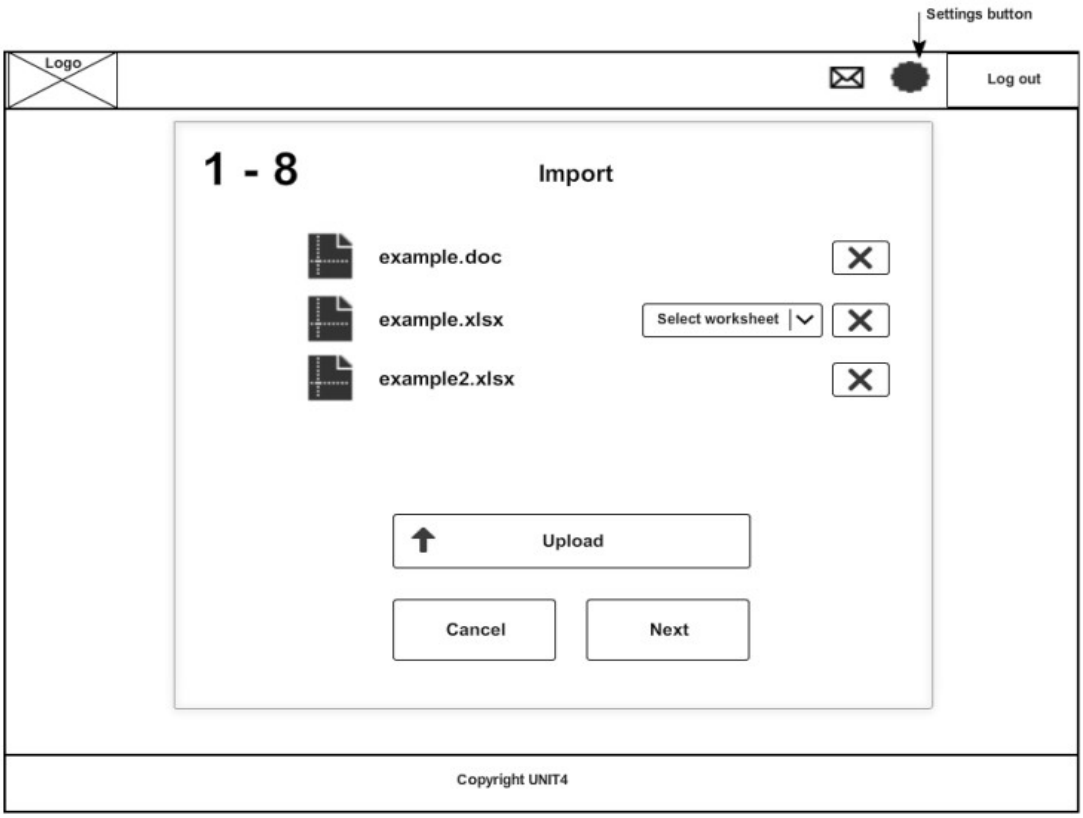


Joonis 17: Impordi samm 1 telefoni vaade

Peatükis 2.13 tuleb välja, et kasutajatel võib tekkida frustratsioon, kui nad ei leia ülesse informatsiooni, mida nad tahavad ja kui nad ei saa aru, kas nad jõuavad oma eesmärgile lähemale. Impordi esimese sammu vaates, joonised 13 ja 14 , üleval vasakus ääres on sümbol “1-8” kirjeldamaks, kui kaugel impordiga kasutaja on. Jätsin selle eelmisest disainist (joonis 19) sisse, sest see annab head tagasisidet kasutajale ja kasutaja näeb, et ta jõuab oma eesmärgile lähemale.

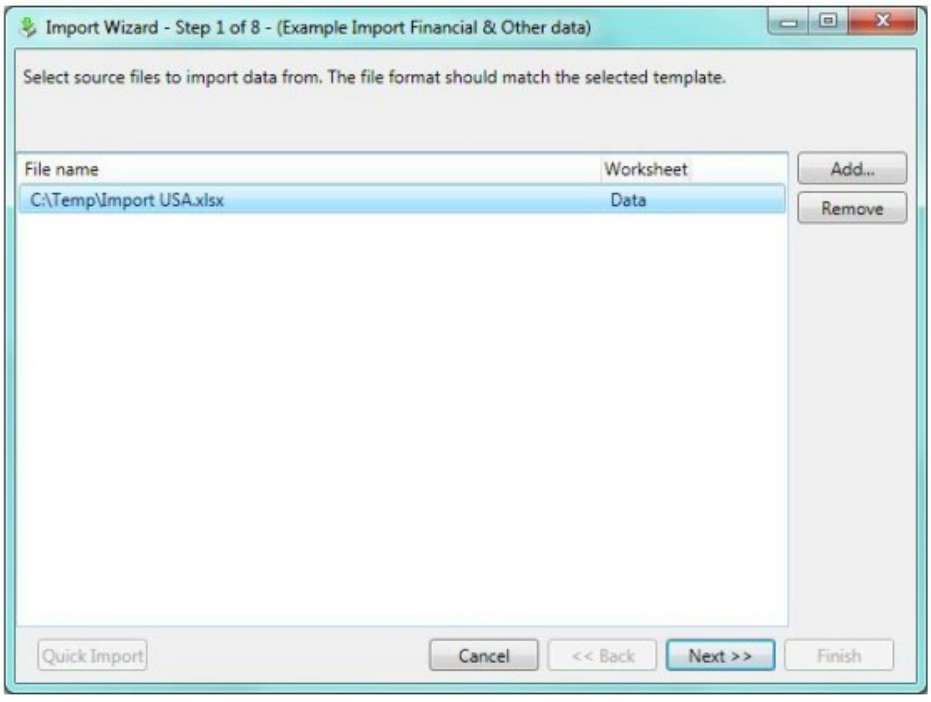
Üldine elementide struktureerimine on tehtud Gestalti printsiipide järgi, mis on kirjeldatud teooria osas 2.10. Lehe asetus on tehtud *responsive* disaini jaoks *grid*-i põhiselt, kirjeldus peatükis 2.6. Selle järgi veebilehe elemendid paigutasin, nii et need asuksid vastavates veergudes ja ridades. *Responsive* disain toetab "*Liquid Web Application*" põhimõtteid "*Liguid Web Application*" kirjeldatud peatükis 2.3.

Edasistel vaadetel hoian nuppude, mis korduvad, disaini sama ja positsiooni samuti, sest muidu võivad kasutajad nuppude asukoha muutusel harjumusest kogemata vale nuppu vajutada (peatükis 2.9 kirjeldus vastava loogika kohta). Nuppude *Back* ja *Next* asukoht jääb samaks, kuni sammuni 7, kus *Next* nupp kaob ära ja asetus on erinev.



Joonis 18: Impordi samm 1 telefoni ja arvuti vaade

Impordi sammu 1 arvuti või ka tahvli vaade (joonis 18) on disainitud sama eeltoodud loogika järgi. Ainus erinevus on see, et telefonis peaksid olema elemendid, mida kasutaja peab näiteks klõpsima, vähemalt 44x44 pikslit suured (peatükk 2.12).

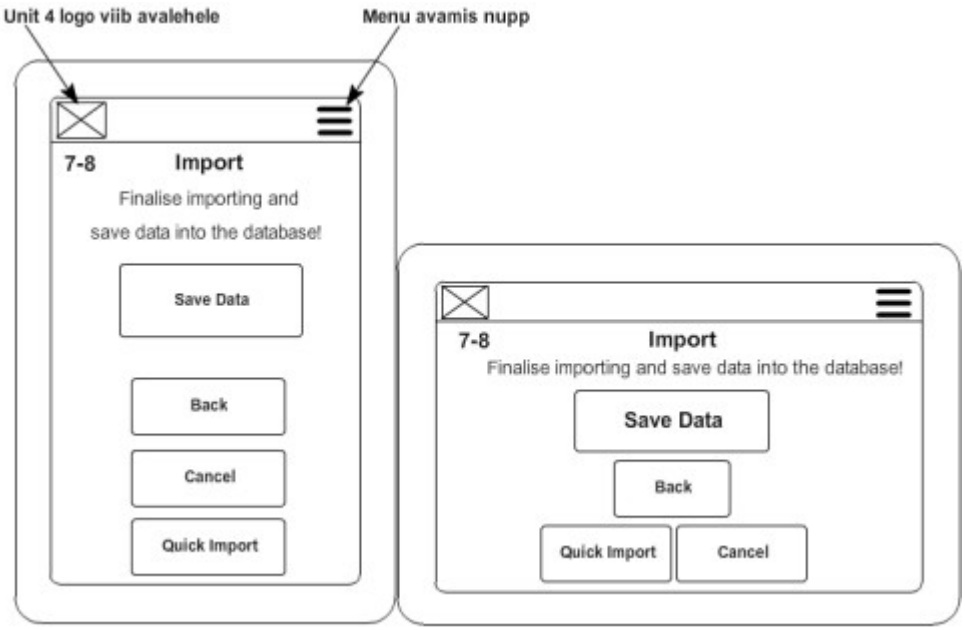


Joonis 19: Impordi samm 1 disain enne

3.3.4 Impordi sammud 2 kuni 6 ja 8

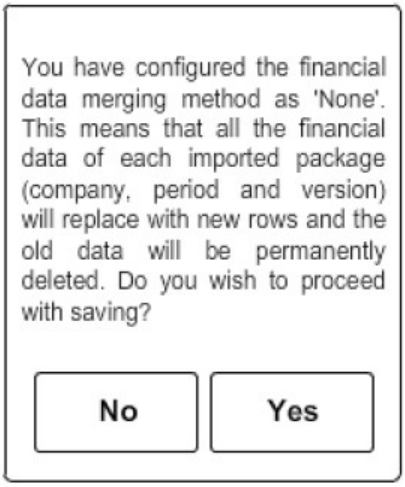
Vastavad sammud asuvad lisas 1 Vaated.

3.3.5 Impordi samm 7



Joonis 20: Impordi samm 7 telefoni vaade

Impordi sammu 1 juures mainisin, et nuppude, mis korduvad, erinevatel vaadetel, disain ja positsioon peaksid jääma samaks. Kui positsioon muutub võivad kasutajad nuppude asukoha muutusel harjumusest kogemata vale nuppu vajutada (peatükis 2.9 kirjeldus vastava loogika kohta). Nuppude *Back* ja *Next* asukoht jäid samaks, sammust 1 kuni sammuni 7, kus *Next* nupp kaob (joonis 20). Impordi sammul 7 on muudetud nuppude asetust lehel, et kasutaja ei saaks paremale alla nurka klõpsates vale nuppu vajutada. Samuti märgatakse elemente, mis on teistest erinevad, näiteks suuremad, paremini (teooria osa 2.12), sellepärast on ka nupp *Save Data* teistest nuppudest suurem. Gestalti läheduse ja sarnasuse printsiipide järgi aitab nuppu *Save Data* teistest eristada ka see, et nupud *Back, Cancel* ja *Quick Import* on ühe suurused ja asuvad üksteise suhtes samadel kaugustes ja teistest elementidest kaugemal (peatükk 2.10).



Joonis 21: Impordi sammu 7 pop-up

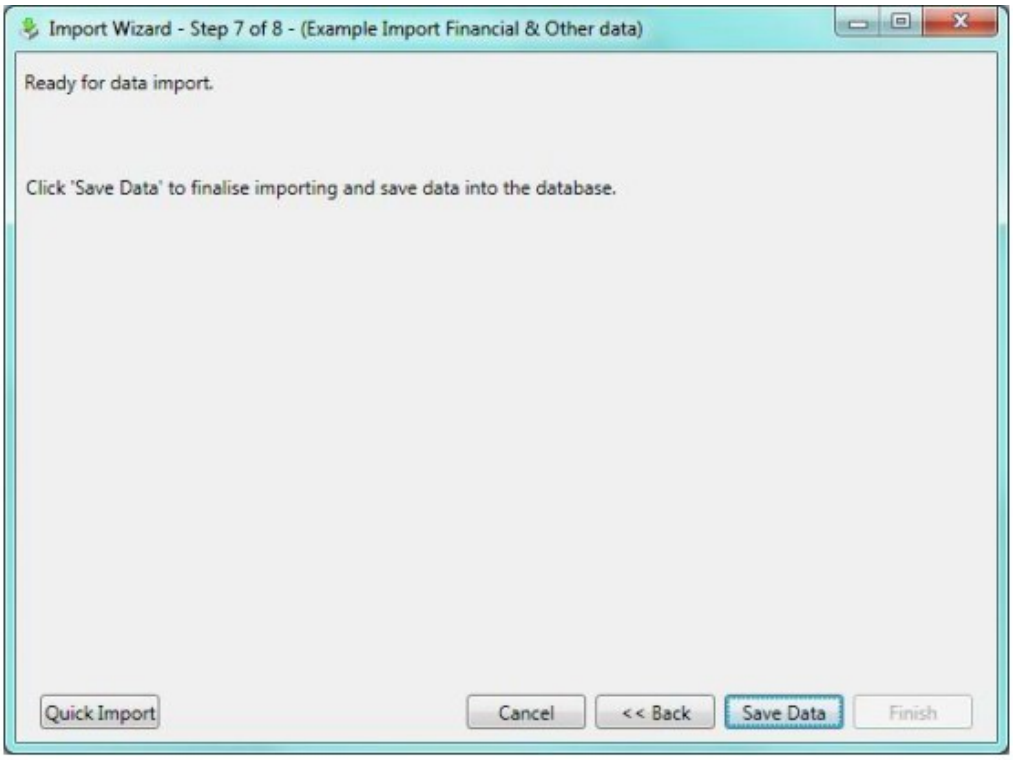
Olenevalt, kuidas on seadistatud importimis seaded võib ilmuda *pop-up* impordi sammul 7 nagu joonisel 21.

Joonis 22: Impordi samm 7 arvuti ja tahvli vaade

Arvuti ja tahvli vaate (joonised 22) struktuur ja asetus on sama, mis telefonis, et kasutajal oleks lihtsam seadmete vahetamisel veebilehe funktsioone kasutada (peatükk 2.12).

Impordi sammu 7 *wireframe*-del on elemendid asetatud *grid*-põhiselt, et toetada R*esponsive* disaini, mis omakorda toetab "*Liguid Web Application*" ehitamist. Vastavad seosed seletatud peatükis 3.2.

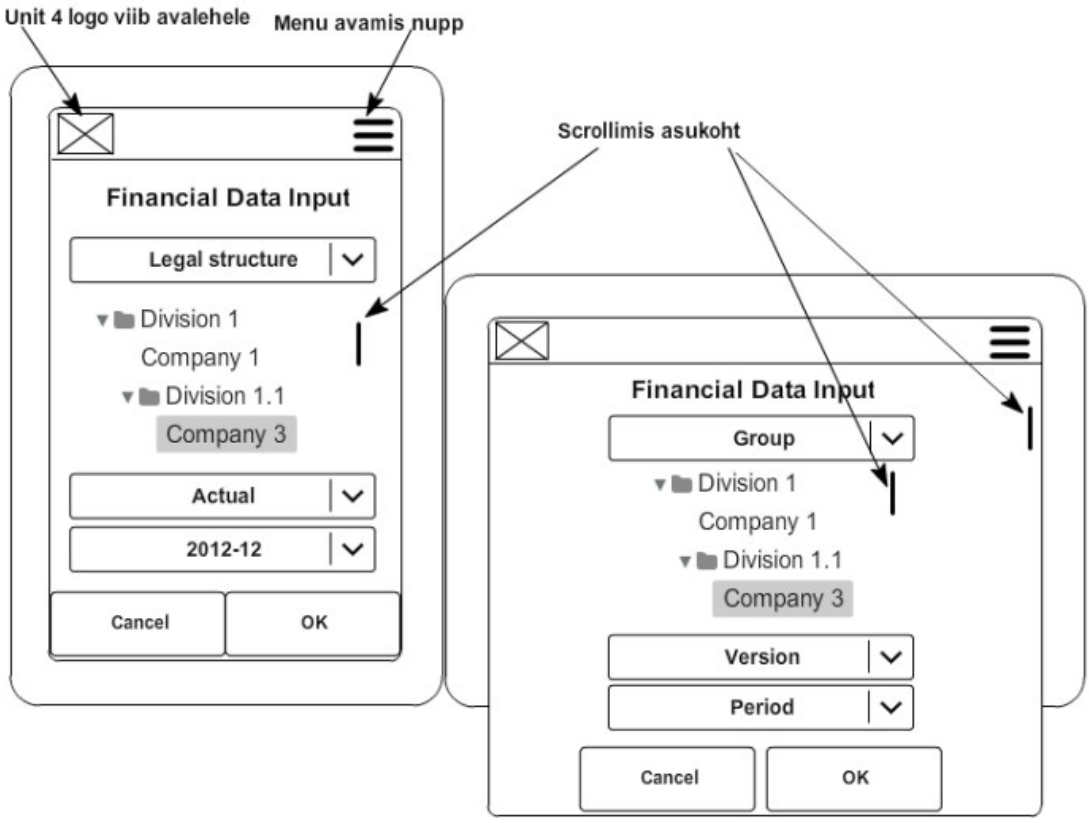
Impordi sammu 7 disain *windows* vormil



Joonis 23: Impordi sammu 7 disain enne

3.4 *Financial Data Input* osa

3.4.1 *Financial Data Input* avaleht



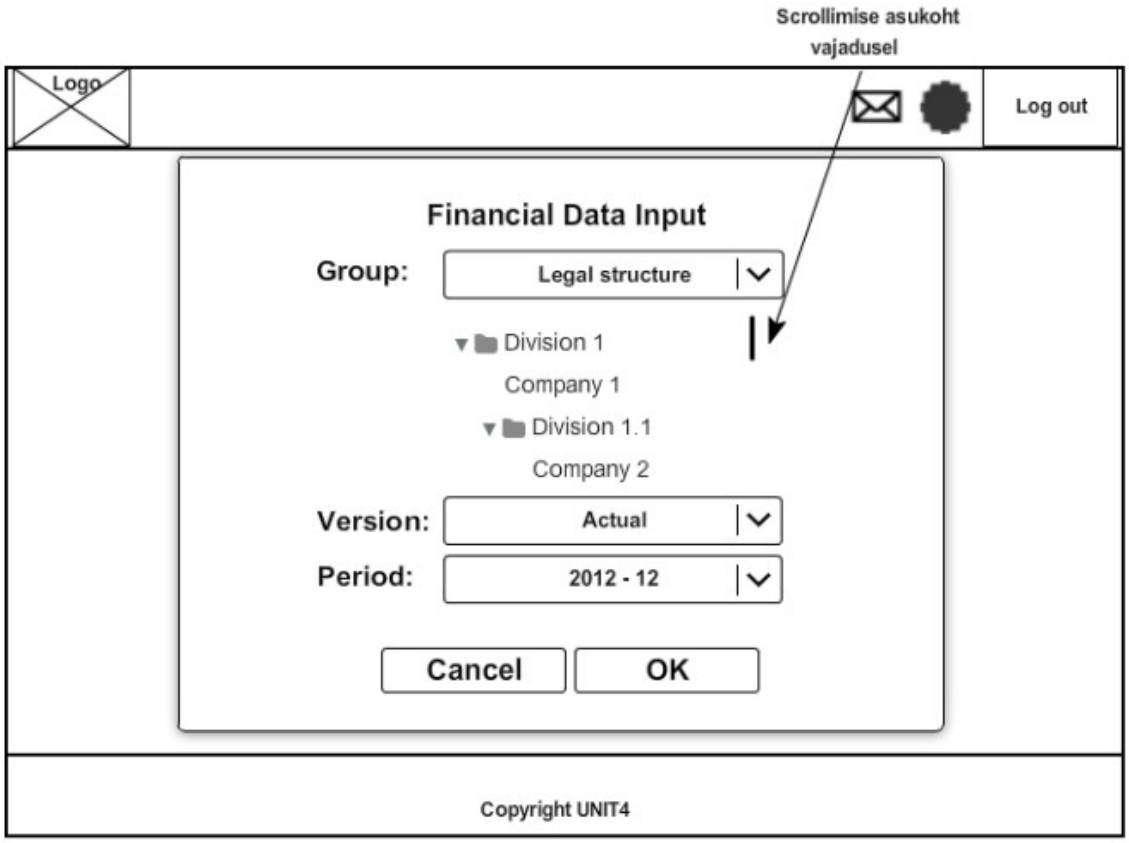
Joonis 24: *FDI* avaleht telefonile

Peatükis 2.12 "Kohanemisvõimeline asetus erinevatel seadmetel" tõin välja, et suuremad elemendid jäävad kasutajale paremini silma ja elementide gruppeerimine aitab kasutajal informatsiooni lihtsamini *app*-ist kätte saada.

