Technical University of Denmark

e-conomic Mobile. A re-design and development of a mobile application targeted Apple's iOS devices based on existing app.

Morten Hulvej Andersen (s083117)

B.Eng's Thesis, September 2013

IMM-B.Eng-2013-9

Supervisor:
Bjarne Poulsen, DTU Compute
Building 303B, room 052
2800 Kgs. Lyngby

DTU Compute
Department of Applied Mathematics and Computer Science

Technical University of Denmark
Applied Mathematics and Computer Science
Building 303B, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@compute.dtu.dk
www.compute.dtu.dk IMM-B.Eng-2013-9

This report was written in LaTeX using ShareLaTex - `https://www.sharelatex.com/`

Sourcecode available on `https://github.com/eco-inovation/eco-mobile`
Using credentials: U/N: `s083117`, PW: `s083117morten`

Forord

Denne Diplom bachelor rapport er udarbejdet i perioden 25. Marts 2013 til 9. September 2013 og resultatet af et projekt der markerer afslutningen af IT-Ingeniør uddannelsen på Danmarks Tekniske Universitet i Lyngby. Projektet er udført i samarbejde med e-conomic international a/s, København.

Jeg vil i denne forbindelse gerne takke flere personer der har bidraget til arbejdet med dette projekt. **Jes Brinch**, Director, Business Development ved e-conomic for tro støtte og opbakning, samt for idéer og sparring gennem hele projektet. **Mette Walsted**, Head of UX & Design ved e-conomic, for hjælpsomhed og udlevering af design materiale og dokumentation. **Morten Schmidt**, Senior Developer ved e-conomic for teknisk hjælp og test. En stor tak skal også lyde til min vejleder **Bjarne Poulsen** for råd og vejledning igennem dette arbejde.

København, 9. September 2013

Morten Hulvej Andersen
Tlf. 60 61 65 30

Abstract (English - Engelsk)

e-conomic international a/s is a leading provider of online bookkeeping and accounting software in Europe and also have apps that supplements the company's main product, *e-conomic* – an user-friendly, online accounting program.
e-conomic international have been having various challenges with those apps in production, due to the design nature of the apps and the techniques used in the development.
This paper documents the process in re-designing and re-writing the e-conomic app targeted for iOS (iPhone/iPad), using the native programming language and tools, and with respect to commonly known design patterns and industry standards.
This paper also documents the implementation of a REST API and a SOAP webservice in an iOS environment.

The paper is written with the assumption that the reader have a fairly high IT delevelopment knowledge or a IT background.

Abstract (Danish - Dansk)

e-conomic international a/s førende leverandør af online bogførings- regnskabsprogrammer i Europa og har samtidig mobile apps som tillæg til virksomhedens hovedprodukt, *e-conomic* – et brugervenligt, online regnskabs program.
e-conomic international har haft flere udfordringer med driften af disse apps, bl.a. på grund af deres opbygning og de valgte udviklingsteknikker.
Denne rapport dokumenterer processen at omdesigne og genskrive e-conomic app'en, målrettet iOS (iPhone/iPad), ved brug af de rigtige til formålet programmeringssprog og værktøjer, og med respekt for gængse design-mønstre og alment anerkendte industristandarder. Denne rapport dokumenterer samtidig også implementeringen af et REST API og en SOAP webservice i et iOS miljø.

Denne rapport er skrevet ud fra antagelsen af, at læserens har en IT-faglig baggrund eller på anden måde har et højt IT udviklings kendskab.

Contents

List of Figures

List of Tables

Listings

1 Introduktion

Dette kapitel vil give en introduktion til projektet, beskrive projektets interessenter, formål og problemstilling, herunder også motivationen for projektet.

1.1 e-conomic

e-conomic international (herefter e-conomic) er en af Europas førende leverandører af online regnskabsprogrammer og henvender sig primært til små og mellemstore virksomheder. e-conomic er stærkt repræsenteret i Norden, med den største kundebase i Danmark. Desuden har e-conomic kunder i det meste af Europa og sammen med en ny faktureringsservice *Debitoor*, løber det op i en kundebase på over 150.000 kunder.

e-conomic er også navnet på deres hovedprodukt, nemlig regnskabsprogrammet e-conomic. Det kører 100% online og er et SaaS (Software-As-A-Service) system, hvilket gør udvikling og vedligeholdelse nemmere end traditionelle programmer, som skal installeres lokalt på hver computer i f.eks. en virksomhed. *e-conomic* sikrer samtidig en mere proaktiv udviklingsproces, da opdateringer hurtigt kan rulles ud til samtlige brugere uden deres indgriben. Programmet er løbende blevet udviklet siden e-conomics blev skabt i 2001 af to revisorer og en software arkitekt. I dag beskæftiger e-conomic mere end 130 ansatte fordelt på over 12 lande.

1.2 e-conomic app-partnere & e-conomic API

Fra en tidlig alder har e-conomic været åben for integrationer i form af et API. Dette bruges i dag af rigtig mange "app-partnere", der leverer skræddersyede integrationsløsninger til kunder, der har et særligt behov. API'et er et åbent API, hvilket betyder, at alle kan udvikle op imod systemet, så længe man har en konto hos e-conomic. Hvis man derimod har en særlig god idé, som man tror kan bruges man mange brugere, kan man blive app-partner, som giver mere eksponering af ens løsning på specialindrettede app-sider på e-conomics websider.

e-conomics API bliver desuden også brugt internt til understøttelse af e-conomics egne apps til smartphones (herunder iOS og Andriod), som er udviklet af eksterne virksomheder i Makedonien.

Sideløbende med dette API, er et nyt type API under udvikling. Det nye API bygger på en mere tidsvarende teknologi og kan klare fremtidens udfordringer i forhold til belastning, tilgængelighed og skalérbarhed.

1.3 Eksisterende app

På nuværende tidspunkt er der en mobil app i drift, der giver e-conomics brugere adgang til deres e-conomic konto fra deres smartphone. Denne app er udviklet som en mobil *web-applikation*, og køres derfor ikke direkte på enheden, men på en webserver. App'en har været i drift siden sommeren 2012 og har fået skarp kritik af mange af e-conomics brugere for generelt at være for langsom. Desuden bliver også UI, brugbarhed og udnyttelse af en smartphones muligheder kritiseret.

1.4 Motivation

Mobil apps bliver i større og større grad brugt i industrien, som en udvidelse eller supplement til i forvejen etablerede produkter. Dette forøger efterspørgslen efter dedikerede app-udviklere og -designere. App udvikling er i løbet af de sidste par år blevet moden nok til at give merværdi for virksomheder, fremfor blot at være et lille hoppyprojekt eller "støj" på et ellers spil- og underholdingsdomineret app marked.

Jvf. udfordringerne beskrevet i afsnit 1.3, er der derfor brug for en tidssvarende, veldesignet og -konstrueret mobil app, der kan imødekomme de udfordringer den nuværende app besider.

Den primære motivation for dette projekt er at tilegne større viden og indsigt inden for iOS app udvikling, samt at tilegne større erfaring inden for udvikling af systemer med høje krav til UI, brugervenlighed og ydelse.

Motivationen for e-conomic i dette projekt, er at få udviklet en 100% *native* iOS app, som eliminerer de fejl og mangler som er tilstede i den eksisterende app. Dette vil give virksomheden større kundetilfredshed og desuden styrke dens position i forhold til dens konkurrenter .

1.5 Vision

Visionen for dette projekt er at udvikle et Proof-of-Concept (PoC) til en helt ny e-conomic app specielt til iOS, der tager mange af kritikpunkterne fra den oprindelige app op til diskussion. Samtidig er målet at udvikle og forbedre alle funktioner fra den oprindelige app og udbygge funktioner så den mobile platform kan udnyttes bedst muligt. Dette PoC skal agere som prototype for en app, der ville kunne videreudvikles og sættes i drift som et bedre alternativ til den eksisterende app.

1.6 Problemformulering

e-conomic står på nuværende tidspunkt med en web-app, der ikke er tilfredsstillende. De søger et alternativ til den eksisterende app, som dog stadig skal tilbyde de samme funktioner som den eksisterende app, blot tilpasset specifikt til Apples iPhones og iPads. Den nye app må meget gerne også tilbyde nye funktioner. Den skal være tilpasset, tidssvarende og udvikles i et fuldt iOS miljø.

Dette rejser flere spørgsmål til projektet, hvorfor der er udarbejdet følgende problemformulering:

1 Hvad er de største udfordringer med den eksisterende app?

2 Hvordan løses disse udfordringer i dette projekt?

3 Hvilke (i e-conomic) kendte funktioner er de vigtigste at implementere på en mobil platform?

4 Hvilke typer af e-conomic kunder vil have gavn af en mobil version af e-conomic?

5 Hvilke teknologier kan være relevante at bruge i udviklingen?

6 Hvilket udviklingsmønster skal der bruges i projektet?

7 Hvilke API'er skal appen gøre brug af?

8 Hvilke teknologier kan der bruges til at implementere disse API'er?

9 Hvordan skal disse API'er implementeres?

10 Hvordan sikres der at API'erne ikke dominerer implementationen?

11 Hvordan skal API-data håndteres?

12 Skal data gemmes lokalt på enheden?

13 Hvilken type af brugergrænseflade skal der benyttes?

14 Hvilken type navigation er bedst at benytte i brugerfladen til store datamængder?

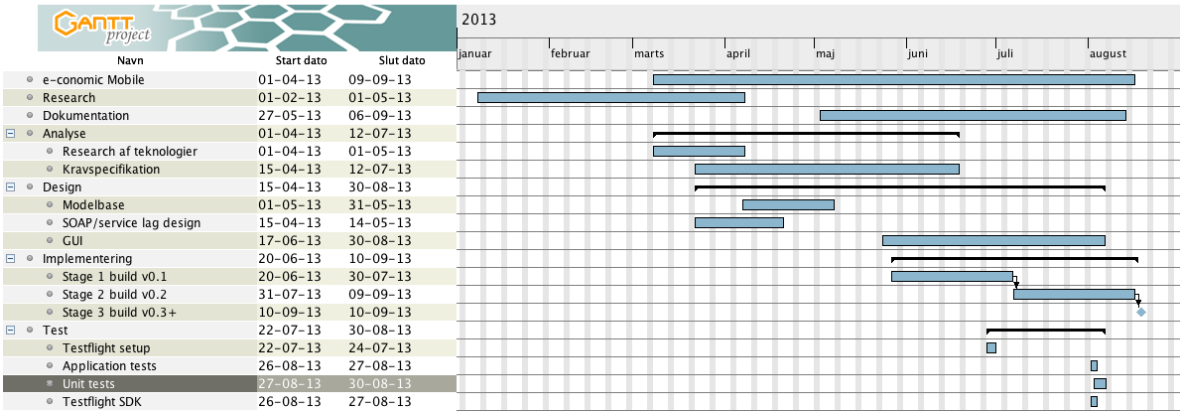
15 Hvad skal der lægges særlig vægt på i brugerfladen?

16 Hvordan opnås adgang til brugerdata hos e-conomic?

17 Hvordan kan/skal app'en testes?

1.7 Projektplanlægning

I projektforløbet arbejdes der bl.a ud fra en projektplan, der er med til at holde fokus på de centrale dele i projektet. Til dette formål er der udarbejdet et GANTT projekt, se figur 1. Fordi appen har et stort driftspotentiale er det vigtigt hurtigt at komme ud med basale, men dog essentielle funktioner, således at app'en hurtigt kan videreudvikles til en evt. idriftsættelse.



Navn	Start dato	Slut dato
e-conomic Mobile	01-04-13	09-09-13
Research	01-02-13	01-05-13
Dokumentation	27-05-13	06-09-13
Analyse	01-04-13	12-07-13
Research af teknologier	01-04-13	01-05-13
Kravspecifikation	15-04-13	12-07-13
Design	15-04-13	30-08-13
Modelbase	01-05-13	31-05-13
SOAP/service lag design	15-04-13	14-05-13
GUI	17-06-13	30-08-13
Implementering	20-06-13	10-09-13
Stage 1 build v0.1	20-06-13	30-07-13
Stage 2 build v0.2	31-07-13	09-09-13
Stage 3 build v0.3+	10-09-13	10-09-13
Test	22-07-13	30-08-13
Testflight setup	22-07-13	24-07-13
Application tests	26-08-13	27-08-13
Unit tests	27-08-13	30-08-13
Testflight SDK	26-08-13	27-08-13

Figure 1: Projektplan

1.8 Udviklingsmetode

Under projektet benyttes der udviklingsmetoder indenfor *Unified Process*[1] (UP), da denne passer godt i spænd med udviklingsplanen og den objekt-orienterede tilgang til analyse og design (OOA/D) i forhold til problemformulering.

Under udviklingen vil der altid være en fungerende app tilstede, blot med meget skrabede, eller slet ingen funktioner fra starten. Det ville ikke give megen mening at fokusere på yderligere konkrete udviklingsmetoder, som f.eks. *Scrum* da dette fokuserer meget på teamarbejde. Med projektets størrelse taget i betragtning, vil et generelt forhold til UP være tilstrækkeligt. Dog vil udviklingen bære meget præg af generelle agile softwareudviklingspricipper[2], da projektet opfylder nogle nøglepunkter heraf:

- Der vil altid være en kørbart udgave af produktet.
- Kravene kan (og vil) ændre sig i løbet af udviklingen, samarbejde med e-conomic.
- Der er konstant dialog med e-conomic og mulige brugere af produktet.
- Komponenter i produktet vil konstant blive testet, for at imødekomme "godt design", herunder bl.a. GRASP principperne[3].

Krav til projektet kan ændre sig løbende, og der er derfor i projektet konstant dialog med e-conomic omkring, hvad deres kunder har brug for. Dette vil sige, deres *mobile* kunder frem for deres primære kunder – som sagtens kan være de samme. Det kan være, at der bliver frigjort en funktion A, der ikke giver mening uden funktion B.

Der vil løbende blive udgivet prototyper eller test-versioner af app'en, i en tilstand hvor den kan køres nogenlunde uden uoprettelige fejl, således at flere internt hos e-conomic kan følge med i udviklingen, og give tidlig feedback på projektet.

TestFlight[4] vil blive brugt til at distribuere app'en mellem medarbejdere fra e-conomic og andre interessenter under udviklingen. Desuden vil TestFlight SDK blive inkorporeret i app'en således, at der automatisk kan indsamles nyttige informationer, når app'en bruges. Se mere i afsnit 5.3.

GRASP[5] er et sæt design mønstre som kort sagt sikrer lav kobling og høj binding i et softwaredesign. Hvis man inkludere GRASP i en tidlig fase af et udviklingsforløb, vil man ende op med et softwaredesign der indvendigt er robust mod ændringer, skalerbart, delbart, og nem at fejlrette. GRASP handler også meget om ansvarsfordeling af opgaver. De rette opgaver til de rette objekter. Model-View-Controller mønsteret er et mønster der overholder GRASP principperne, hvis implementeret korrekt. I analyse- og designfase af dette projekt, vil GRASP blive benyttet som en rettesnor til at sikre et godt design.

[1] Larman, s. 18
[2] Larman, s. 29
[3] Larman, s. 291
[4] https://www.testflightapp.com
[5] Larman s. 271, (General Responsebility Assignment Software Patterns or Principles)

1.8.1 Versionsstyring

Til versionsstyring og –sporing benyttes *Git*[6] til først og fremmest at versionere koden-basen og dokumentationen. Git er et *distribueret versionsstyrings værktøj*, og er særdeles god til team-udvikling, herunder især i teams der udvikler efter agile metoder med mange korte iterationer. Under dette projekt bliver det dog hovedsageligt brugt som historik og versions-backup. Git kan efter noget tid også nemt komme til at fungere som dokumentation, eftersom loggen kan fremvise forskellige designbeslutninger og –ændringer over tid. Git kan dermed bidrage til opklaring af spørgsmål senere i udviklingen, eller hvis der kommer andre udviklere ind over projektet.

Alt kildemateriale og alle ressourcer bliver så vidt muligt lagt under versionsstyring, således at man til enhver tid har adgang til et fuldt kørbart udviklingsmiljø, selv ved en reetablering af hele projektet på f.eks. en anden maskine. Hertil skal også nævnes, at der bestræbes på *kun* at tilføje kørbare tilstande af projektet, dvs. hvor der kan foretages en succesfuld kompilering og app'en kan køres til en hvis grad fejlfrit.