Lisaks sellele on suuremaid elemente mugavam telefonis kasutada. Peatükis 2.12 on kirjeldatud, et telefonis on mõistlik anda interaktiivsetele elementidele vähemalt 44x44 piksline suurus.

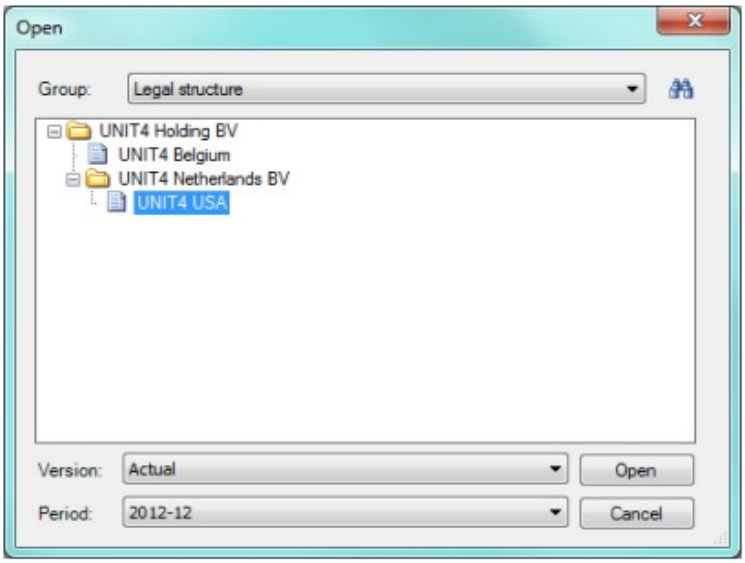
Vaatel joonisel 24 on elementide grupeerimiseks kasutatud Gestaldi läheduse printsiipi kirjeldatud peatükis 2.10 "Gestaldi printsiibid." Üks põhimõte, mis seal välja tuleb on see, et erinevates gruppides olevaid elemente ei pea piirjoontega eraldatama kastidesse vaid piisab visuaalsest joondamisest. Varasemal disainil on kasutatud piirjoontega kastide struktuuri joonis 26 uuel disainil mitte (joonised 24 ja 25). Uuel disainil on erinev ka elementide asetus, *grid*-põhine (teooria osast 2.6).

Arvuti ja tahvli vaadetel (joonis 25) on struktuur sama, mis telefoni vaatel (joonis 24), et vältida segadust, kui kasutajad liiguvad telefonilt arvutisse või vastupidi. Nimelt teooria osas 2.12 tuleb välja, et veebilehe või siis *app*-i põhifunktsioonile peaks jääma fookus kõikides keskkondades. Seal on ka mainitud, et tähtis on ka vältida suuri muudatusi *app*-i struktuuris.

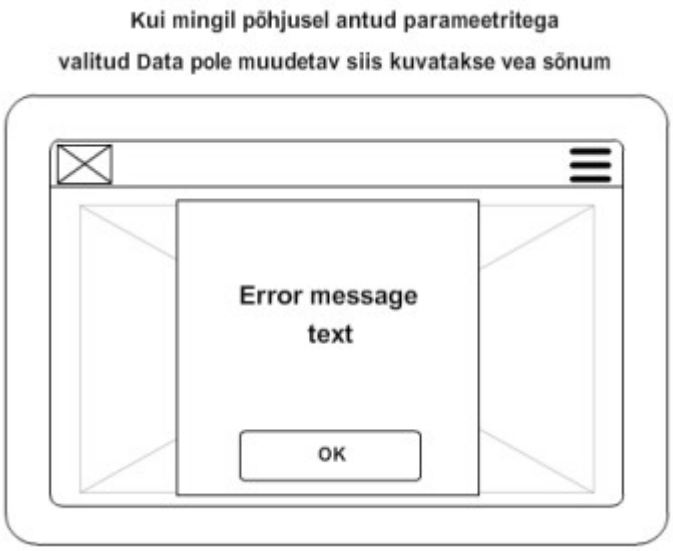


Joonis 25: *FDI* esileht arvuits, tahvlil

Financial Data Input osa esileht *windows* vormil.

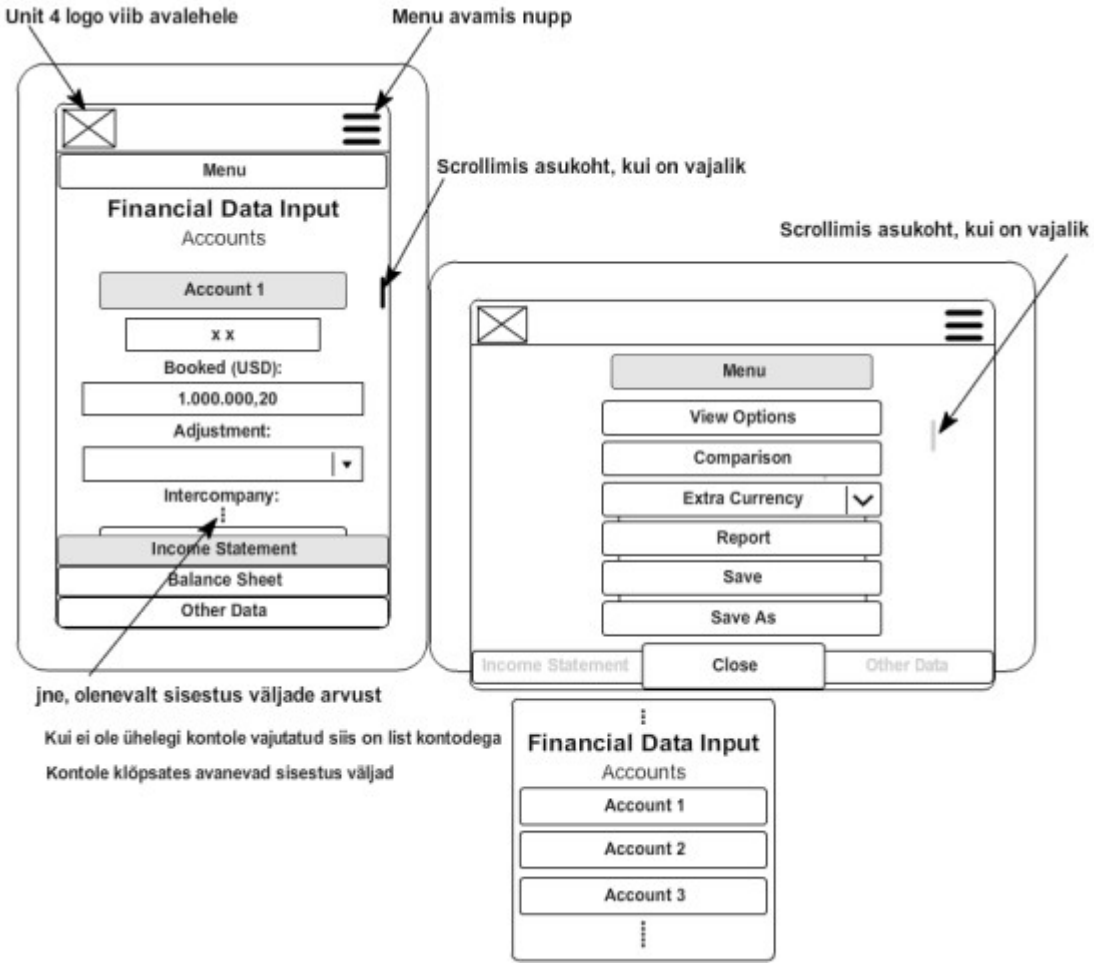


Joonis 26: *FDI* avalehe disain enne



Joonis 27: *FDI* avalehe vea sõnum telefonis

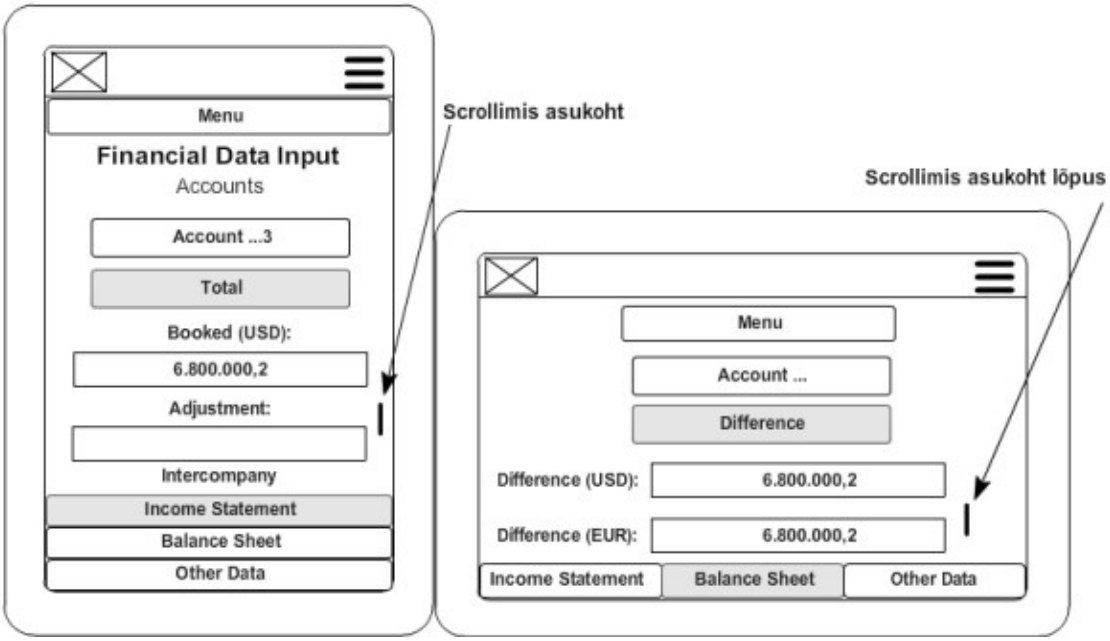
3.4.2 *FDI* tabelid andmetega töötlemiseks



Joonis 28: *FDI* andmete töötlus tabel ja avatud *Menu* vaade telefonil

Kuna vaadete *wireframe*-de loomisel ehitasin kõigepealt telefoni vaated siis olen antud vaate (joonised 28 ja 29) tabelite asetust muutnud (varasem disain joonis 30). Peatükis 2.8 “Telefon ennem” on kirjeldatud vastav printsiip. Elementide joondamiseks on kasutatud Gestalti printsiipe, peatükk 2.10.

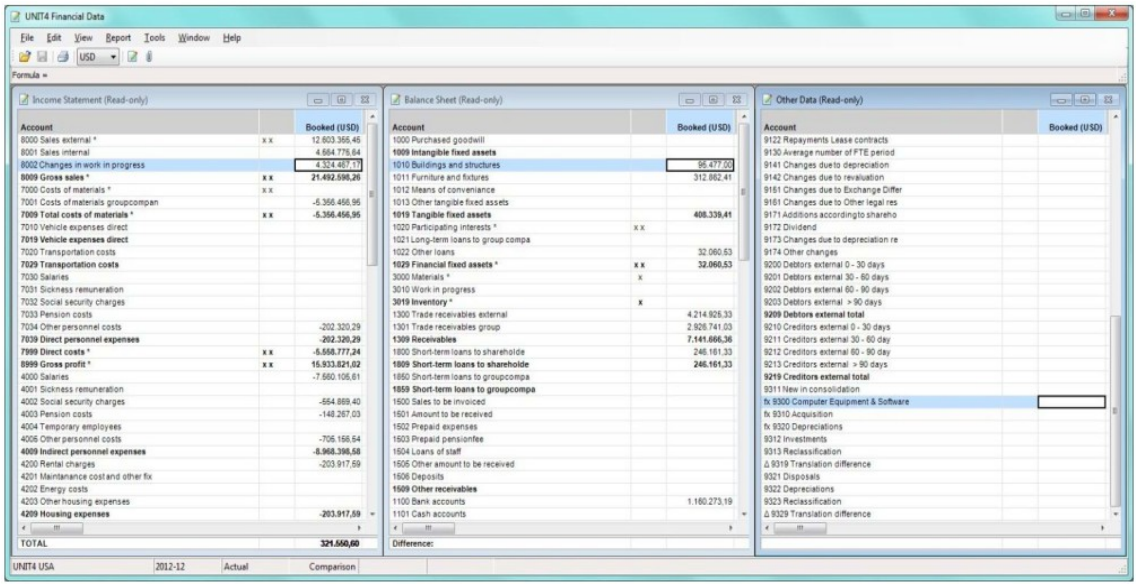
Joonisel 28 parempoolsel telefoni vaatel on kirjeldatud elemendid, mis on asetatud nupu *Menu* alla. Nii peab kasutaja tegema vähem samme/klõpse, et jõuda vastava seadeni (“Kolme sammu/klõpsu reegel” peatükk 2.13). Menüüs asuvad ka valikud, mis on kõik ülesande põhised. Selline paigutus, sest peatükk 2.14 “Viie reegel” on mainitud ära ka, et hea on jagada navigatsiooni tüüpi valikud ja ülesande tüüpi valikud erinevatesse menüü osadesse.



Joonis 29: *FDI* andmete töötlus tabeli viimased väljad telefonil

Jooniste 28 ja 29 vaadete elementide asetuses on lähtutud *grid*-põhisest asetusest (peatükk 2.6).

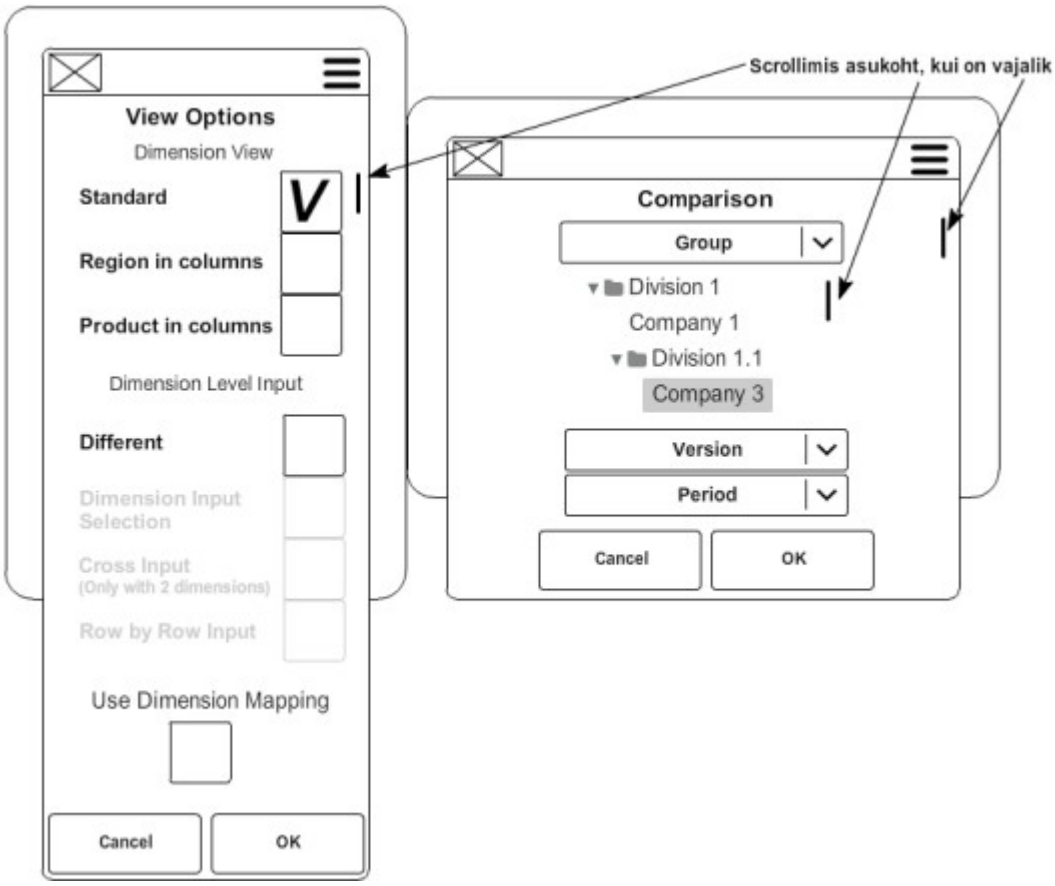
Arvuti vaade andmete töötlus tabelist on lisades.



Joonis 30: *FDI* Andmete töötlus tabeli disain enne

Disain enne.