1.9 Risikoanalyse

Med e-conomics to forskellige API'er, er der en udfordring i at skulle implementere dem begge på en iPhone. For det første kan der opstå en situation, hvor SOAP API'et viser sig at være for forskellig i natur til at implementeringen i et iOS miljø vil kunne lade sig gøre. For det andet er REST API'et stadig under udvikling og det er endnu ukendt, hvorvidt det når at komme i en tilstand, der kan bruges til dette projekt.

[6] http://git-scm.com/

1.10 Rapportens opbygning

Denne rapport vil afspejle en dokumentation af et softwareprojekt og derfor vil rapportens opbygning være præget af UP udviklingsmetoden, hvorfor rapportens kapitler også vil afspejle de forskellige faser i UP[7], dog med andre navne:

- Inception vil være analyse og teknologi
- Elaboration vil være design
- Construction vil være implementering
- Transition vil være test

De forskellige kapiteler vil ikke nødvendigvis dække præcist eller til fulde den beskrivelse Larman angiver i referencen. Der vil også være ord, eller mindre afsnit skrevet på engelsk, hvor det pågældende afsnits natur ellers ville være usammenhængende eller på anden måde upræcist hvis skrevet på dansk. Alle efterfølgende kapitler i rapporten vil indeholde en afsluttende delkonklusion, der samler op på, hvad det pågældende kapitel omhandler, og hvad kapitlet har dækket af problemformuleringen:

Analyse og teknologi vil dække over punkterne 1-8, samt punkt 15 jvf. problemformulering.

Design vil dække over punkterne 9-13 jvf. problemformulering.

Implementering vil dække over punkterne 11, 14 og 16 jvf. problemformulering.

Test vil alene dække over punkt 17 jvf. problemformulering.

Bemærk at kapitlerne kan dække flere af de samme punkter.

[7] Larman, s. 33

2 Analyse & Teknologi

I dette kapitel vil der blive gennemført en analyseproces, der fører til en identifikation af aktører, use-cases og en samlet kravspecifikation. En krav-matrix vil sammenfatte use-cases med opstillede krav og en udviklingsplan vil sammen med et fokusafsnit afgrænse projektet. Alle potentielle teknologier, der er relevante for projektet, vil blive fremlagt og diskuteret, ligesom en domænemodel vil blive præsenteret.

Det er vigtigt at gøre opmærksom på, at dette projekt skal ende ud med en prototype, hvor et Proof-Of-Concept senere kan etableres, hvorfor også bl.a. use-cases og krav kun er udfærdiget i simple versioner.

2.1 Kravspecifikation

Under indhentning af krav til en mobil app, der skal supplere et eksisterende produkt, skal man altid give sig god tid til at tænke over hvilken platform, man udvikler det supplerende produkt til. Der findes til stadighed apps, der fejlagtigt blot er en kopi af produktet, kogt ned til lommestørrelse. Man formår ofte ikke at udnytte de mange nye muligheder den mobile platform bringer med sig.

Her henvises der selvfølgelig også til e-conomic's eksisterende mobil app, der forsøger at tilbyde *hele pakken* gennem et hav af lister, tekstfelter og dialog bokse. Det er ikke fordi stor funktionalitet er negativt, tværtimod, i jo højere grad man kan udføre det samme arbejde fra mobilen og dermed *på farten*, som man ellers ville lave ved skrivebordet, jo bedre. Begeret "på farten" er netop nøgleordene i denne sammenhæng. Det nytter ikke noget, at man skal bruge samme mængde tid på at udføre opgaver på den mobile platform, som man ville bruge ved skrivebordet. I stedet drejer det sig om at udnytte de naturlige genveje den mobile platform tilbyder, som f.eks. *swipe-gestures* og *multitouch*, for bare at nævne nogle få. Desuden handler det om at kunne forvandle trivielle arbejdsgange til en simpel bevægelse eller et ryst med hånden.

2.1.1 e-conomic's kunder

e-conomics kundebase består af primært små og mellemstore virksomheder[8]. Netop pga. deres fokus på dette segment, stilles der særligt store krav til brugervenligheden i den software, de tilbyder deres kunder. De små virksomheder har hverken tiden eller ressourcerne til at sætte sig ind i et stort og komplekst system, som tager fokus væk fra netop *deres* kerneforrentning. Netop også derfor nyder e-conomic da også stor succes med deres produkter, da de er tilpasset specielt til *den lille virksomhedsdrivende*.

På figur 2 ses et eksempel på en e-conomic kunde, hvor man har sammenfattet og generaliseret flere kunde-interviews foretaget af e-conomic selv.

Interviews der ligger til grund for figur 2, kan findes i bilag A.

[8] http://www.e-conomic.dk/om/mission

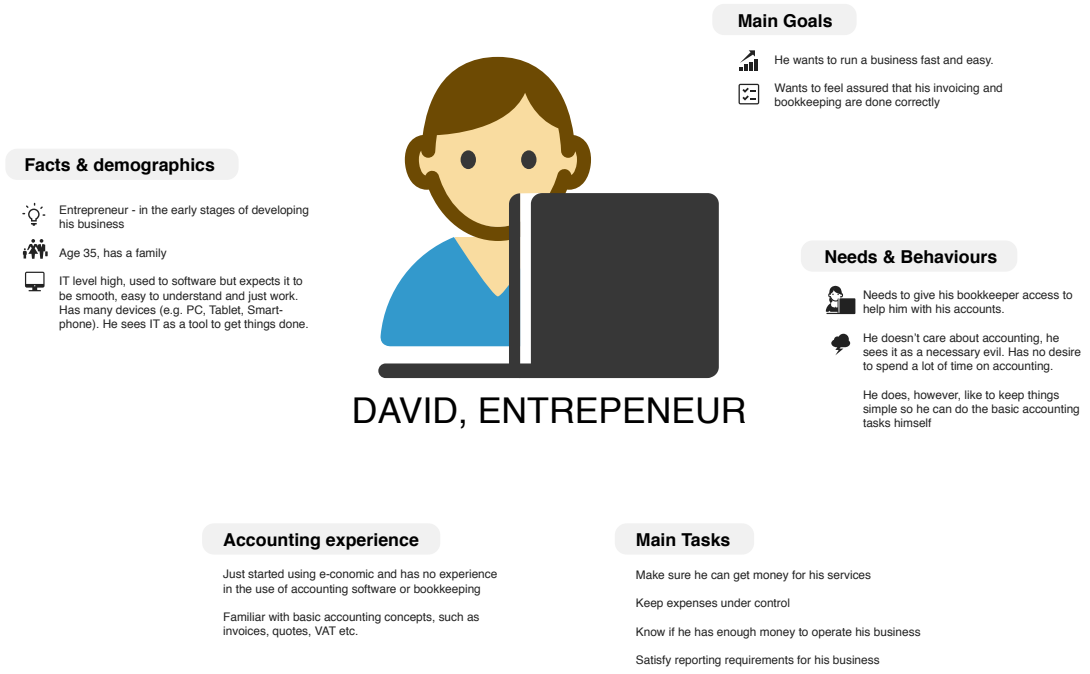


Figure 2: David, the entrepeneur. *Kilde: e-conomic UX Team*

2.1.2 Fokus

Baseret på dialoger med e-conomic, samt de føromtalte brugerinterviews[9], kan man forsøge at udspecificere krav og prioritere enkeltdelene i udviklingen af app'en.

Med fokus på de mest brugte funktioner i e-conomic, og ønsket om at samle disse i en app, der henvender sig til små virksomheder på farten, kan følgende udviklingsområder opstilles:

- Salg
  - Faktura håndtering
  - Ordre håndtering
  - Kundekartotek
  - Varekartotek
- Regnskab
  - Kassekladde, herunder Posteringer med/uden bilag
  - Bilagshåndtering
- Rapportering

[9]Se bilag A

- Omsætnings statistik
- Vare statistik
- Kunde statistik

Området *Salg* er særligt i fokus, fordi det potentielt kræver mere input fra brugeren, end andre områder. F.eks. vil fakturering kræve flere former for interaktion med brugeren, hvor denne skal have forberedt sig på, hvilken af dennes kunder og produkter der skal tilknyttes en given faktura.

Det vil også være i Salg, dette projekt tager sit udgangspunkt.

2.1.3 Aktører og use-cases

Med fakturering som eksempel, vil der i det følgende afsnit blive beskrevet hvilke former for aktører og use-cases, der indtræder i dette område og globalt i app'en i al almindelighed.

Generelt findes der kun få typer aktører, og altså brugere af e-conomic. Dog findes en speciel **administrator** rolle, som ofte er selvstændige bogholdere eller større revisionshuse med en stor kundebase, som gennem dem har oprettet konti hos e-conomic. I første omgang lægges der vægt på, at app'en skal kunne bruges af **slut kunder**, hvilket vil sige, dem som har et fast log-in til e-conomics web interface. En bruger med administrator adgang har adgang til *denne administrators* respektive kunder og kræver dermed en ekstra dimension i udviklingsprocessen. Der er foretaget en designmæssig beslutning om at kunne tilbyde adgang for administratorer gennem app'en på længere sigt, men på nuværende tidspunkt vil der ikke blive fokuseret på at implementere en fuld understøttelse af denne funktionalitet. **e-conomic API** vil også præge projektet og agere som en systemaktør, da den for det første er et eksternt system, og desuden har en central rolle i app'en.

En fakturering indeholder flere trin, hvor de forskellige elementer kan være helt eller delvist afhængige af hinanden.

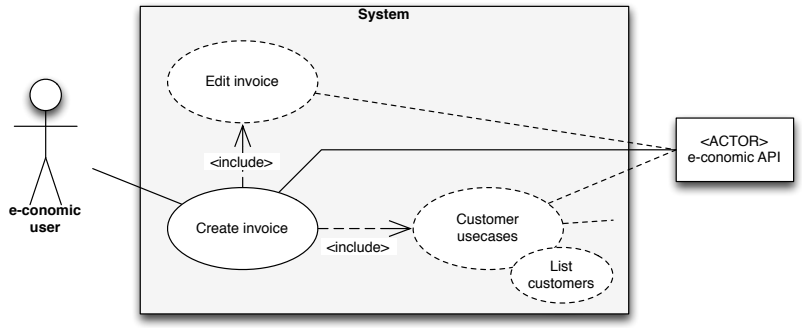
	UC: In1
Use Case Title	Create invoice
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to create an invoice with only the absolute required attributes.
Main Success Scenario	1. user have a ongoing sale with a customer. 2. user creates new invoice. 3. user performs List customers. 4. user selects the customer from customer-list. 4. system signals that invoice is saved 5. user is now performing Edit invoice
Extensions	
Preconditions	The customer must exist in the e-conomic system. (prior to v0.3)
Success Guarantee	Invoice is created in users e-conomic account User can perform Show invoice and see output of invoice,

	UC: In4
Use Case Title	Show invoice
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to see the invoice in its full form, as the customer will see it.
Main Success Scenario	1. user finds a particular invoice. a. user performs List invoice or Search invoice 2. user asks the system for a visual presentation of invoice. 3. system fetches the invoice's visual presentation from e-conomic system and presents it.
Extensions	

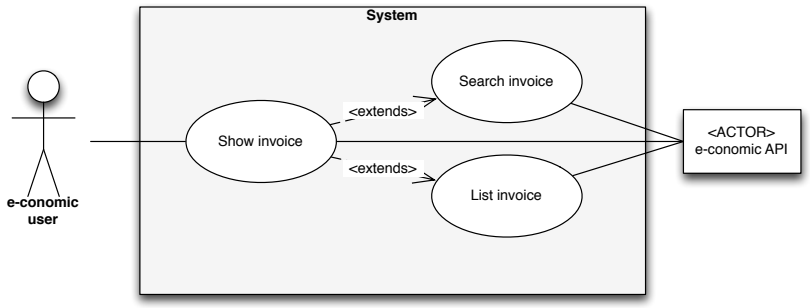
Figure 3: Use-case UC:In1 (opret faktura) og UC:In4 (vis faktura).

Der er udarbejdet use-cases for faktureringsdelen af dette projekt. På figur 3 kan man se nogle resultater af dette. Usecase UC:In1 og UC:In4 omhandler oprettelse og visning fakturaer. Se alle use-cases indenfor fakturering i bilag C.

Diagrammerne på figur 4a og 4b er en visuel præsentation af de foregående usecases på figur 3.



(a) Opret faktura.



(b) Vis faktura.

Figure 4: Usecase diagrammer

2.1.4 Indsamling af krav

I samarbejde med e-conomic[10], er der konstrueret nogle funktionelle og ikke-funktionelle krav til en ny e-conomic app, der udspringer af usecases beskrevet tidligere.

2.1.5 Funktionelle krav

Der kan nu opstilles en række funktionelle krav, baseret på user stories og use-cases, begrænset til fakturerings funktionaliteten:

1 Der skal kunne oprettes fakturaer, der knyttes til en kunde.

- 1a Ifm. oprettelsen af fakturaer, skal man kunne gennemsøge eksisterende kunder og vælge én ud.

2 Eksisterende fakturaer skal kunne redigeres, og/eller slettes.

- 2a Ifm. redigeringen af fakturaer, skal man kunne gennemsøge eksisterende kunder og vælge én ud.
- 2b Ifm. redigeringen af fakturaer, skal man kunne gennemsøge eksisterende produkter og vælge én ud.
- 2c Man skal kunne oprette en ny kunde ved oprettelse af faktura, hvis denne ikke eksisterer. (UC for kunder v0.3+)
- 2d Man skal kunne oprette en ny vare ved oprettelse af faktura(-linie), hvis denne ikke eksisterer. (UC for kunder v0.3+)

3 Man skal kunne bogføre sine fakturaer.

4 Man skal kunne se sine fakturaer i et "færdigt" format.

5 Man skal altid kunne sende/gensende sendte fakturaer.

6 Man skal kunne få vist en liste af alle fakturaer.

- 6a En listning skal kunne filteres på type (bogførte og ikke-bogførte)

7 Der skal kunne søges på fakturaer, bogførte og ikke-bogførte.

- 7a En søgning skal kunne filtreres på nummer, kunde, vare eller dato.

[10] Se bilag B

2.1.6 Ikke-funktionelle krav

Efter interview og lignende samtaler med e-conomics Design- & UX-afdeling, kan der også opstilles en liste med ikke-funktionelle krav:

- Appen skal implementeres på Apples iOS platform med Cocoa Touch Framework.
- Appens operationer må ikke blokere. Dvs. at ingen operationer må hindre brugeren i at interagere med UI, og evt. afbryde en kørende operation.
- Appen UI skal benytte sig af kendte UI mønstre til iOS platformen, således at brugeren føler sig "hjemme" allerede fra første møde med appen.
- Appens grafisk udtryk/design skal afspejle e-conomics design fra browser versionen.
- Appen skal ved alle operationer vise en indikation til brugeren om at en kørende operation i er gang.
- Globalt igennem appen skal der forsøges at mindske nødvendigheden for indtastning (tekst) fra brugeren.
- Der skal tilstræbes brug af forud indtastet data, eller et foretages et kvalificeret gæt hvad brugeren skal til at indtaste.
- Hvor der er mulighed for at erstatte indtastningsfelter med *idoms*, såsom en dato-vælger, skal dette prioriteres.
- Der skal tilstræbes at operationer, der hindrer brugerens videre workflow, eller på anden måde deaktiverer brugeren, såsom downloading af data, ikke varer længere end 4-5 sekunder.
- Er denne øvre grænse ikke mulig at overholde, skal operationen kunne afbrydes, hvis brugeren ønsker dette.

2.1.7 Krav matrix

Tabel 1 viser en kravmatrix der sammenligner de funktionelle krav til projektet (begrænset til fakturering), med alle use-cases som eksisterer i forhold projektets nuværende status. Se de fulde use-cases i bilag C på side 72.

Hvad der skal bemærkes her, er krav 2c og 2d, der tilsyneladende ikke er tilknyttet nogen use-cases. Dette er rigtigt, men på grund af projektet status (version 0.2, jvf. udviklingsplanen på side 22) er det endnu ikke muligt at redigere hverken kunder eller produkter.