3.4.3 *FDI View options* ja *Comparison*



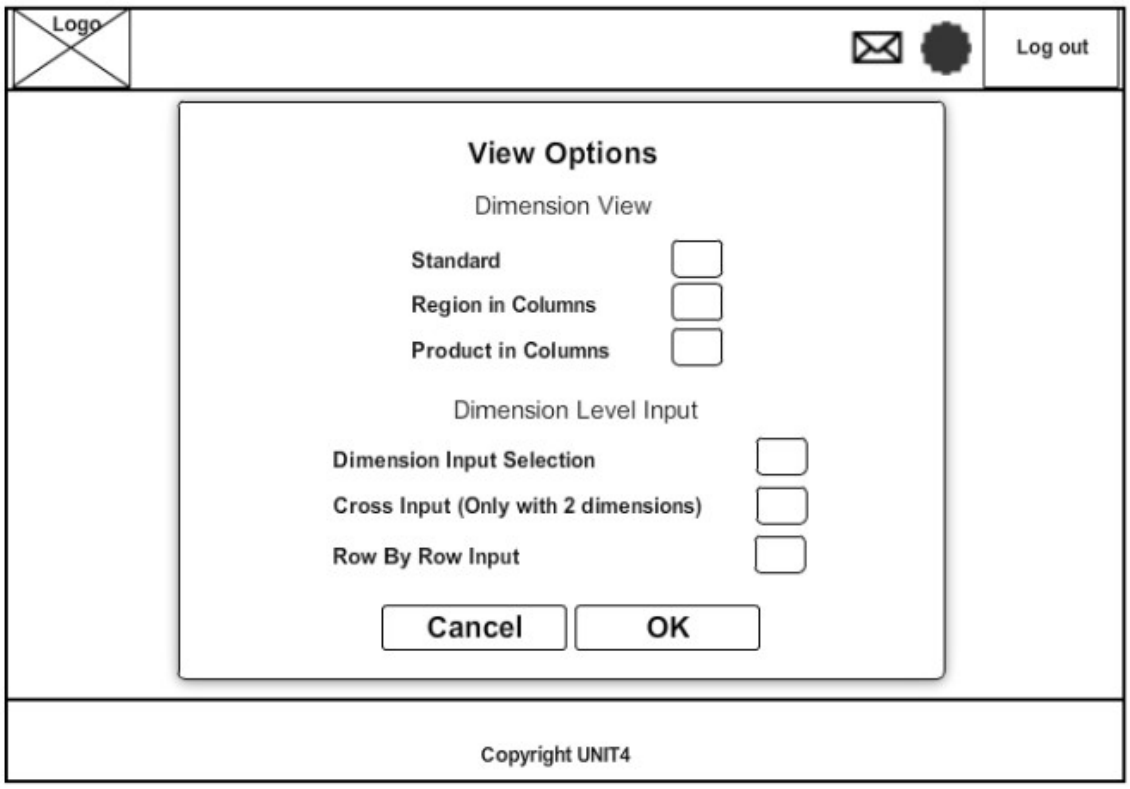
Joonis 31: *FDI View Options Comparison*

Vaadetel kujutatud joonistel 31, 32 on kujutatud vastavad valikud joonise 28 vaate menüüst (*Comparison* ja *View Options*). Valiku *Comparison* arvuti ja tahvli vaade on sama struktuuriga, mis vaade joonisel 25.

Kasutades *Standard* valikut tekib *Dimension Level Input*-ile juurde valik *Different* (joonis 31).

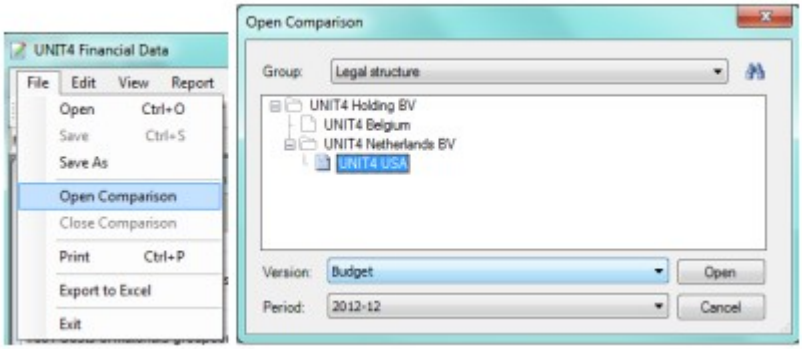
Vaate joonisel 31 disainimiseks on kasutatud printsiipi peatükist 2.12 "Kohanemisvõimeline asetus erinevatel seadmetel," kus tuuakse välja välja, et suuremad elemendid jäävad kasutajale paremini silma ja elementide gruppeerimine aitab kasutajal informatsiooni lihtsamini *app*-ist kätte saada. Lisaks sellele on võetud arvesse, et suuremaid elemente on mugavam telefonis kasutada. Peatükis 2.12 on kirjeldatud, et telefonis on mõistlik anda interaktiivsetele elementidele vähemalt 44x44 piksline suurus.

Üldiselt on struktrueerimiseks kasutatud Gestalti läheduse ja sarnasuse printsiipe, peatükk 2.10 ja visuaalselt gruppideks aitab jagada ka lühikeste alapealkirjade andmine, nagu joonise vasakpoolsel vaatel on *Dimension View* ja *Dimension Level Input*. Printsiip alapealkirjade kasulikkuse kohta, visuaalse hierarhia loomisel on peatükis 2.11. Antud vaate erinevatel seadetel olevate vaadete elementide asetusel on *responsive* disaini jaoks, mis toetab "*Liquid Web Application*" disaini printsiipe, lähtutud *grid*-põhisest asetusest (antud seose seletus peatükis 3.2).



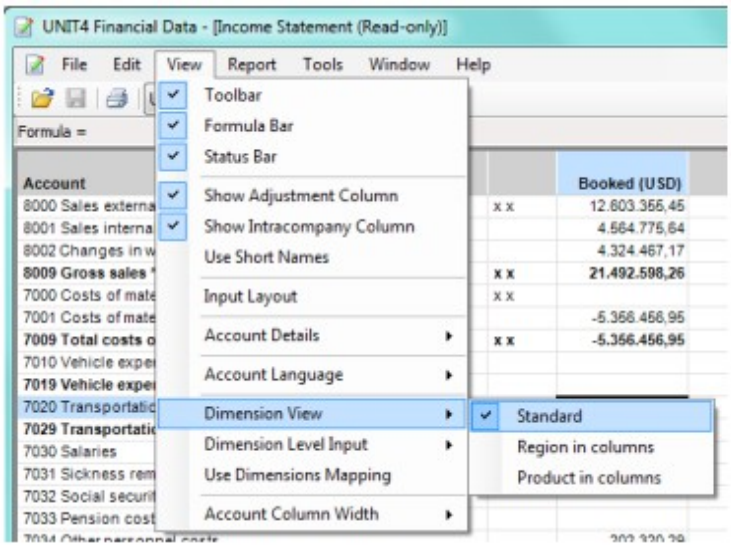
Joonis 32: *FDI View options* arvuti, tahvli vaade

Comparisoni avamine *windows* rakenduse disainis



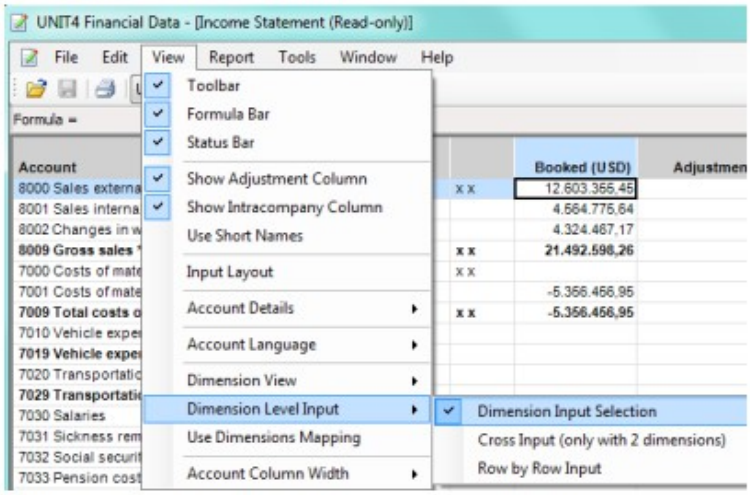
Joonis 33: *Comparison*-i avamine varasemas disainis

View optionsi dimension view avamine varasemas disainis



Joonis 34: *FDI Dimension view* avamine eelmises disainis

Eelmises disainis *Dimension Level Input* valik



Joonis 35: *FDI Dimension Level Input* eelmises disainis

4 Tulemused ja soovitused edasiarenduseks

Antud bakalaureusetöö tulemused on *wireframe*-id "*UNIT4 Financial Performance Management*" osade "*Import*" ja "*Input - Financial, Other & Intercompany data*" uute vaadete ehitamiseks veebi vormiks. "*Liquid Web Application*" disain (kirjeldus peatükis 2.3) *wireframe*-idele on saavutatud *responsive* disaini abil (kirjeldus peatükis 2.5), mis aitab veebilehti teha erinevatele seadmetele sobivaks. Töös valmisid ka mudelid (vooskeemid) tarkvara vastavate osade funktsionaalsustest, mis aitasid vaadete arendamist ja süsteemi funktsionaalsuse mõistmist.

Antud bakalaureusetöö edasiarendusena saaks alustada, toodetud vaadete *wireframe*-de põhjal, rakenduse veebiversiooni arendamist. Abiks rakenduse funktsionaalsuse mõistmisele tulevad ka loodud mudelid (vooskeemid) rakenduse funktsionaalsuse kohta. Edasisel arendamisel tuleks rakendust testida reaalsete kasutajate peal ja seejärel teha sellele parandusi vastavalt testimis tulemustele. Näiteks on võimalik läbi viia n-ö soojus kaartide testimine (kirjeldus peatükis 2.15).

5 Kokkuvõte

Antud bakalaureusetöö eesmärk oli ehitada ümber tarkvara "*UNIT4 Financial Performance Management*" osade "*Import*" ja "*Input - Financial, Other & Intercompany data*" disain w*indows* vormilt veebi vormile. Töö pidi produtseerima uued *wireframed*, mille alusel saaks uued vaated ehitada. Uue veebilehe disain pidi olema selline, et see vastaks "*Liquid Web Application*" loogika disainimist mõjutavale printsiibile, mille kohaselt veebileht peaks olema kohanduv erinevatele seadmetele. *Liquid Web Application* loogika saavutamiseks on kasutatud *Responsive Web Design* printsiipe. Lisaks pidi koostama ka mudelid tarkvara disainitavate osade kohta, mis aitaksid koostada *wireframe*-e.

Töö tulemused olid vajaminevad *wireframe*-id "*UNIT4 Financial Performance Management*" osade "*Import*" ja "*Input - Financial, Other & Intercompany data*" uute vaadete ehitamiseks veebi vormiks. Uuringu osas olevate *wireframe*-dega koos on ka vastavad seletused, miks need on tehtud just nii ja mis printsiipe on kasutatud. Valmisid ka mudelid (vooskeemid) tarkvara vastavate osade funktsionaalsuse kohta, mis aitasid kaasa vaadete arendamisele ja süsteemi funktsionaalsuse mõistmisele.

Antud bakalaureusetöö edasiarendusena oleks võimalik alustada toodetud vaadete *wireframe*-de põhjal rakenduse veebiversiooni arendamist. Abiks rakenduse funktsionaalsuse mõistmisele tulevad ka loodud mudelid (vooskeemid) rakenduse funktsionaalsuse kohta. "*Liquid Web Application*" disain on saavutatud *responsive* disaini läbi, mis aitab veebilehti teha erinevatele seadmetele sobivaks. Edasisel arendamisel tuleks rakendust testida reaalsete kasutajate peal ja seejärel teha sellele parandusi vastavalt testimis tulemustele.

Lõputöö autorile andis töö tegemine juurde uusi teadmisi erinevate disainimis printsiipide ja lähenemiste kohta. Vastavate printsiipide kasutamine süvendas autori teadmisi nendest ja aidates paremini mõista kasutatud printsiipide olemust.

Kasutatud kirjandus

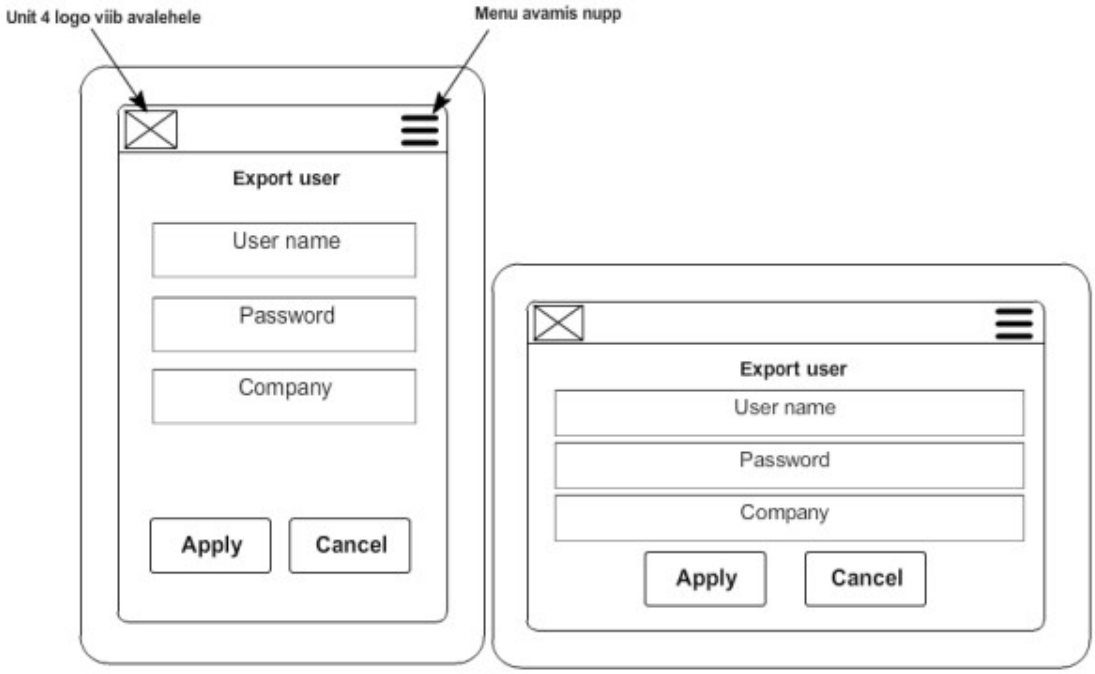
1. Akiki, Pierre A.; Bandara, Arosha K.; Yu, Yijun (2015). Adaptive model-driven user interface development systems. [WWW] http://oro.open.ac.uk/39809/1/Akiki_Bandara_Yu_ACMCSUR2014.pdf (14.03.2016)
2. Mikkonen, T., Systä, K., Pautasso, C. (2015). Towards Liquid Web Applications [WWW] http://design.inf.usi.ch/sites/default/files/biblio/2015_ICWE-liquid-web-app.pdf (09.05.2016)
3. Marcotte, E. (2011). Responsive Web Design : Brief books for people who make websites : A Book Apart. [WWW] http://www.tapan.am/files/Marcotte_E._-_Responsive_Web_Design (A_Book_Apart_Brief_books_for_people_who_make_websites)_-_2011.pdf (09.05.2016)
4. Johnson, J. (2010). Designing with the Mind in Mind : A Simple Guide to Understanding User Interface Design Rules : Morgan Kaufmann Publishers. [WWW] http://ebook.eqbal.ac.ir/Web-Design/Designing%20with%20the%20Mind%20in%20Mind%20Simple.pdf (09.05.2016)
5. iOS Developer Library. [WWW] https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/LayoutandAppearance.html#//apple_ref/doc/uid/TP40006556-CH54-SW1 (09.05.2016)
6. Porter, J. (2003). Testing the Three-Click Rule. [WWW] https://articles.uie.com/three_click_rule/ (09.05.2016)
7. Zeldman, J. (2001). Taking Your Talent to the Web : A Guide for the Transitioning Designer : New Riders Publishing. [WWW] http://takingyourtalenttotheweb.com/Taking%20Your%20Talent%20to%20the%20Web.pdf

8. Hergul, S. Understanding Simple Heat Maps for Smarter UI Design. [WWW] https://studio.uxpin.com/blog/understanding-simple-heat-maps-smarter-ui-design/ (09.05.2016)
9. Research-Based Web Design & Usability Guidelines. [WWW] http://www.usability.gov/sites/default/files/documents/guidelines_book.pdf (09.05.2016)
10. Introduction to Windows Forms. [WWW] https://msdn.microsoft.com/en-us/library/aa983655(v=vs.71).aspx (09.05.2016)
11. Flowcharts. [WWW] https://www.smartdraw.com/flowchart/
12. Introduction to Web Forms Pages. [WWW] https://msdn.microsoft.com/en-us/library/65tcbxz3(v=vs.71).aspx (09.05.2016)
13. Garett, Jesse J. (2011). The Elements of User Experience : User-Centered Design for the Web and Beyond. 2nd ed : New Riders [WWW] https://bpv-tese.googlecode.com/hg/src/referencias/Garrett2011%20-%20The%20Elements%20of%20User%20Experience%202nd%20edition.pdf (27.04.2016)

Lisa 1 – Vaated

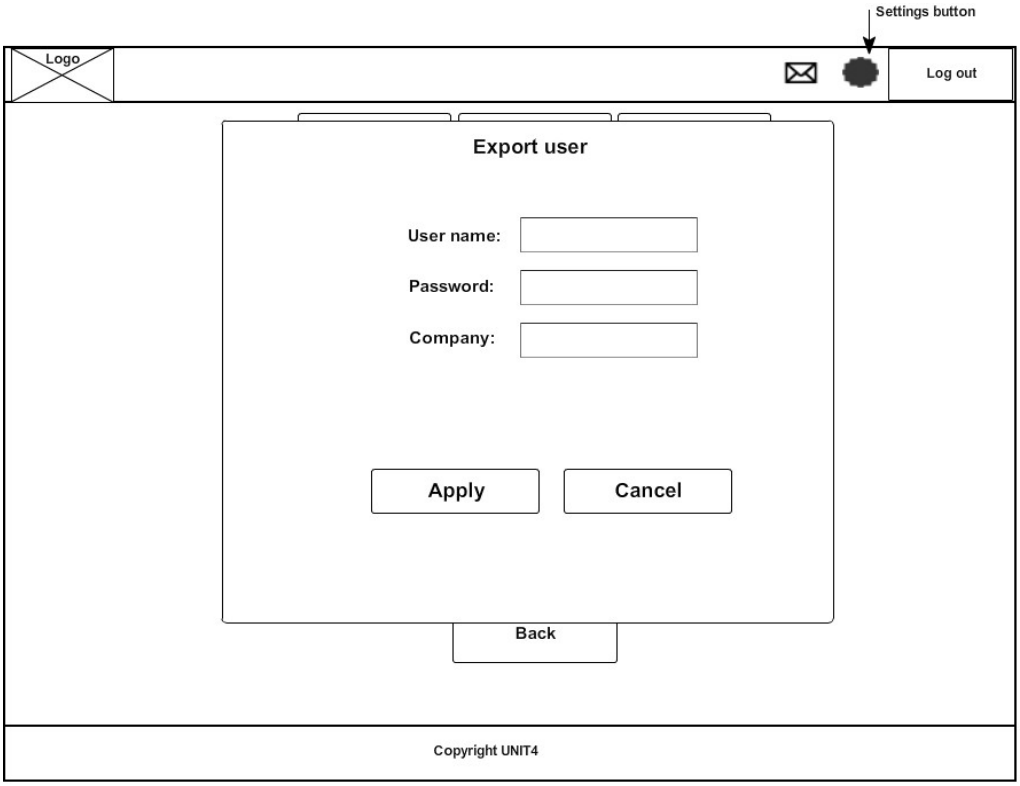
Import lisa kasutaja

Import lisa kasutaja – telefoni vaade.