Ligeledes er use-cases UC:Cu2 og UC:Pr2, ikke tilknyttet nogle krav. Her kan man diskutere om man i forbindelse med krav 1a, skal kunne få vist detaljeret info omkring en kunde, *inden* men tilføjer denne til en faktura, og ligeledes med krav 2b, hvor man ville kunne se detaljer omkring et produkt, før det blev tilføjet til en faktura. Analysen her er, at man blot skal have nok info tilgængeligt på listen til at kunne bedømme om man har fat i den rigtige kunde, eller det rigtige produkt. Krav 2a omhandler samme diskussion som 1a.

		Requirements													
Use-case	UC title	1	1a	2	2a	2b	2c	2d	3	4	5	6	6a	7	7a
UC:In1	Create invoice	X													
UC:In2	Edit invoice			X											
UC:In3	Book invoice								X						
UC:In4	Show invoice									X					
UC:In5	Send invoice										X				
UC:In6	List invoice											X	X		
UC:In7	Search invoice													X	X
UC:Cu1	List customer		X		X										
UC:Cu2	Show customer														
UC:Pr1	List product					X									
UC:Pr2	Show product														

Table 1: Kravmatrix der viser sammenhængen mellem use-cases og funktionelle krav for faktureringsdelen af projektet.

2.2 Projektafgrænsning

I dette projekt er der planlagt at afgrænse funktionaliteten, således at der ville kunne frigives en tidlig version af appen, med begrænset men dog gennemprøvet funktionalitet. Dette understøtter også tanken om en agil udviklings- og frigivelsesmetode.

Med et fokus på salg, som tidligere nævnt i denne rapport, vil omfanget af dette projekt dog være begrænset til version 0.2 jvf. udviklingsplanen i afsnit 2.3.

2.3 Udviklingsplan

Hermed følgende udviklingsplan:

v0.1 Første udgave. Indeholder kun læse funktioner inden for fakturering, kunder og varer. Grafisk design dog så komplet som muligt

v0.2 Første udgave med skrive funktioner. Primært indenfor fakturering.

v0.3 Skrive funktioner indenfor kunder og varer

v0.4 Rapportering og statistik

v0.5 Regnskab, herunder kassekladde og bilagshåndtering.

Fordi der allerede er en app i drift, der varetager disse funktioner, er der intet større tidspres på disse udgivelser. Men det er klart, at jo hurtigere udgivelserne kommer til at foregå, jo bedre. Især skal v0.2 hurtigt implementeres, da appen ellers ville have et ringe formål hvor der kun kan læses data. Der forbedres også løbende i udviklingen sideløbende med planen, således at fejl eller mangler kan elimineres i opløbet. Der kan også laves større radikale ændringer, hvis denne ses nødvendig for det videre forløb, eller hvis et bedre design kan implementeres på det givne tidspunkt, hvorfor at eksempelvis de beskrevne use-cases også kan blive ændret over tid.

2.4 Netværksarkitektur

Fordi en smartphone enhed har direkte adgang til internettet, kan den nemt kommunikere direkte med e-conomics api servere. På figur 5 ses et blackbox-diagram over netværksarkitekturen, som den kommer til at se ud ved første udgave (v0.1).
Bemærk at der benyttes flere API (-teknologier) til kommunikation mellem enheden og e-conomic. Se mere under afsnittene Systemarkitektur og Teknologi.

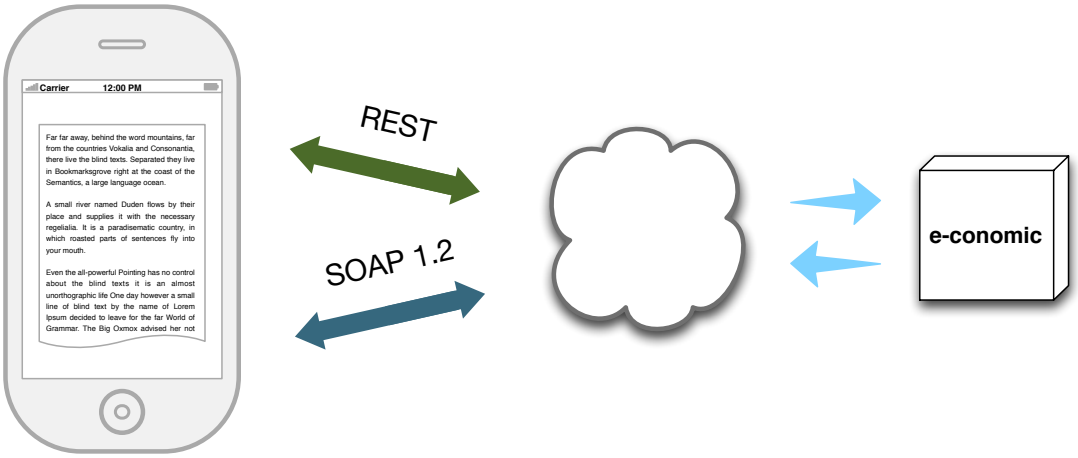


Figure 5: Model over netværksarkitekturen. Enheden har direkte adgang til e-conomic gennem to API'er.

2.5 Systemarkitektur

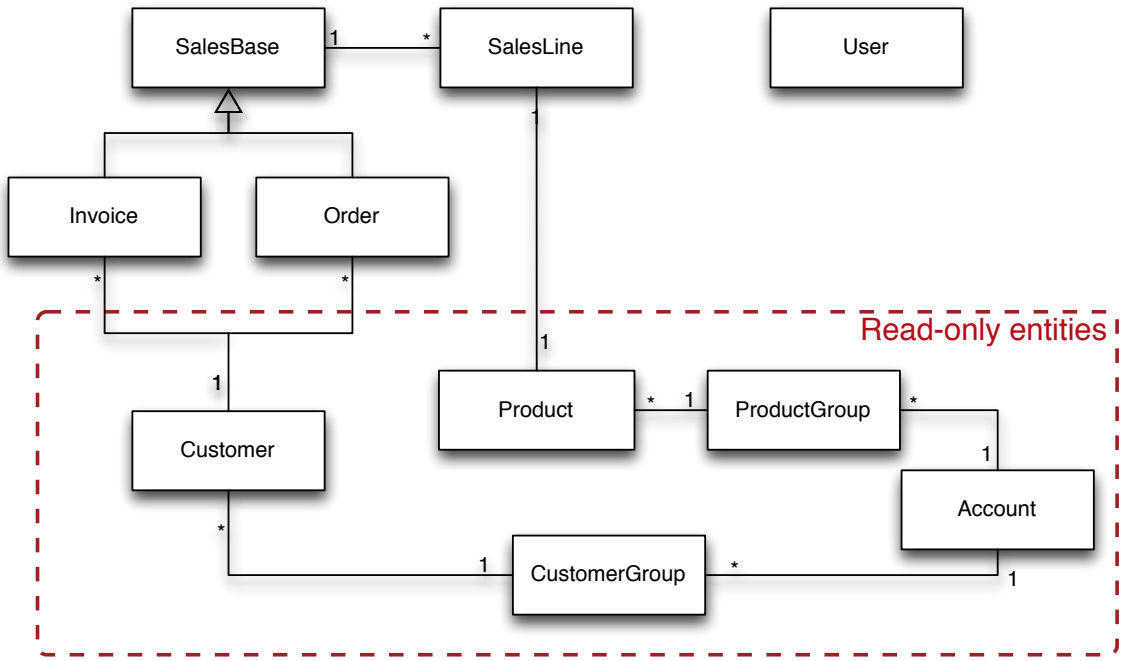


Figure 6: Domænemodel.

Figur 6 viser en domænemodel af *e-conomic mobile*, der er fremstillet ud fra de omdiskuterede use-cases tidligere. Bemærk den røde stiplede linje, der viser hvilke entiteter der ikke skal kunne manipuleres af brugeren i de første udgaver af appen jvf. udviklingsplanen i afsnit 2.3 på side 22.

2.5.1 MVC

Under design og udviklingen af appen udvikles der ud fra design-mønstret Model-View-Controller (MVC), der også er de-facto mønstreret at benytte når man udvikler til iOS platformen. Se mere under Teknologier, afsnit 2.6. Desuden vil der blive fokuseret på at udvikle komponenter som opfylder GRASP-principperne[11].

2.5.2 e-conomic API

e-conomic tilbyder et åbent API som er bygget op omkring SOAP teknologien. Dette understøtter alle use-cases i dette projekt, og vil også være den primære indgangsvinkel til at kommunikere med e-conomic.

I skrivende stund er et nyt API i støbeskeen hos e-conomic, et såkaldt RESTful API. Der er, under udviklingen af denne app givet ekstraordinær adgang til en beta version af dette API. Med tidlig adgang til et kommende nyt API, er der en unik mulighed for at kunne tilbyde en app, der understøtter den nyeste teknologi. Samtidig giver det også e-conomic som udviklingsvirksomhed en god mulighed for at teste det nye api "udefra", selvom det i praksis langtfra er færdigudviklet eller gennemtestet. E-conomic får på den måde mulighed for at ændre eller optimere forskellige rutiner og funktioner i en tidlig fase, bl.a. på baggrund af dette projekt.

[11] Larman, s. 271

2.6 Teknologier

I dette afsnit vil der blive beskrevet hvilke teknologier der er, og har været med i overvejelserne under udviklingen af appen, herunder også på hvilken måde man kan gøre brug af den valgte teknologi.

2.6.1 iOS

I sagens natur bliver man nødt til at nævne iOS, som er navnet på Apples operativ system til alle deres mobile enheder, heriblandt iPod Touch, iPad, iPhone og Apple TV (i en tilpasset udgave). Operativ systemet har efterhånden været i produktion i over 6 år. Skal man udvikle applikationer til dette system, eller andre mobil operativ systemer for den sags skyld, skal man være opmærksom på, at der generelt bliver lagt vægt på *User Interface* (UI), og det i væsentlig højere grad end i andre operativ systemer. Det vil sige at udgangspunktet for alle apps er et veludviklet UI, hvor man på den "stationære" platform ligger vægt på funktionalitet og efterfølgende, hvis det er nødvendigt, ligger et brugervenligt UI som et ekstra hjælpemiddel. I iOS *er* UI og det visuelle i al almindelighed, en del af funktionaliteten.

iOS har gennemgået mange forvandlinger, og udviklinger siden starten. Dels har nye, kraftigere enheder, sat større krav til softwaren, dels har dette også skabt nye muligheder for hvad der kan lade sig gøre på en mobil platform. Gennem årene er iOS og Apples *desktop* operativ system, OS X, blevet mere ensrettet, hvilket gør udviklingen til disse to systemer betydelig nemmere, mere ensartet og mindre tidskrævende.

Af udviklingsmiljø brugers der i projektet *Xcode*, som er Apples eget, og anbefalet udviklingsmiljø til udvikling af al software til Apples økosystem. Inkluderet i Xcode er en meget anerkendt *Interface Builder* (IB), som gør udvikling af UI betydelig nemmere (og sjovere).

Objective C

iOS bygger sin kodebase, eller sit *SDK* og generelle udvikling på programmeringssproget *objective-C* (ObjC), hvilket er et super-set til lav niveau-sproget C. ObjC har været under megen forandring gennem 00'erne, men bygger generelt også på et sprog kaldet smalltalk. Samtidig eksekvering af C og ObjC kode er muligt i iOS, og idéerne fra SmallTalk har gjort, at man "sender" beskeder til objekter, i stedet for traditionelle kald til eller eksekvering af metoder på det respektive objekt. Specielt disse beskeder, gør sproget meget dynamisk, fordi man kan sende beskeder til objekter, man reelt ikke kender typen på. Med denne dynamik og blandt andre det faktum at NULL, som hedder nil i ObjC,f.eks. aldrig er ingenting, men er værdien 0, gør dette ObjC til det mest robuste sprog denne forfatter har set. Mulighederne er mange.

Cocoa Touch

Det klare fundamentet i iOS er et UI framework kaldet *Cocoa Touch*. Dette er et massivt udbygget og veldokumenteret standard bibliotek, der muliggør nem og hurtigt, ens artet udvikling, hvor UI er i centrum. Der er dog rig mulighed for at tilpasse efter projektets behov, og dette kan heller ingen iOS udvikler komme udenom, hvis projektet har bare en nogenlunde størrelse.

Der findes andre moduler eller frameworks, som også er essentielle for udviklingen, bl.a. *CoreAnimation*, *CoreGraphics*. Dog er *UIKit* det primære og også det der som standard er

inkluderet i alle kilder ved oprettelsen af et nyt projekt i Xcode.

Core Data

CoreData[12] er også værd at nævne. CoreData er et data modellerings værktøj, der på en nem måde muliggør hurtig, visuel datamodellering og data udveksling. CoreData kan gøre brug af den allerede installerede SQLite database, der ligger på stort set alle iOS enheder, men kan også konfigureres til andre lagringsformer. CoreData kan også konfigureres til blot at være et abstraktionsværktøj til f.eks. nem håndtering og opdeling af data, som ikke nødvendigvis skal persisteres.

På nuværnde tidspunkt er CoreData ikke tænkt ind i udviklingsprocessen, men på et senere tidspunkt kan det dog have sine fordele at koble CoreData sammen med eksempelvis RESTkit[13], da det ville minimere koden betydeligt.

2.6.2 SOAP Web Services

I et typisk Microsoft .NET miljø har web service standarden SOAP (Simple Object Access Protocol) været meget populær op gennem 00'erne. Store dele af e-conomic er bygget op omkring Microsoft teknologi, hvorfor også deres API er implementeret med det XML-baserede SOAP (2.0). Skønt dette gør det nemt at udvikle med et hvis abstraktionsniveau med hjælp fra proxyklasser, har SOAP en del overhead, og gør det besværligt at bruge på andre platforme end .NET.

For at bruge et SOAP API i iOS, kræver det at man manuelt konstruerer de forskellige proxyklasser fra bunden med strenge, komplet med headers, body, osv. Skulle man implementere samtlige af e-conomics godt 1500 API funktioner til iOS, ville dette være en uoverskuelig opgave. Der er dog til projektet fundet en russisk side[14] der tilbyder gratis kodegenerering af wsdl webservices. Resultatet er en genereret kodebase der beskriver objekter til alle entiteter, og som indeholder metoder, der automatisk omskriver data til SOAP formatet og vise versa. Da SOAP API'et efterhånden bliver afløst, ses denne løsning som et tilfredsstillende dog *midlertidig* alternativ, indtil alt service funktionalitet kan lægges over på det kommende REST API.

2.6.3 RESTful

e-conomics kommende nye api, er bygget op omkring REST "standarden".
REST (Representational-State-Transfer)[15] ikke en egentlig standard, men blot en populær betegnelse for en primitiv, dog kraftfuld måde at kommunikere via HTTP protokollen i en klient-server arkitektur – oftest sammen med json-formateret data. HTTP protokollen bliver ved brug af REST, udnyttet til sit fulde potentiale, hvilket giver nogle klare fordele, både når vi taler om skalerbarhed og ydelse. Ved at have et REST Api, giver det også bedre integrations muligheder på tværs af alle platforme og udviklingsmiljøer. Dette sikrer også en forholdsvis hurtigt implementering af en given integration med et andet system, således at man kan fokusere på det vigtige i ens projekt.

[12] `https://developer.apple.com/technologies/ios/data-management.html`
[13] Se afsnit 2.6.4
[14] `http://www.wsdl2code.com/`
[15] `https://en.wikipedia.org/wiki/Representational_state_transfer`

2.6.4 RESTful with iOS

I appen implementeres som sagt understøttelse af begge typer API. Skal man kommunikere med et REST API i iOS, er det til forskel fra SOAP dog ikke den helt store opgave. Der findes et tredjeparts bibliotek til iOS; open-source projekt ved navn `RESTkit`[16], som bl.a. indeholder en velbygget *object-mapper*, koblet på respektive endpoints. Dvs. at har man en given data model i sit domæne f.eks. `Customers`, og et API endpoint f.eks. `http://<baseURL>/customers`, kan RESTkit konfigureres til automatisk at linke en `Customer` model sammen med det det specifikke endpoint `/customers`, således at der oprettes objekter af `Customer`, svarerende til de antal entiteter der returneres. Objekterne kan konfigureres meget detaljeret, så man kan opstille regler for hvilke værdier der skal tolkes på hvilke måder. F.eks. kunne man bruge denne egenskab, hvis ens domænemodel ikke helt ligner datamodellen der modtages fra API'et og en bestemt attribut derfor skal gemmes et andet sted i et andet objekt, eller blot helt ignoreres.