Joonis 36: Import lisa kasutaja telefoni vaade

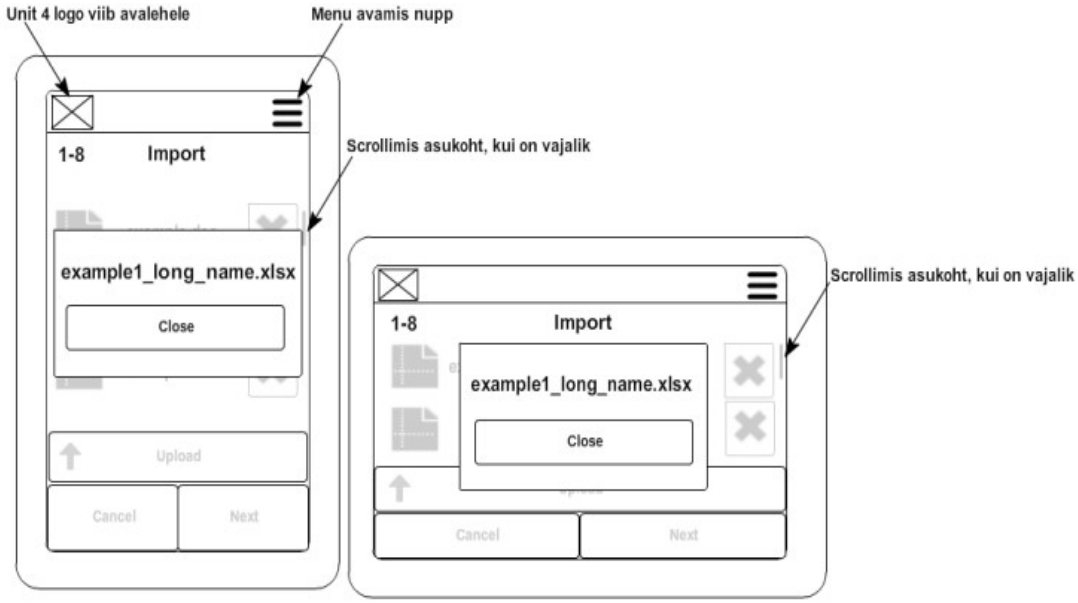
Import lisa kasutaja – arvuti vaade.



Joonis 37: Import lisa kasutaja arvuti vaade

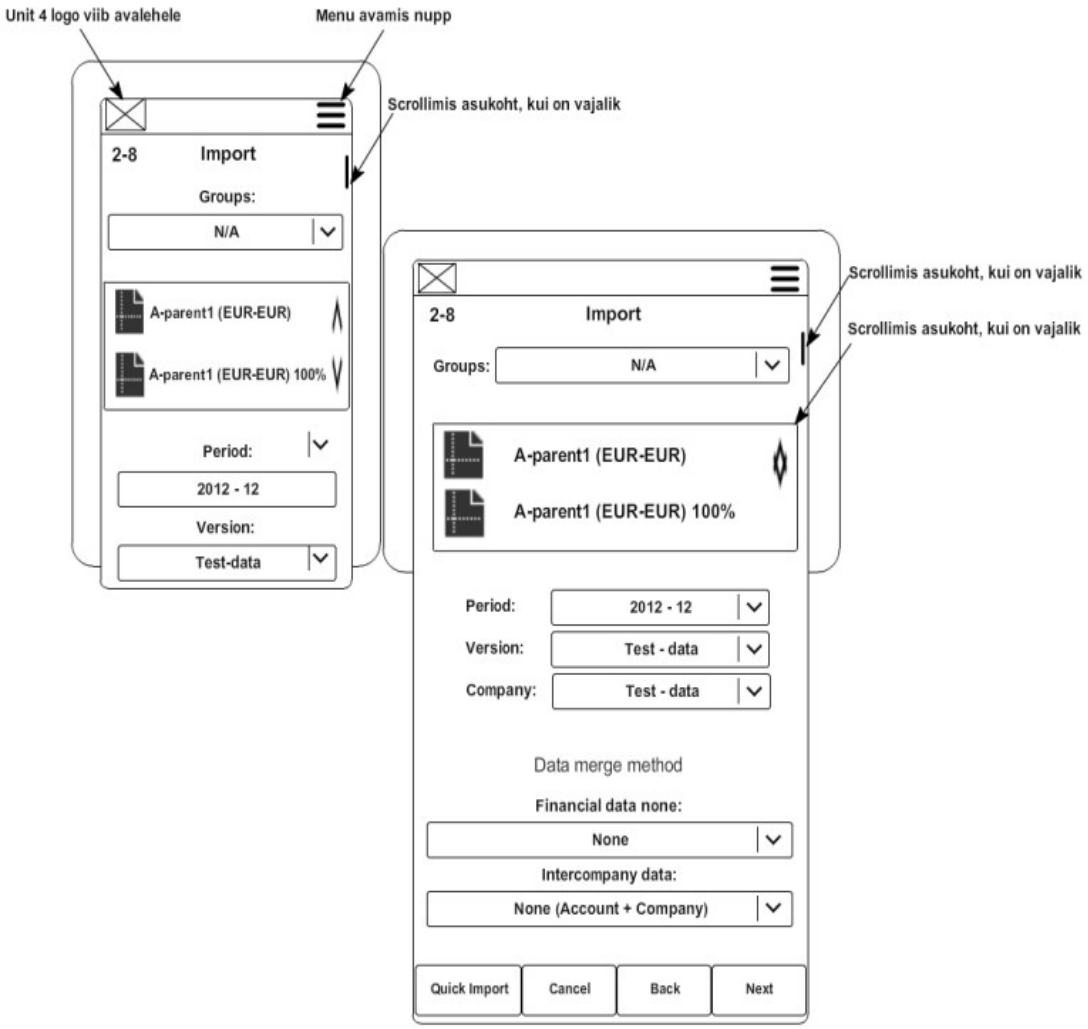
Pikad nimed telefonis

Pikale nimele saab klõpsata telefonis ja seejärel kuvatakse täispikkuses nimi.

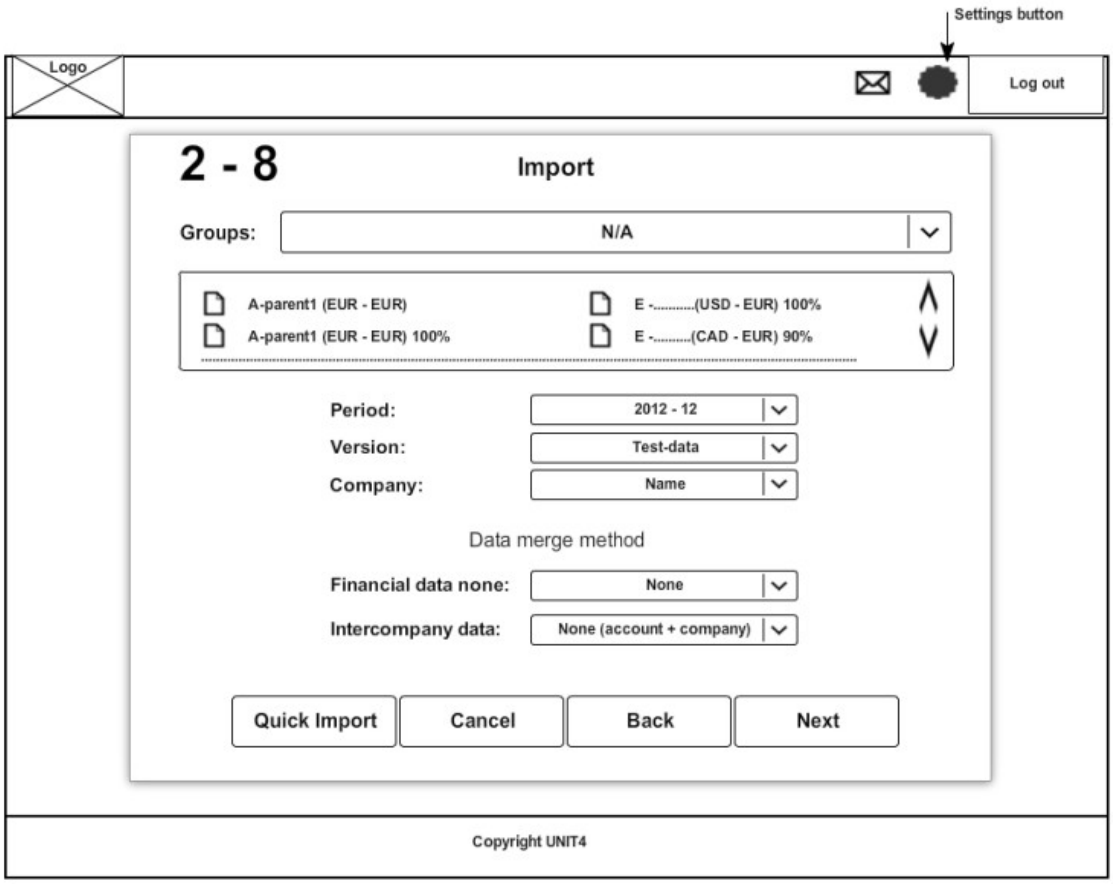


Joonis 38: Pikad nimed telefonis

Impordi samm 2

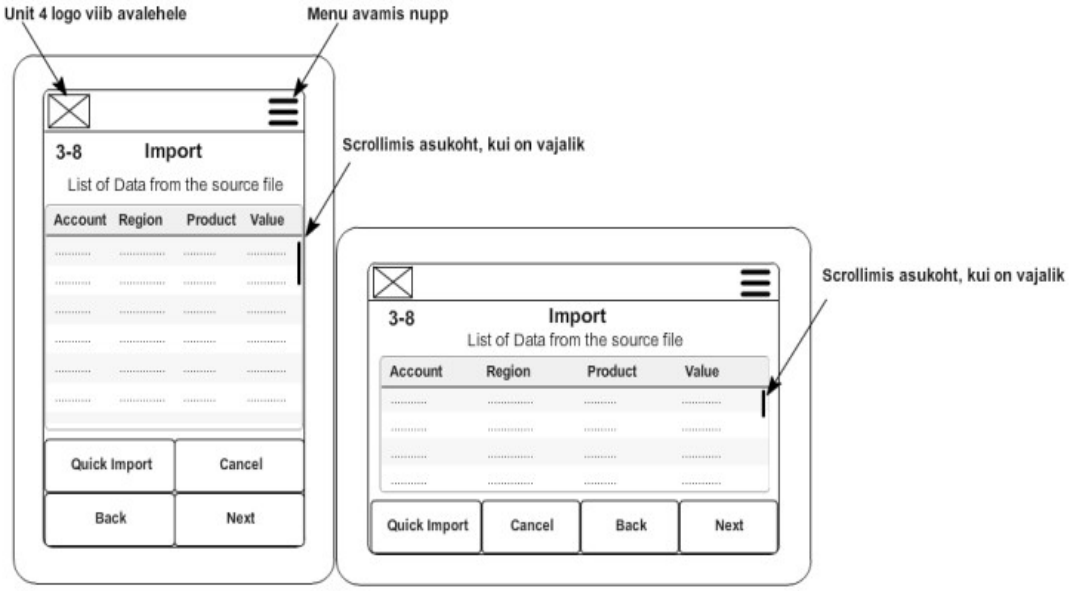


Joonis 39: Impordi samm 2 telefonis



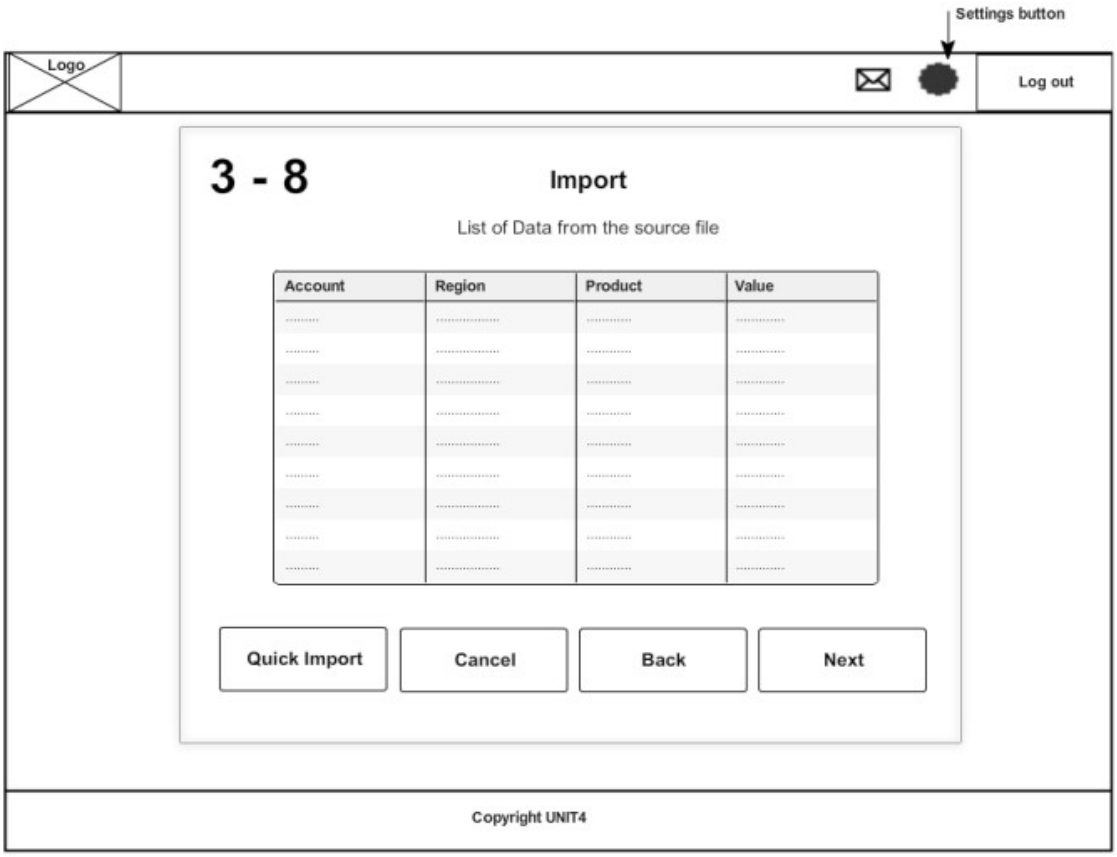
Joonis 40: Impordi samm 2 arvutis ja tahvlis

Impordi sammud 3 ja 4



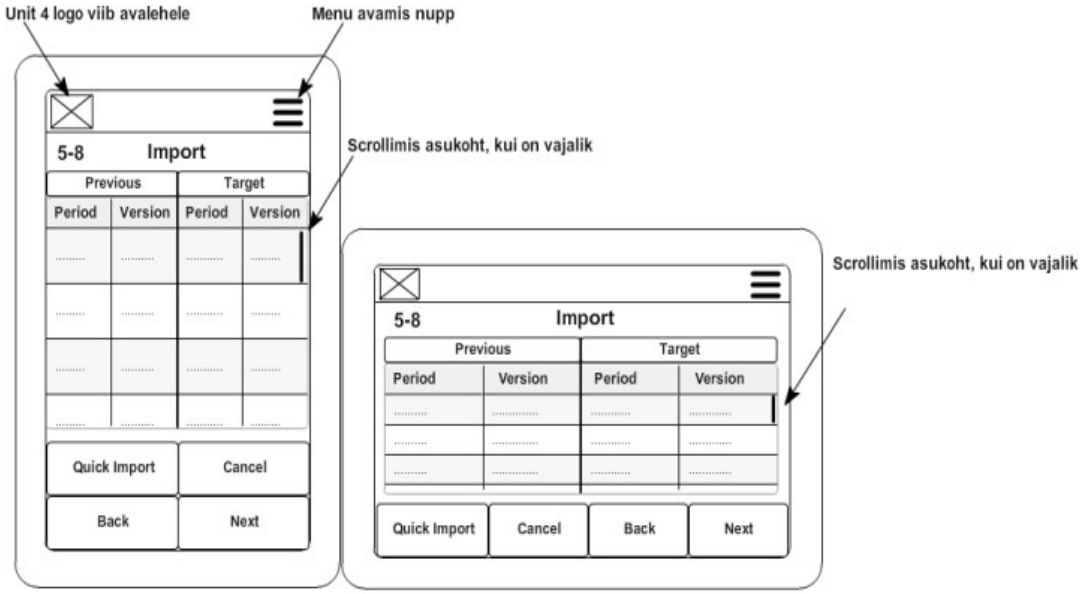
Joonis 41: Impordi sammu 3,4 telefoni vaade

Kuna sammud 3 ja 4 on sarnased siis need on kuvatud sama *wireframe* peal.



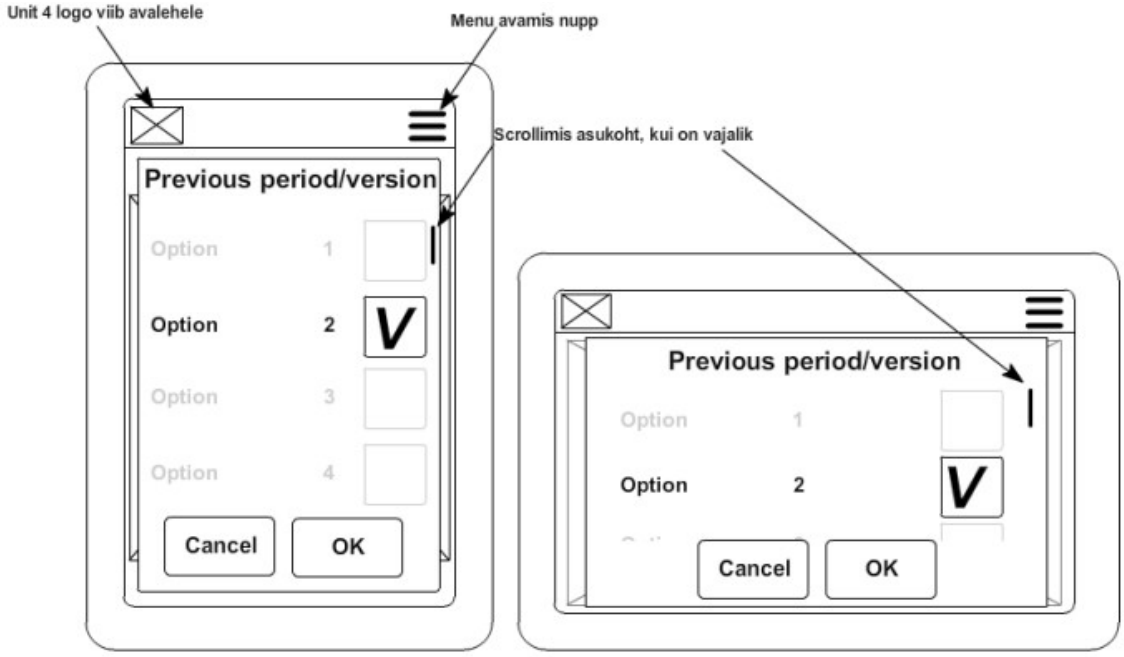
Joonis 42: Impordi sammu 3,4 arvuti ja tahvli vaade

Impordi samm 5



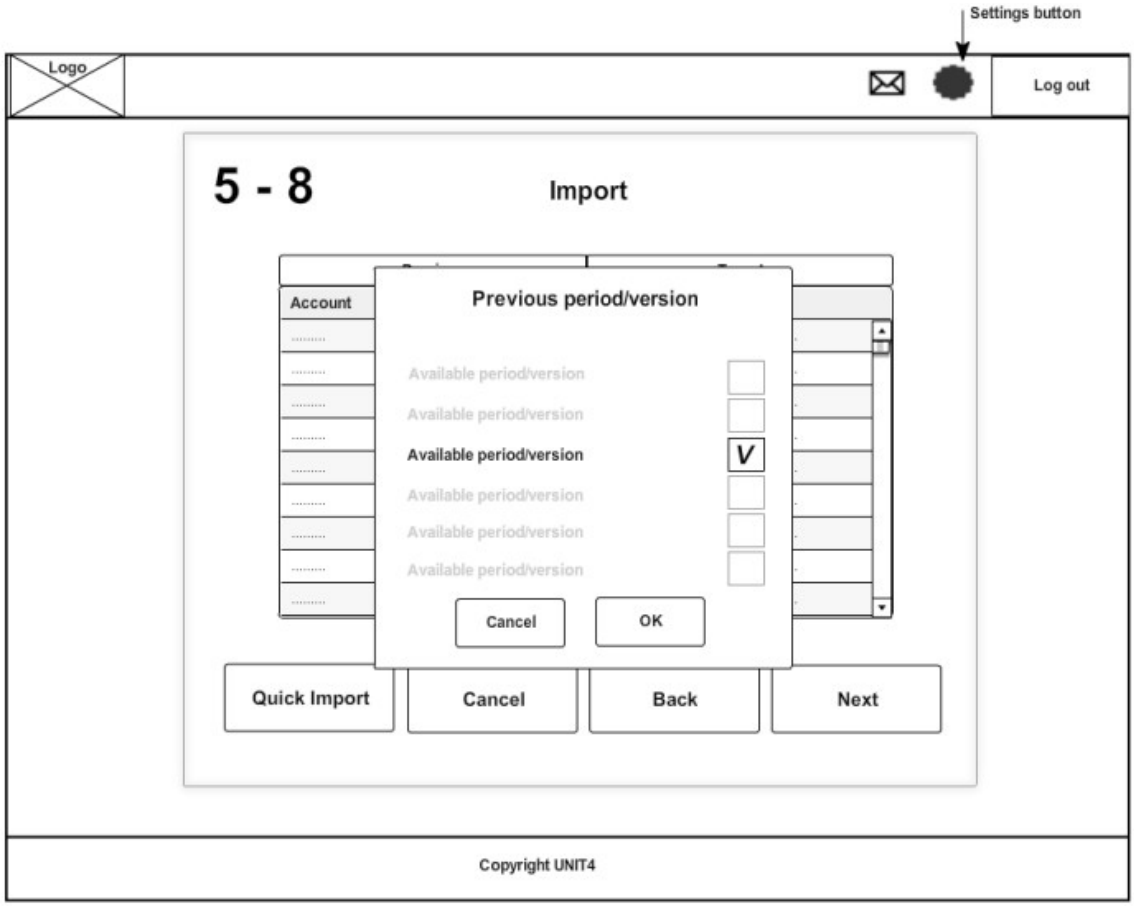
Joonis 43: Impordi samm 5 telefonis

Klõpsates *Previous* osa (joonis 43) tabeli ühel ruudul avaneb valik joonisel 44.



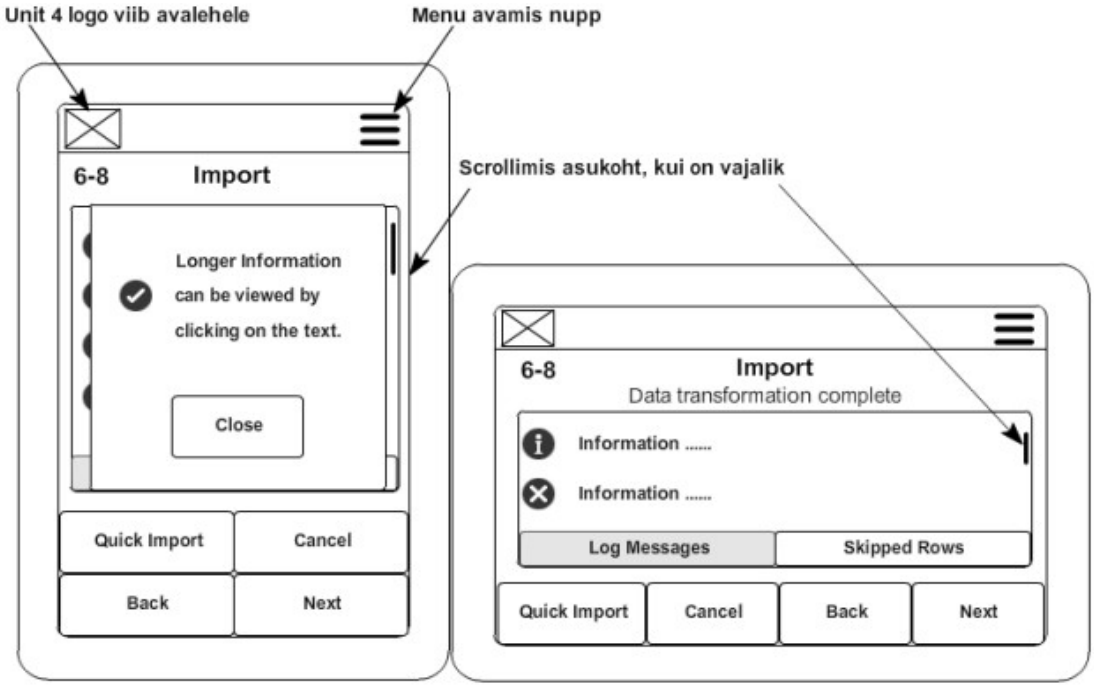
Joonis 44: *Previous* perioodi ja versiooni valik

Joonise 45 sammu 5 tabel on sama, mis joonisel 43.



Joonis 45: Impordi samm 5 arvuti ja tahvli vaade

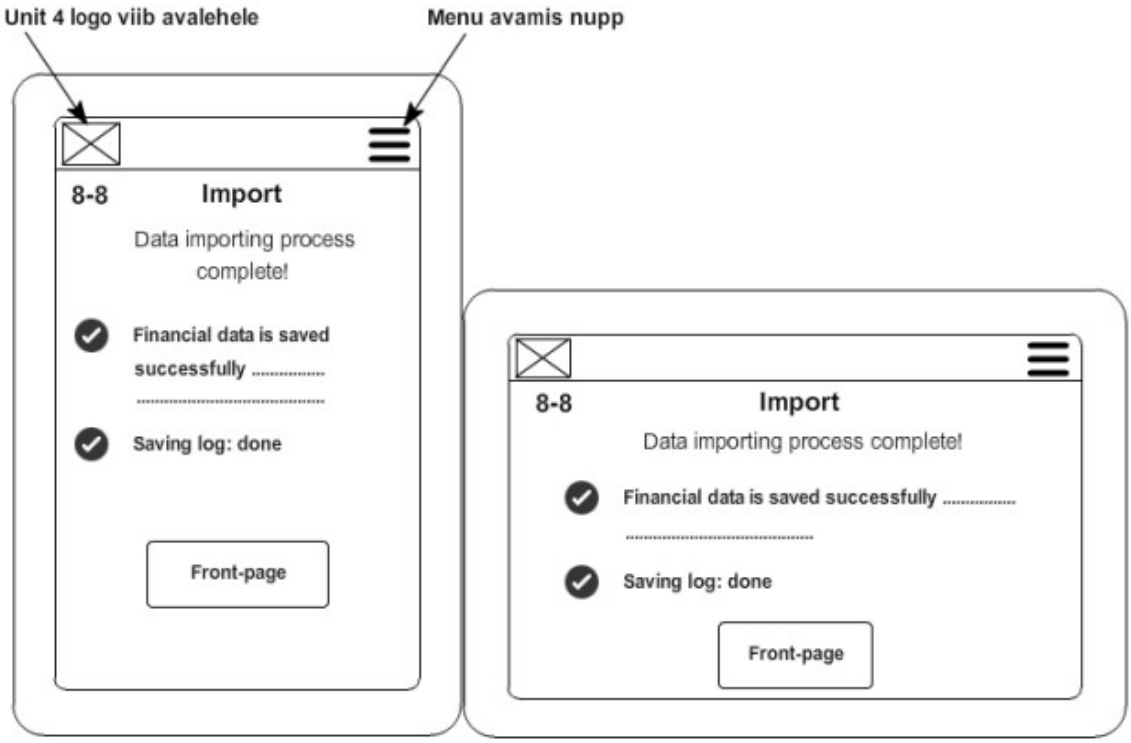
Impordi samm 6



Joonis 46: Impordi sammu 6 telefoni vaade

Joonis 47: Impordi samm 6 arvuti, tahvli vaade

Impordi samm 8

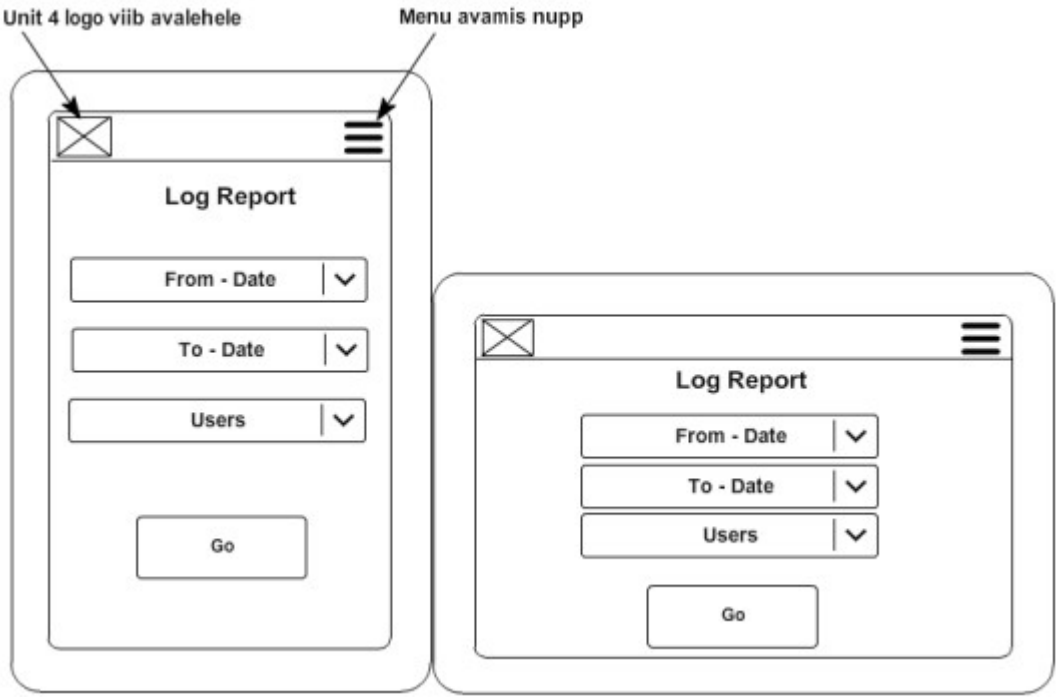


Joonis 48: Impordi samm 8 telefoni vaade

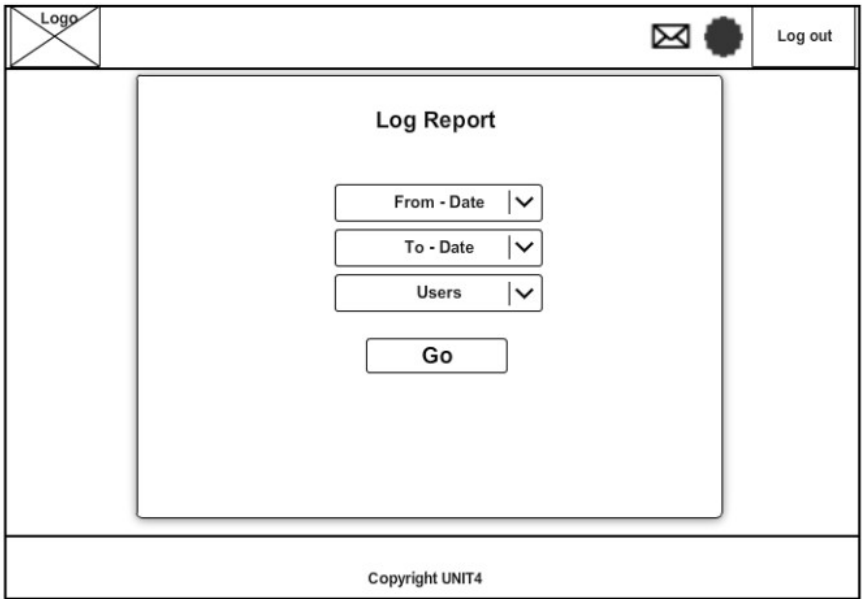
Joonis 49: Impordi samm 8 arvuti, tahvli vaade

Import Log History

Log Reporti valik

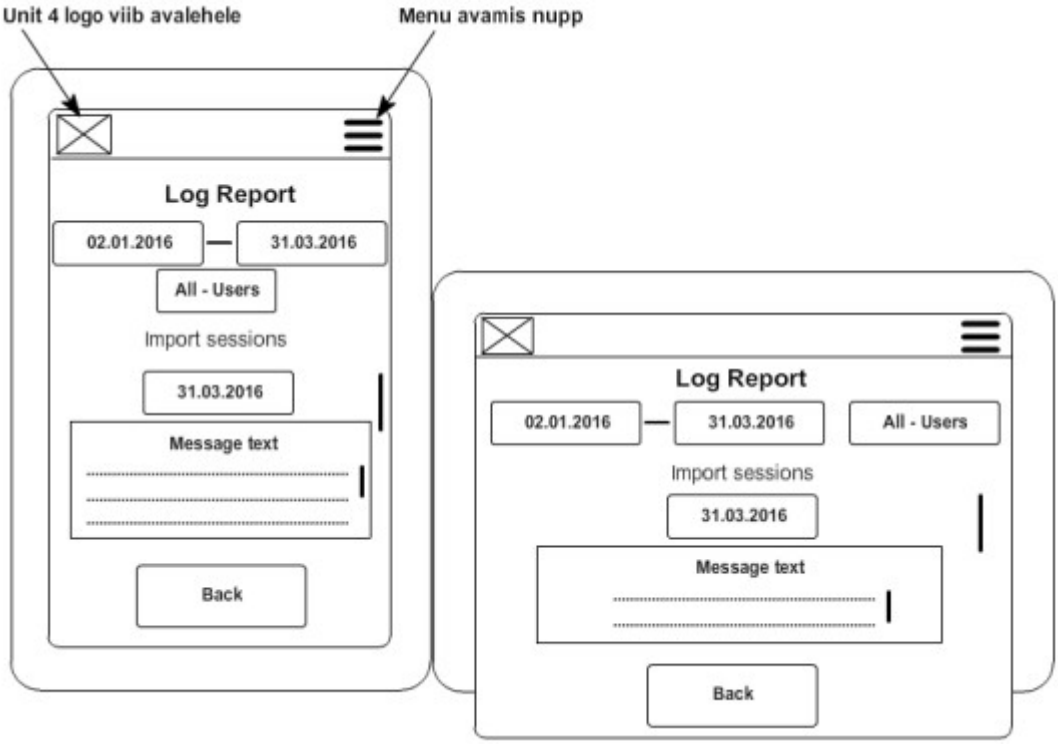


Joonis 50: Vali *Log Report* telefoni vaade

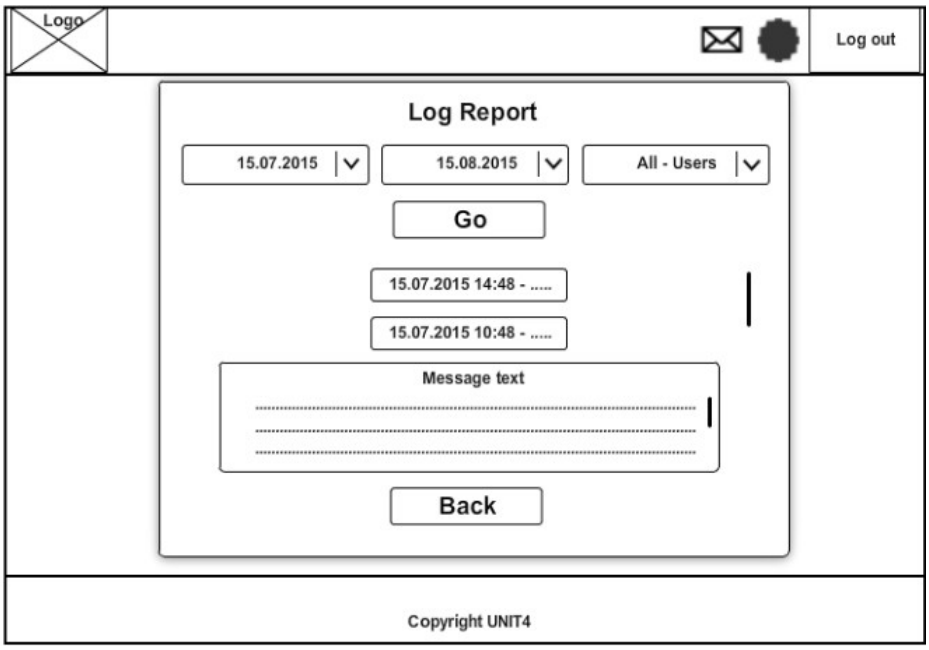


Joonis 51: Vali *Log Report* arvuti, tahvli vaade

Log Report-id



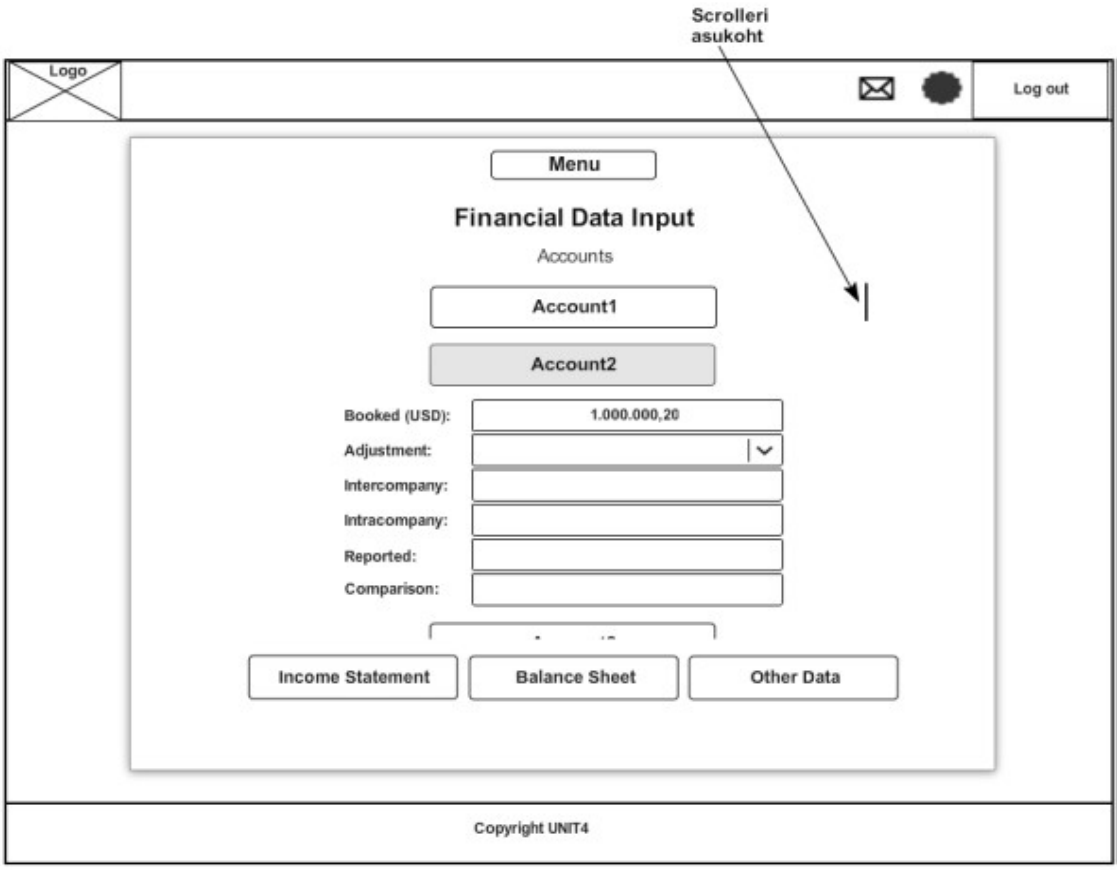
Joonis 52: *Log Report*-id telefoni vaade