RESTKit skal med sikkerhed bruges i dette projekt.

2.6.5 Xamrian (MonoTouch)

Med i overvejelserne var også Xamrian[17], tidligere kendt som MonoTouch. Xamrian er et værktøj til at udvikle bl.a. iOS apps i .NET C#, og stadig drage nytte af iOS's *native* UI. Her kan man altså kode i C#, udnytte alle fordele ved .NET platformen herunder også teknologier såsom LINQ og events, og stadig bruge den anerkendte interface-builder i Xcode. Koden vil i dette tilfælde, blive kompileret til ægte iOS Intermediate Language eksekverbar kode. Dermed kan man genbruge ret så store dele af ens kodebase, når der skal udvikles til andre mobile platform som f.eks. til Andriod, hvor der så kompileres til java platformen istedet. I sagens natur understøtter Xamrian ikke Windows Phone.

At bruge "ægte" værktøjer og programmeringssprog til en given platform, vil dog i denne forfatters optik altid være det bedste, hvis ressourcerne er tilstede og kravspecifikationen er meget specifik. Selvom Xamrian er en veludviklet platform, og mange anerkendte virksomheder gør brug af denne, er det desuden en del af problemformuleringen til dette projekt, at udvikle *fully native* til iOS.

[16] `http://restkit.org/`
[17] `http://xamarin.com/`

2.7 Delkonklusion

Der er i dette kapitel blevet fremlagt to forskellige API'er, som begge skal bruges i app'en. SOAP- og REST API'et, skal begge implementeres således at man på sigt kan helt kan udfase SOAP API'et, hvilket er en forældet teknologi. Ved at forberede til REST allerede nu, vil app'en have et klart forspring når et fuldt REST-baseret API bliver en realitet. Dette svarer på spørgsmål 7 i problemformuleringen.

Gennem dette kapitel er forskellige teknologier blevet diskuteret og vægtet. Xamrian som platform er blevet helt afvist, fordi det ikke stemmer overens med hverken dette projekts problemformulering om et fuldt iOS miljø. SOAP og REST skal begge kunne bruges i appen, men kræver forskellig implementering. SOAP laget bliver kode-genereret, imens REST bliver implementeret vha. RESTkit.

Det er endnu usikkert om CoreData skal bruges i appen, selvom der reelt ikke skal persisteres noget data i appen. CoreData vil på længere sigt være fordelagtigt, at benytte denne teknologi på et senere tidspunkt, SOAP API'et kan udfases, og REST API'et dermed står alene. CoreData kan kobles direkte på RESTkit. Dette besvarer spørgsmål 5 og 8 i problemformuleringen.

Udviklingsmønsteret der bruges i under dette projektet, vil være Model-View-Controller. Skønt iOS dikterer brugen af dette mønster i rammeværktøjet, passer dette mønster også godt til projekttypen. Dette besvarer spørgsmål 6 i problemformuleringen.

I kapitlet er der også sat et fokusområde for projektet hvorpå funktioner i e-conomic er listet. Disse funktioner er valgt til at være de vigtigste at tilbyde på en mobil platform. Der er opstillet funktionelle og ikke-funktionelle krav til app'en på baggrund af dette fokusområde, samt use-cases for faktureringsdelen. Med udspring i faktureringsdelen vil denne liste også være målet for et evt. videre forløb med dette projekt. Dette besvarer spørgsmål 3 i problemformuleringen.

e-conomics slutkunder er blevet beskrevet og påvist som potentielle brugere er app'en. Administrator-brugere af e-conomic er også beskrevet, men disse brugere vil ikke kunne bruge app'en i første omgang. Dette vil desuden kræve en større større håndtering af adgang til data, og er iøvrigt uden for dette projekts rækkevidde. Dette besvarer spørgsmål 4 i problemformuleringen.

Det er desuden blevet fastlagt, hvilke udfordringer der er tilstede med den eksisterene app. Gennem et interview med e-conomic (Bilag B), er en af de største udfordringer hastigheden og responsiveness. Det er blevet fastlagt at dette vil blive sat i højsædet, dels ved at bruge et fuldt iOS miljø da dette generelt højner hastigheden ift. en web-app med HTML5, og dels ved opsætte krav omkring et ikke-blokerene UI. Udnyttelsen af det nye REST API, vil også potentielt hæve hastigheden yderligere. Dette besvarer spørgsmål 1, 2 og 15 i problemformuleringen.

Den fremlagte krav-matrix i dette kapitel viser, at de fastsatte funktionelle krav er blevet opfyldt, pånær 2c og 2d. Krav 2c og 2d er ikke planlagt at implementere i denne udgave af app'en, men indgår først som en del af udviklingenplanen i version 0.3 jvf. udviklingsplanen i afsnit 2.3, side 22.

3 Design

I dette kapitel vil det tidlige design af e-conomic mobile blive beskrevet, og argumenter for designvalg vil blive præsenteret, understøttet af flere designdiagrammer.

3.1 Teknologivalg

Vil man udvikle en native app til iPhone eller iPad, er iOS i sagens natur en del af teknologivalget, som man ikke kan pille ved. Når det er sagt, har der som skrevet tidligere været overvejelser omkring at bruge tredjeparts værktøjer, til at kunne gøre udviklingen mere fleksibel – også på længere sigt. Til dette projekt er der dog valgt at udvikle *fully native*, som gør udviklingen mere målrettet og enkel.

SOAP API'et bliver stadig en nødvendighed, da det kommende REST API i skrivende stund er under udvikling. Det havde skabt større værdi for projektet, hvis REST kunne implementeres til alle use-cases, men dette var dog medregnet i projektets indledende fase. Muligheden for at bruge kode-genereret SOAP proxyklasser vil blive udnyttet, men den genererede kode er dog langt fra optimeret og fejlfri, hvorfor der også i en tidligt konceptuelt design, har været behov for at omskrive forskellige steder af koden manuelt. Kodebasen for SOAP "laget", er også meget stor og derfor er der allerede nu overvejet en refaktorering af denne, da det også kan påvirke den faktiske størrelse på den færdige app. Sammenlagt er der blevet genereret ca. 100.000 linier kode, hvilket alt sammen kan skæres fra, når REST API'et kommer i produktion.

I skrivende stund brugers der ikke CoreData i appen. På et senere tidspunkt kan det dog have sine fordele at koble CoreData sammen med RESTkit, da det kan minimere koden betydeligt og RESTkit i øvrigt aktivt understøtter kobling til CoreData.

3.2 Systemdesign

iOS platform lægger op til en udvikling der ligger sig stramt til MVC mønstreret og derfor er der ikke megen afvigelse herfra i designet af *e-conomic mobile*. Med undtagelse af implementeringen af API laget, vil systemstrukturen være stærkt koblet til iOS. Dette har sine ulemper, så som at *unittests* kun vil kunne skrives til kode, der ikke omfatter UI kode. Det vil sige, at alt fra UIKit biblioteket, som er størstedelen af al UI i iOS, vil blive svært at test via normale unittests, da man i så fald skulle lave stærke koblinger til UIKit i de forskellige tests. I unittests, er det ikke særlig hensigtsmæssigt at teste UI kode, med mindre det netop er denne kode der skal testes. Her ville man dog benytte andre metoder. Der vil derfor i projektet tilstræbes at tillægge ansvaret for forretningslogikken til modellerne. Dette gør modellerne testbare og man behøver derfor ikke at inkludere controllerne i testkoden.

Udover testbarhed, overholder denne struktur også GRASP-principperne, især området omkring "information expert" hvor en given model har netop det ansvar, der passer bedst til modellen og det data, ansvaret påkræver.

3.2.1 API

e-conomic mobile bliver grundlæggende bygget op omkring eksterne API'er. Derfor kommer datamodellen for projektet også til at afspejle, og enkelte steder helt kopiere datamodellen i e-conomics API.