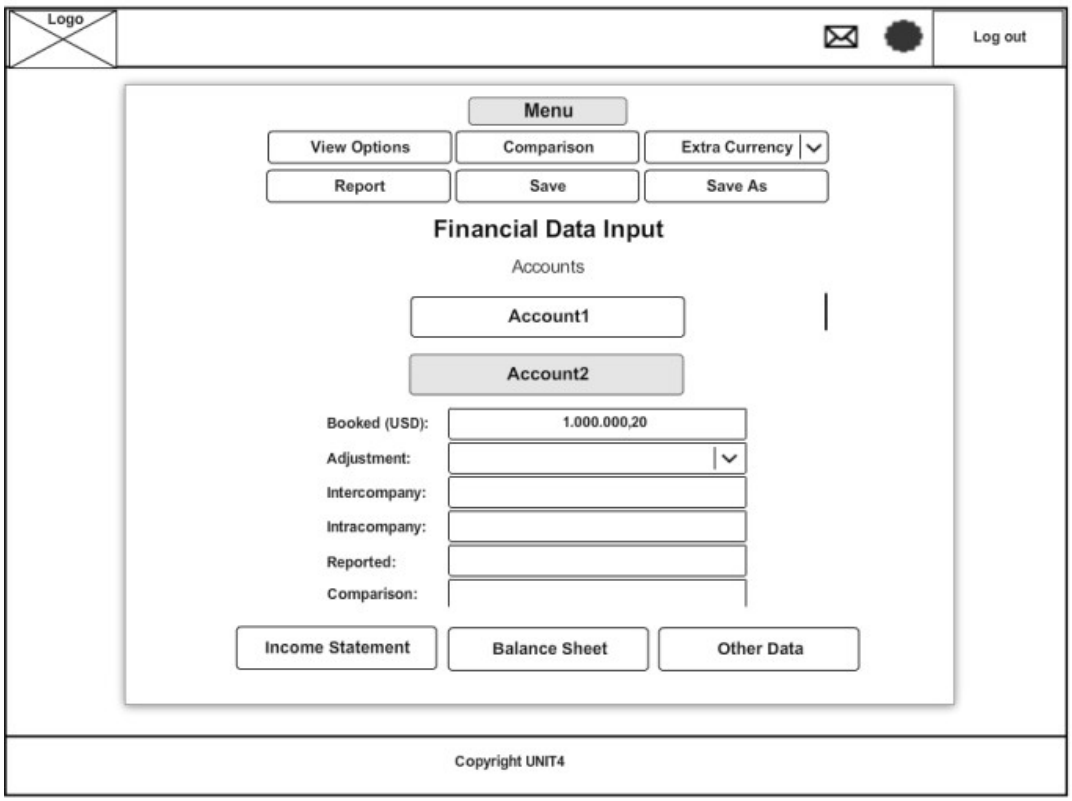
Joonis 53: *Log Report*-id arvuti vaade

FDI osa vaated

Arvuti vaade andmete töötlus tabelist

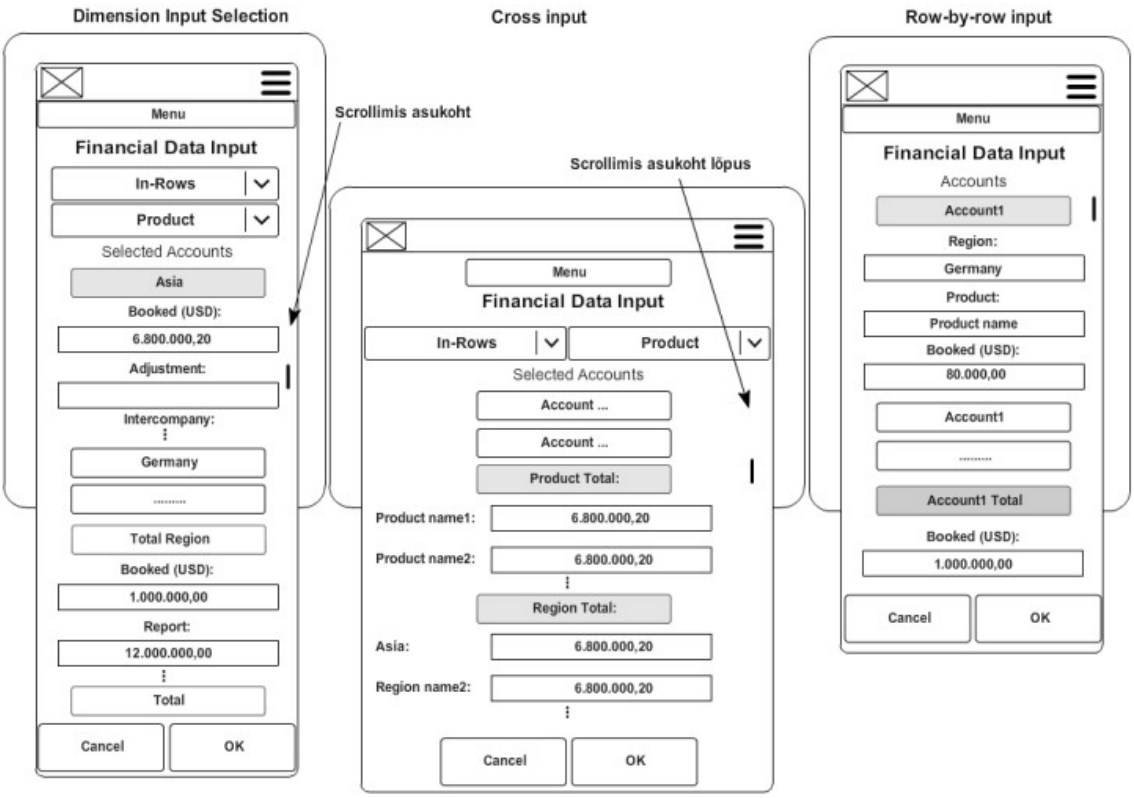


Joonis 54: *FDI* andmetöötluse tabeli vaade 1 arvutil, tahvlil

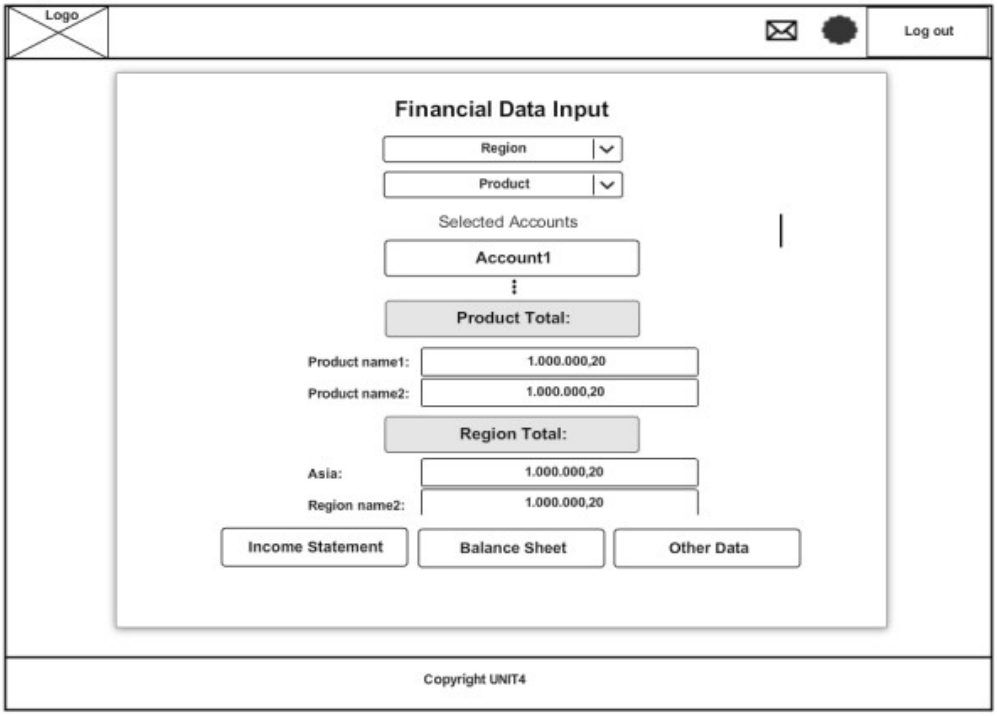


Joonis 55: *FDI* andmete töötlus vaade 2 arvutile, tahvlile

View options erinevad valikud

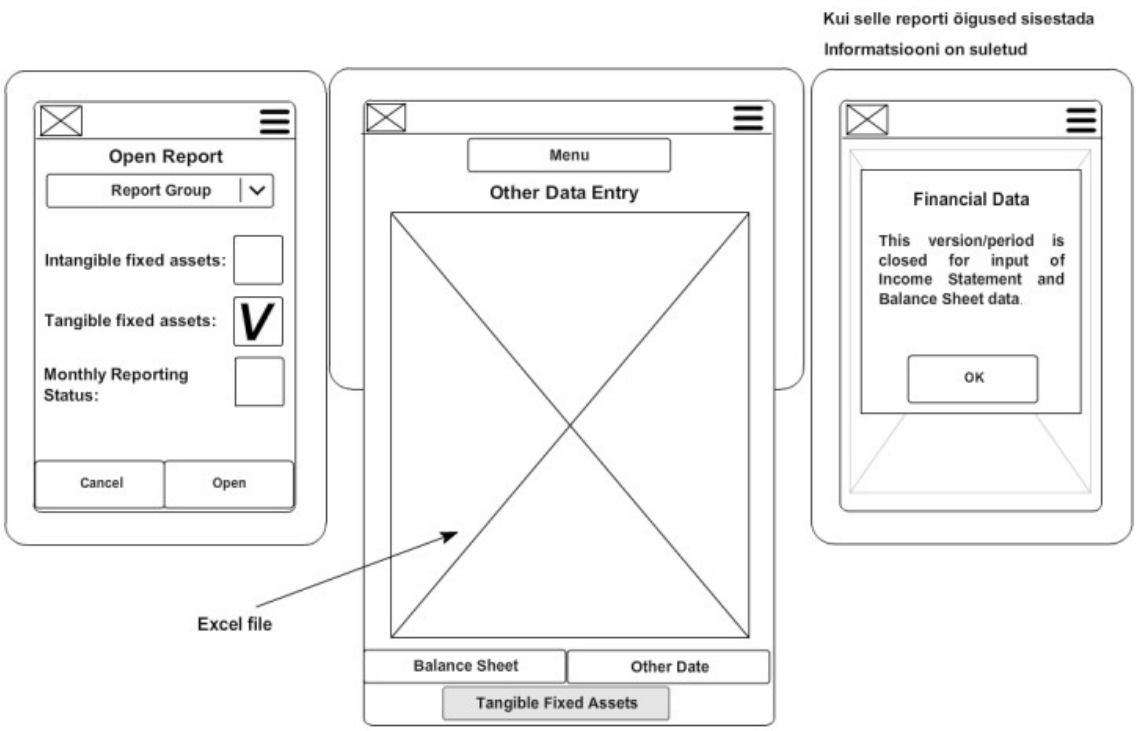


Joonis 56: *View options* erinevad valikud telefoni vaated

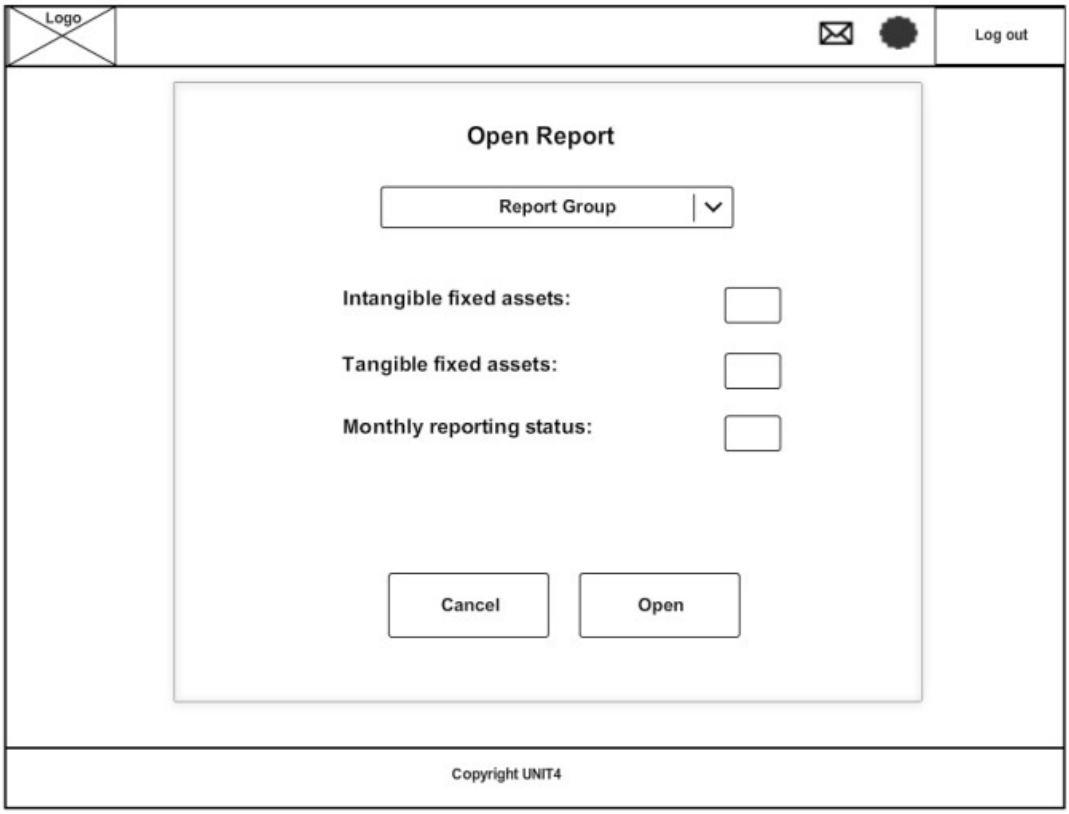


Joonis 57: *View option cross* input vaade

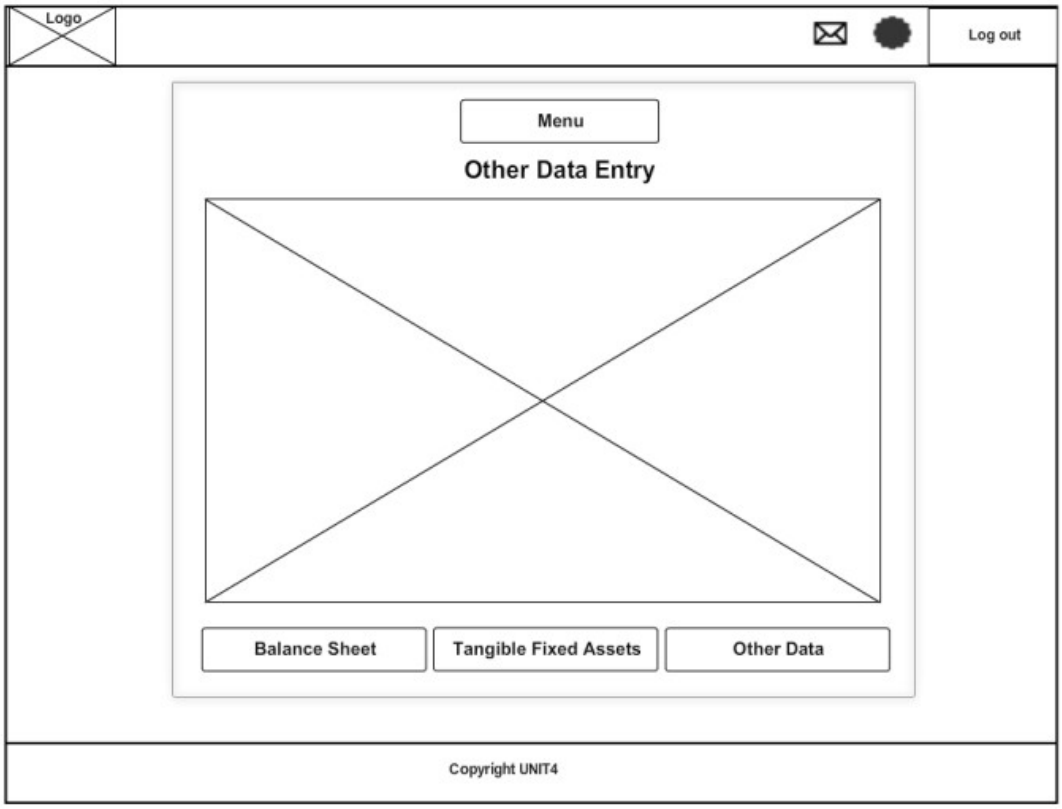
Valik *Open Report*



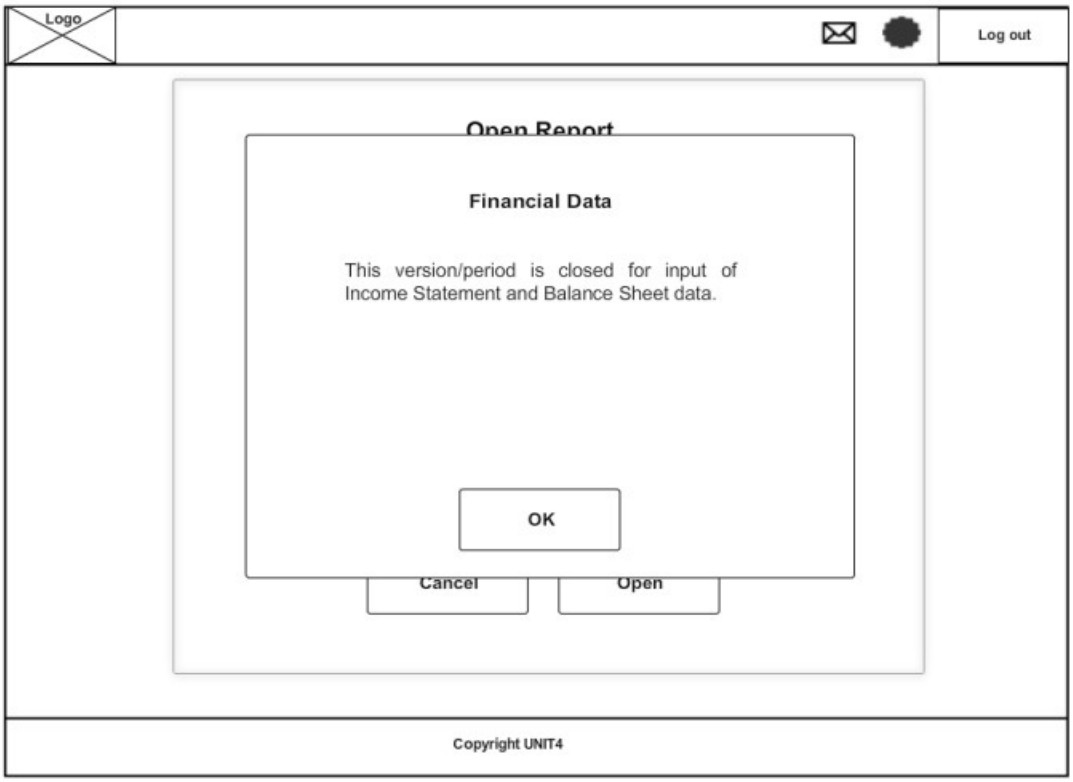
Joonis 58: *Oper Report* telefoni vaade



Joonis 59: *Open Report*-i avamine arvuti vaade

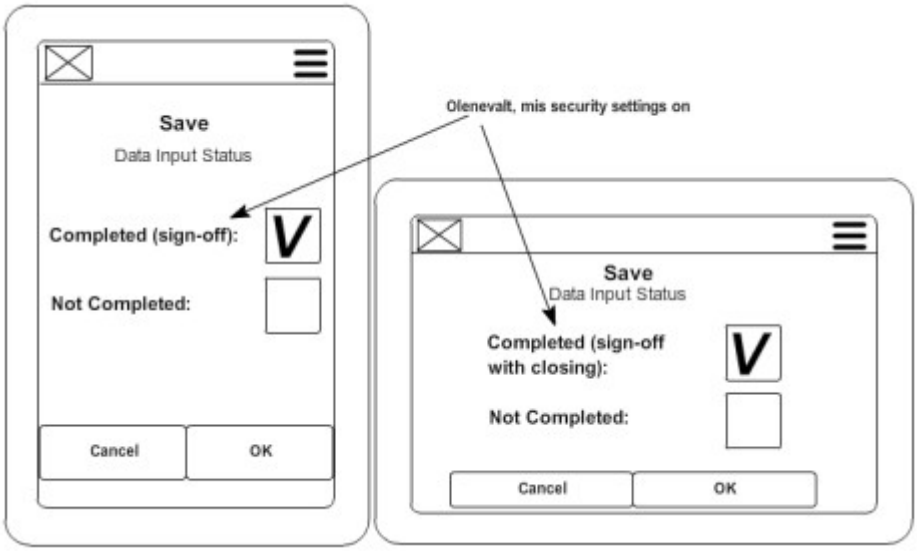


Joonis 60: *Other Data Entry* tabel raportis arvuti ja tahvli vaade

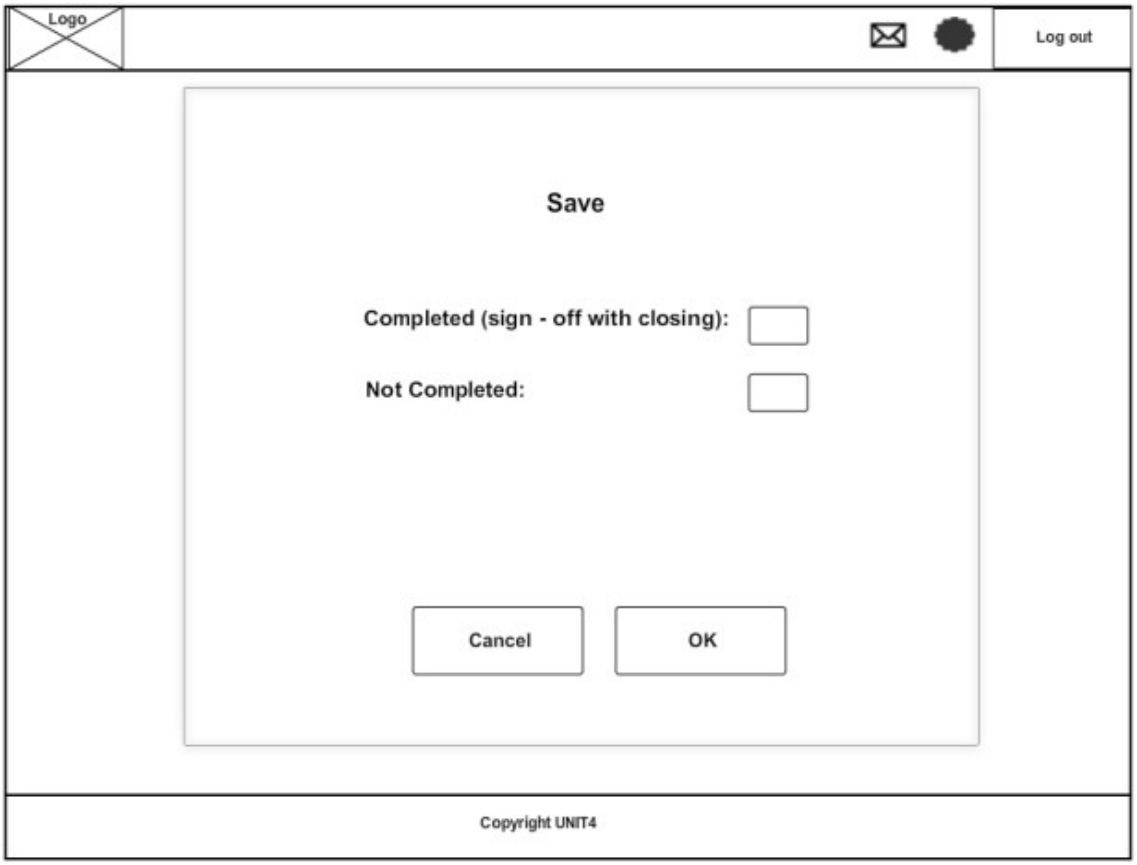


Joonis 61: Vea sõnum *Open Report*-is Arvuti ja tahvli vaade

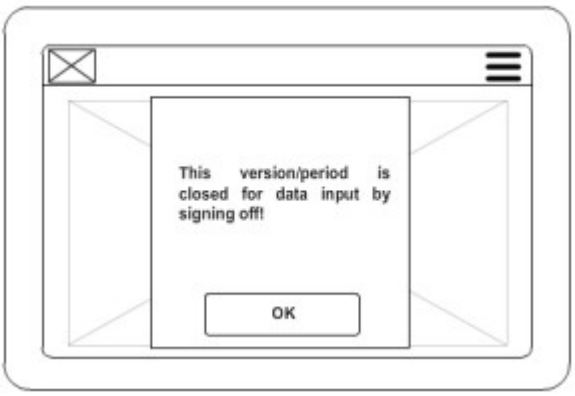
Save vaated



Joonis 62: *Save* telefoni vaade

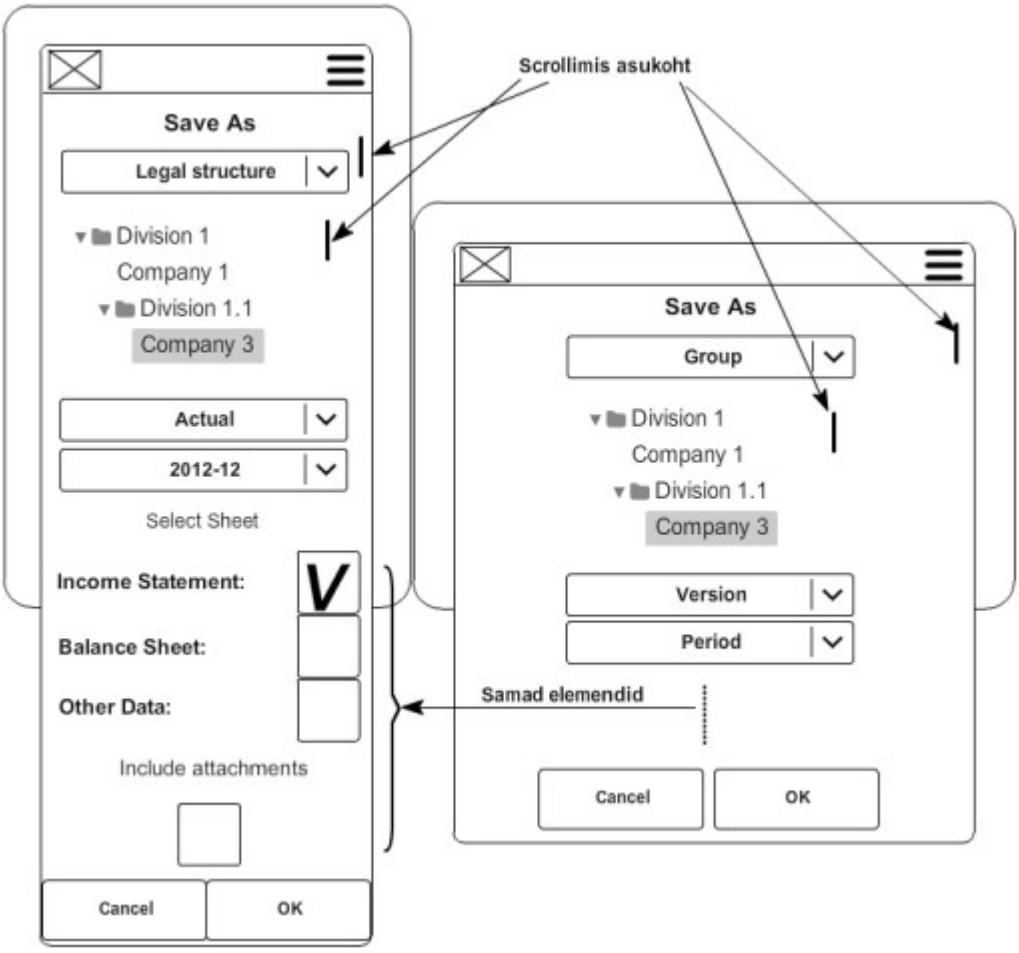


Joonis 63: *Save* arvuti vaade

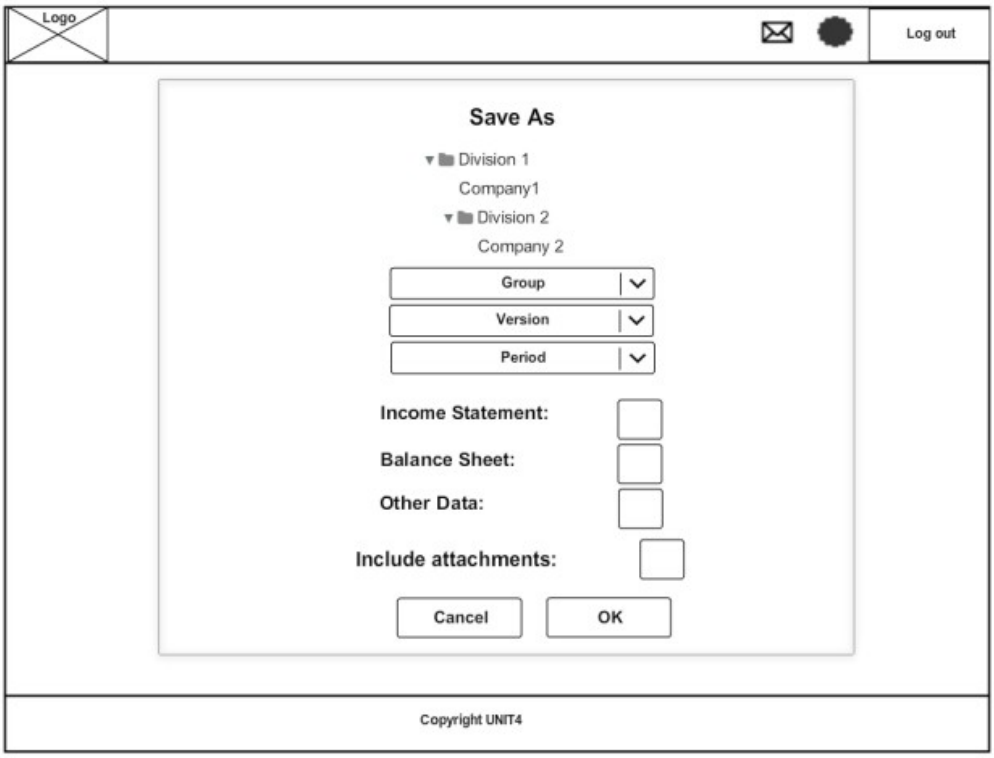


Joonis 64: Vea sõnum salvestamisel

Save As

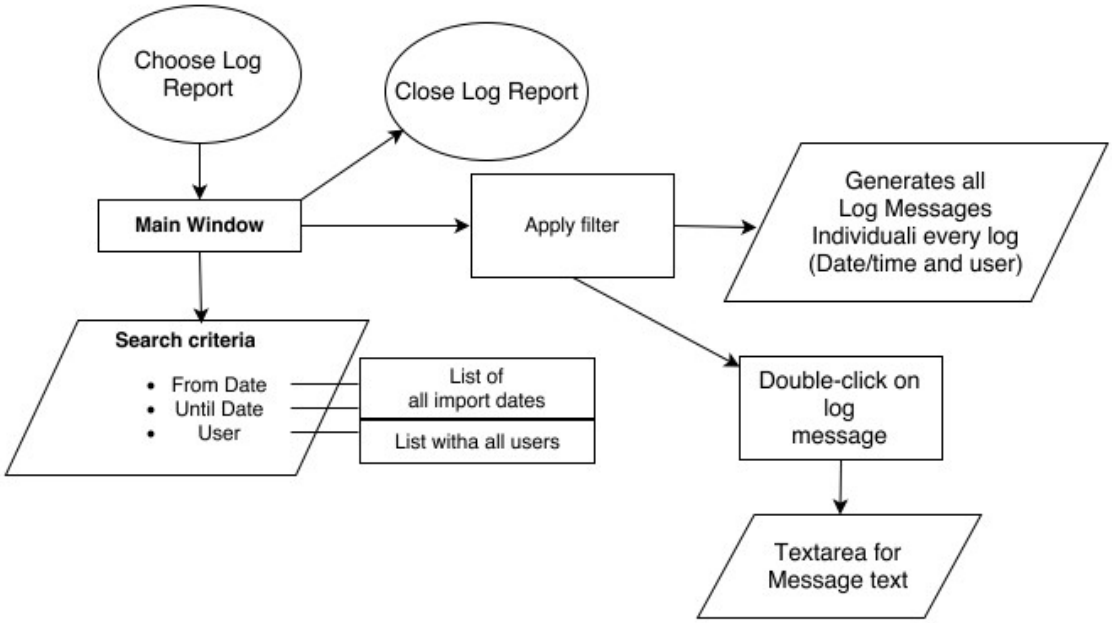


Joonis 65: *Save As* telefoni vaade

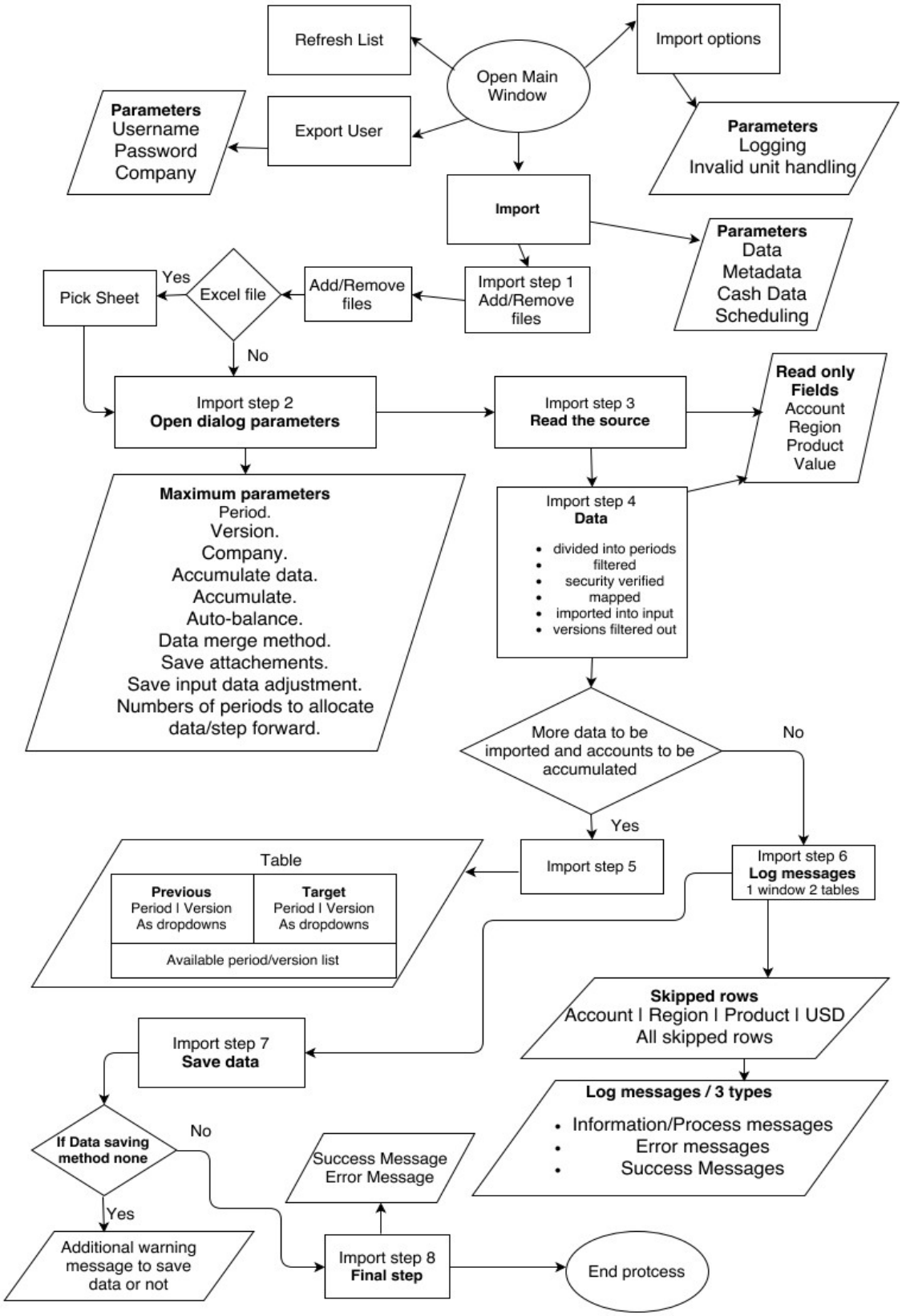


Joonis 66: *Save As* arvuti, tahvli vaade

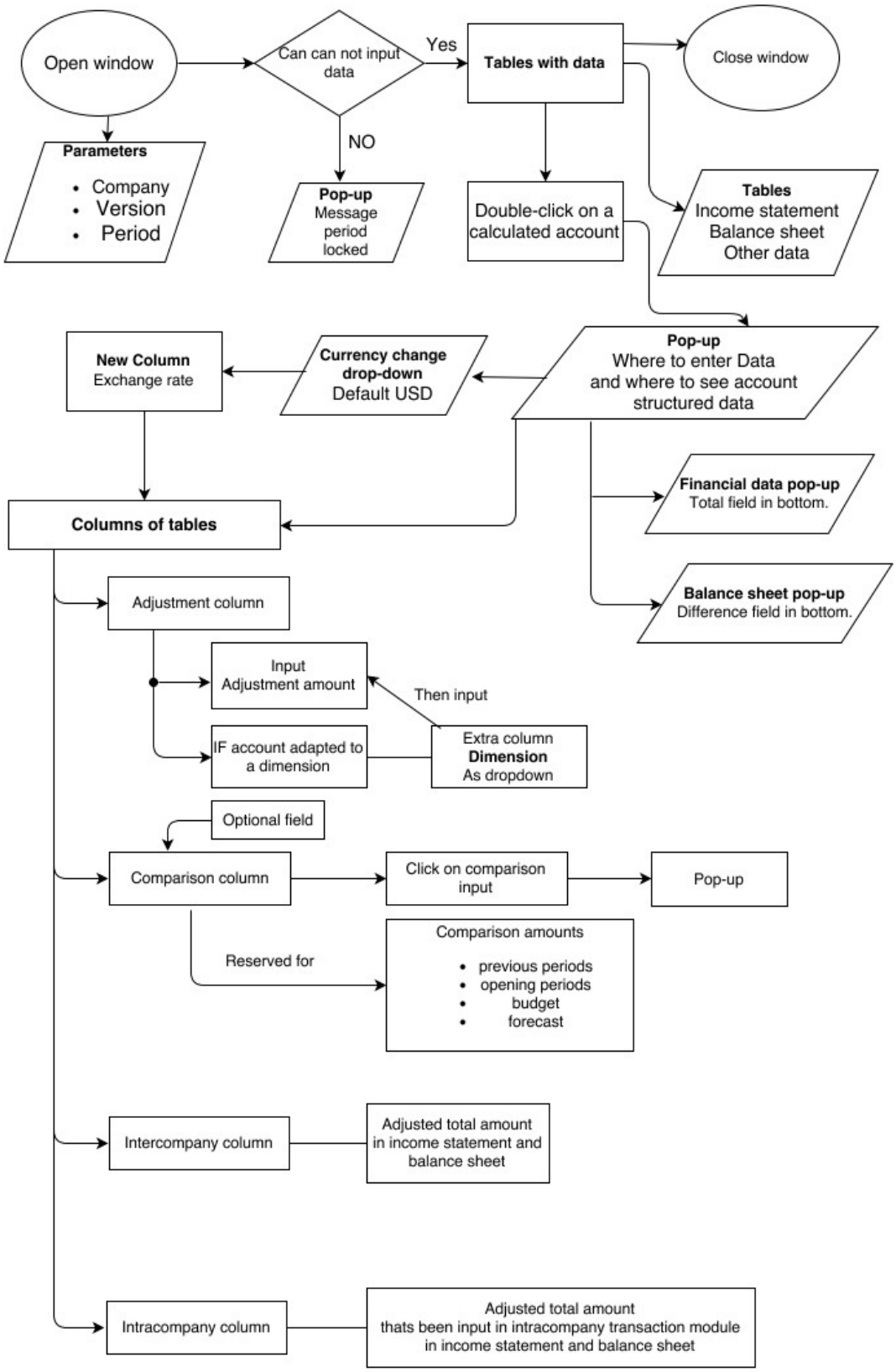
Lisa 2 – Tarkvara funktsionaalsese Vooskeemid



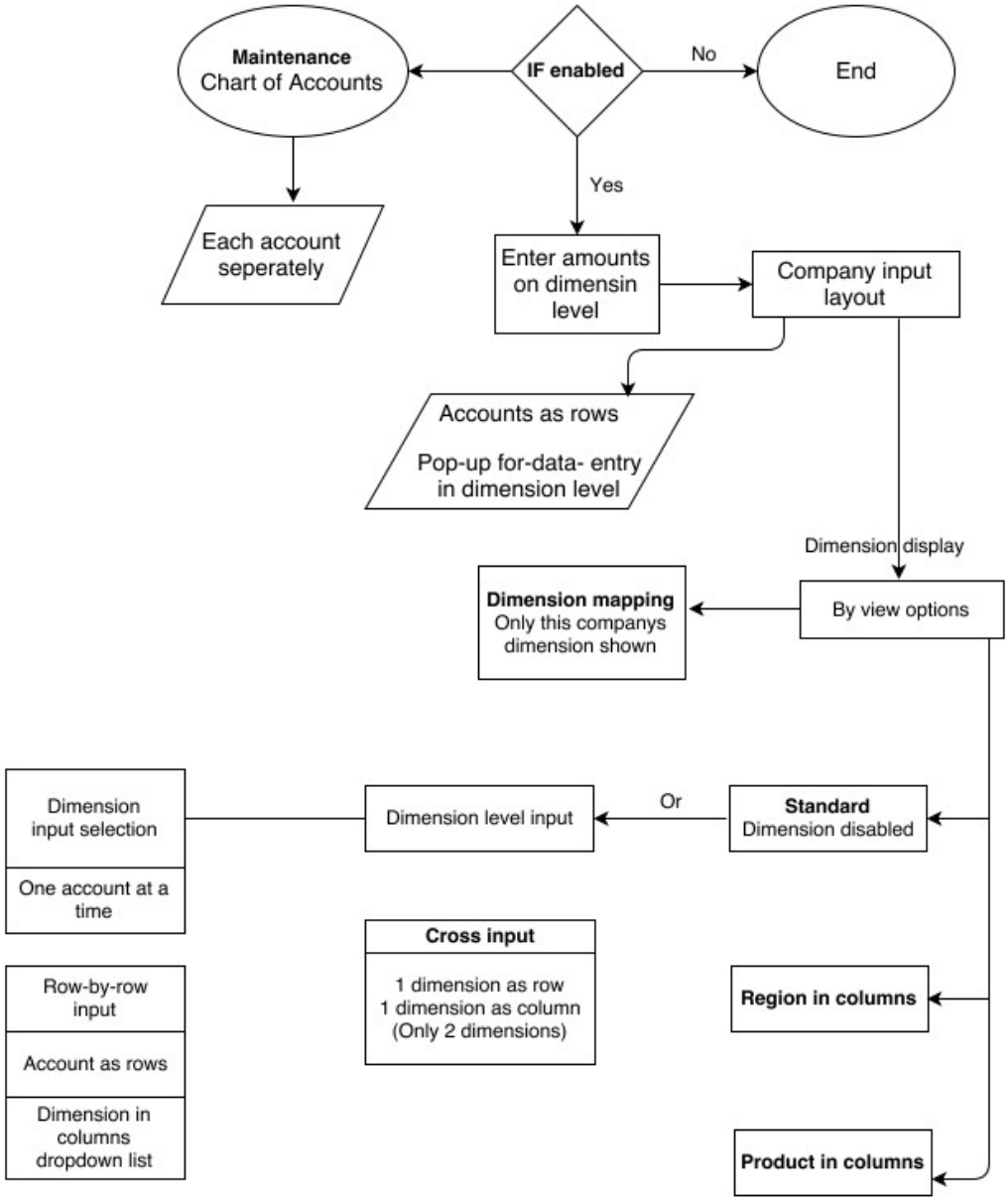
Joonis 67: *Log Report* funktsionaalsuse vooskeem



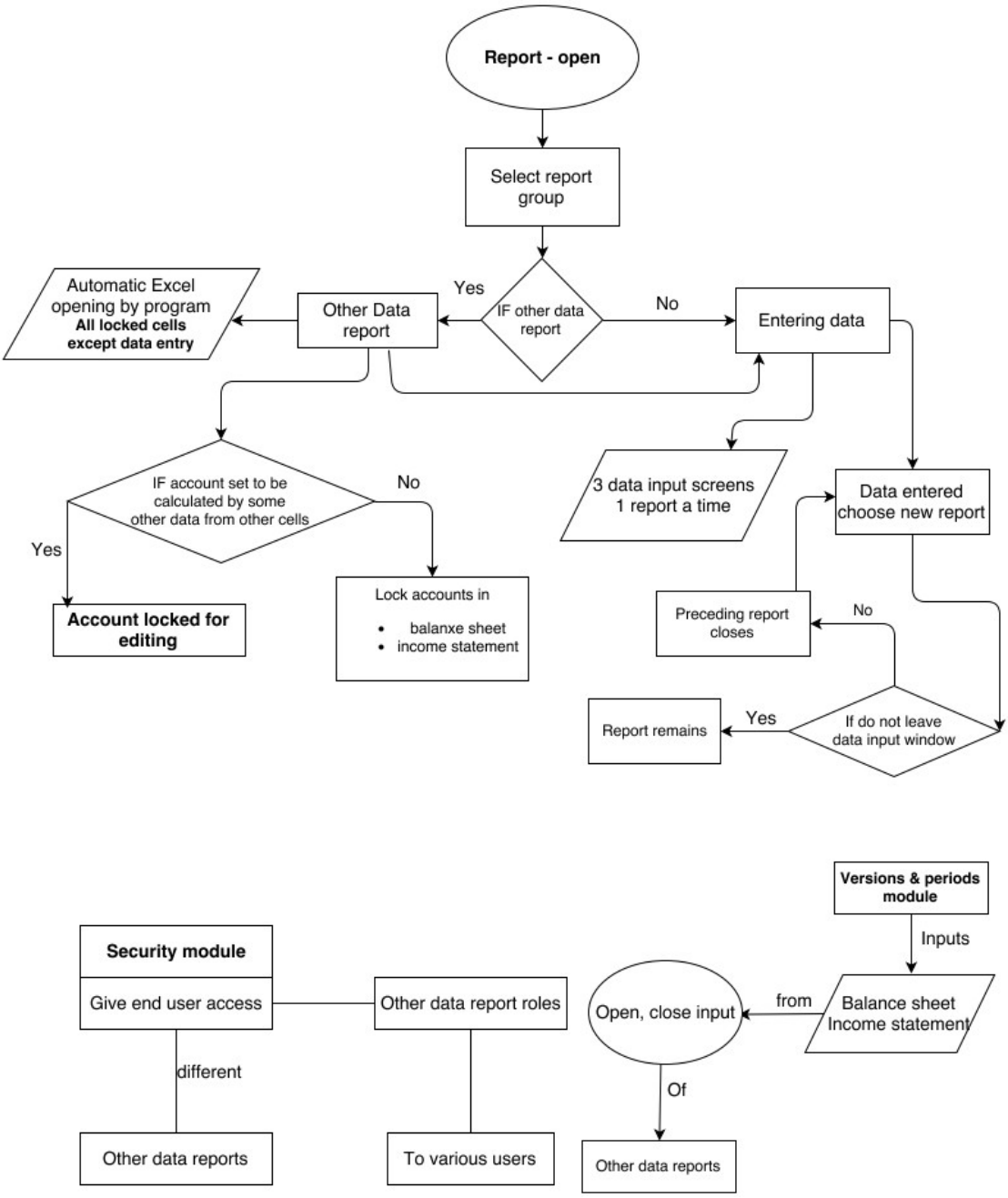
Joonis 68: Import funktsionaalsuse vooskeem



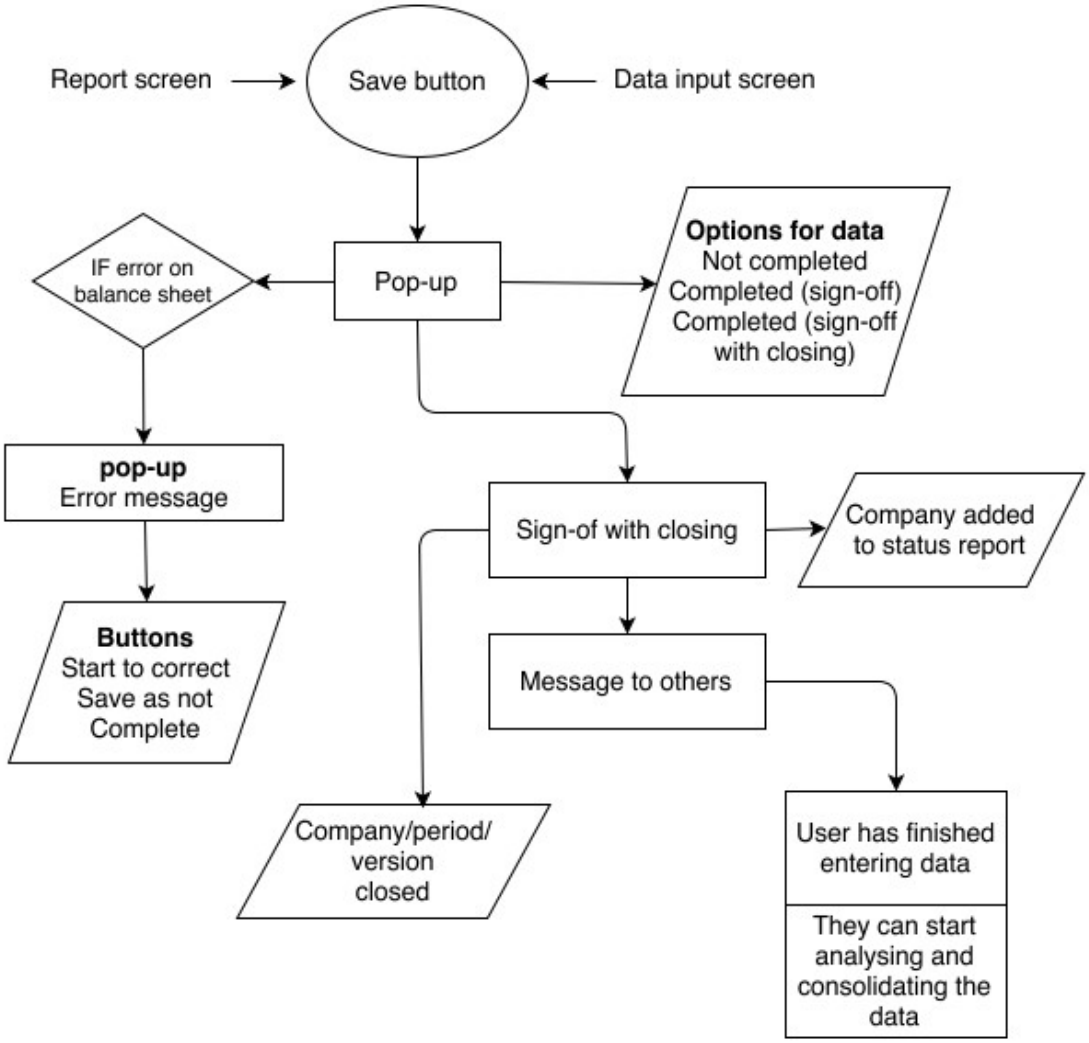
Joonis 69: *FDI* osa funktsionaalsuse vooskeem



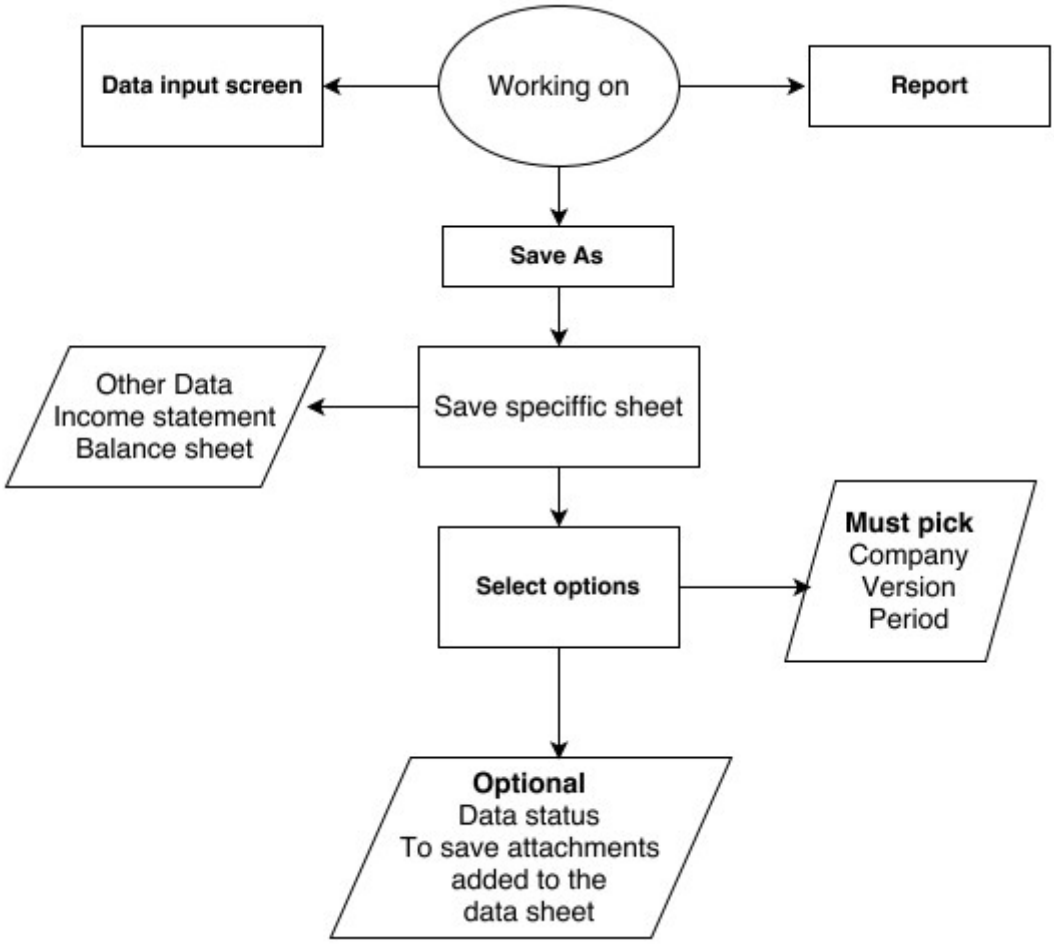
Joonis 70: *FDI* funktsionaalsuse osa *dimensions* vooskeem



Joonis 71: *Open Report* funktsionaalsuse vooskeem



Joonis 72: *Save* funktsionaalsuse vooskeem



Joonis 73: *Save As* funktsionaalsuse vooskeem