3.2.2 Entiteter

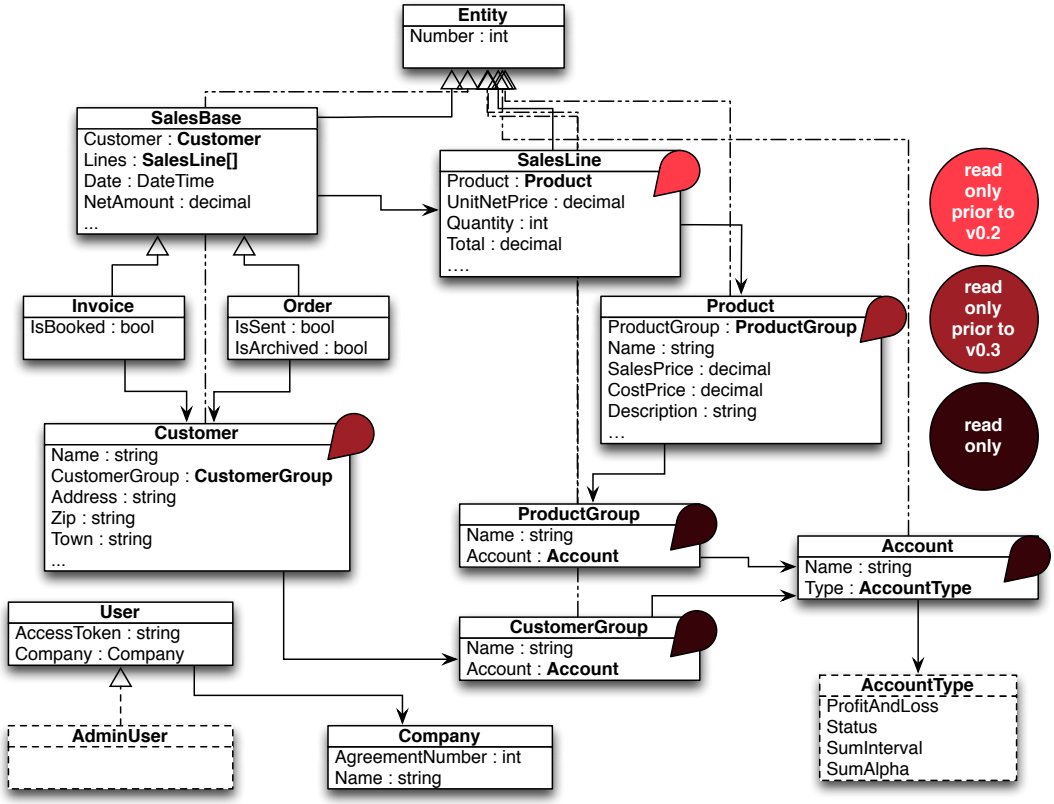


Figure 7: Designklassediagram over e-conomic mobile.

Diagrammet på figur 7 giver et billede af data-modellen i e-conomic mobile. Attributter er kun medtaget her i en sådan grad at forståelsen af modellens funktion i appen gør sig gældende.

I diagrammet ser man bl.a. nedarvningen fra `SalesBase`, der beskriver et *salgs dokument*. Fordi en faktura og en ordre minder om hinanden rent designmæssigt, ville det også være naturligt at disse delte størstedelen af deres attributter. Skønt en ordre ikke er en del af scope for dette projekt, er det klart at der skal forberedes til denne entitet.

`SalesBase` er også et godt eksempel på "protected variations" indenfor GRASP, fordi der oftest ikke er brug for at vide den præcise type af salgsdokumentet, for at eksempelvis at udføre en visning af dokumentet. (En use-case for "vis orde" vil være en næsten tro kopi af UC:In4) Derved bliver de nedarvende type beskyttet, ved blot at bruge `SalesBase` som en form for interface.

3.2.3 Kodegenerering for SOAP

Resultatet af kodegenereringen fra SOAP API'et, er en relativ stor samling af klasser forbundet til to store service klasser, selve service klassen og en såkaldt proxyklasse. Proxyklassen vil være klassen man kalder fra sin forretningslogik, ved enhver kommunikation til e-conomic api'et. Proxyklassen er en stor klasse og er som navnet antyder, en proxyklasse der omslutter service klassen, således man ikke behøver håndtere hverken url og data parsing. Serviceklassen håndtere selve omskrivningen, eller "serialiseringen" om man vil, fra de forskellige api-dataklasser til SOAP formatet. Figur 8 viser resultatet af en sådan kode-generering.

For ikke at have stræk kobling til de generede dataklasser i *domænet*, skal der udvikles en speciel *object mapper*, der kan transformere data fra en SOAP data klasse og til en domæne dataklasse og vise versa.

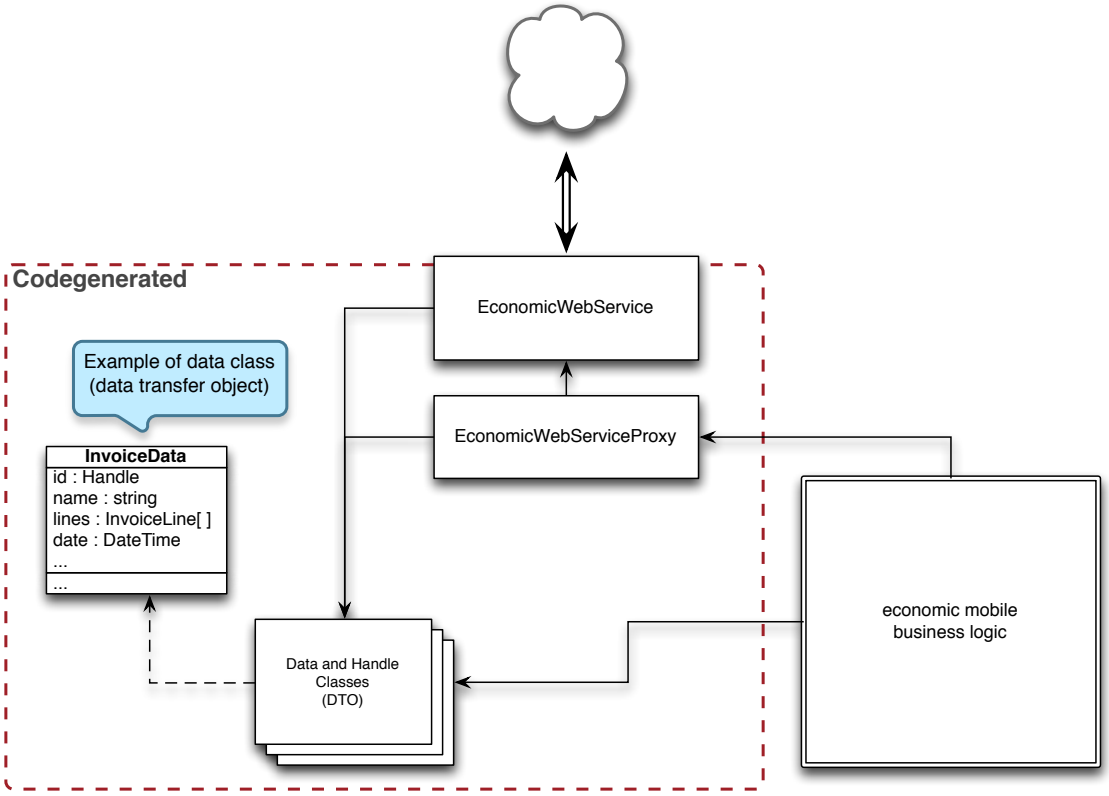


Figure 8: En SOAP service. Det færdige resultat af en kodegenerering.

3.2.4 REST Design

For at imødekomme det nye REST API, vil der i projektet skulle forberedes en håndtering af kommunikation til dette. Tredjeparts løsningen RESTkit er passende til dette formål, og indeholder tilmed værktøjer til object-mapping. Dette muliggør en automatisk kobling mellem domænets datamodel og REST API'et, ved hjælp af en simpel og kompakt konfigurations klasse. Bemærk at der ved RESTkit ikke er samme lag af *proxy*-dataklasser tilstede som ved SOAP.

Figur 9 viser et samlet diagram over et servicelag, som både understøtter et REST- og SOAP api. På diagrammet kan man se at der ikke er behov for en domæne-specifik object-mapper til RESTkit, til forskel fra SOAP. Klassen `ManualMapperManager` sørger for denne mapping til og fra SOAP API'et.

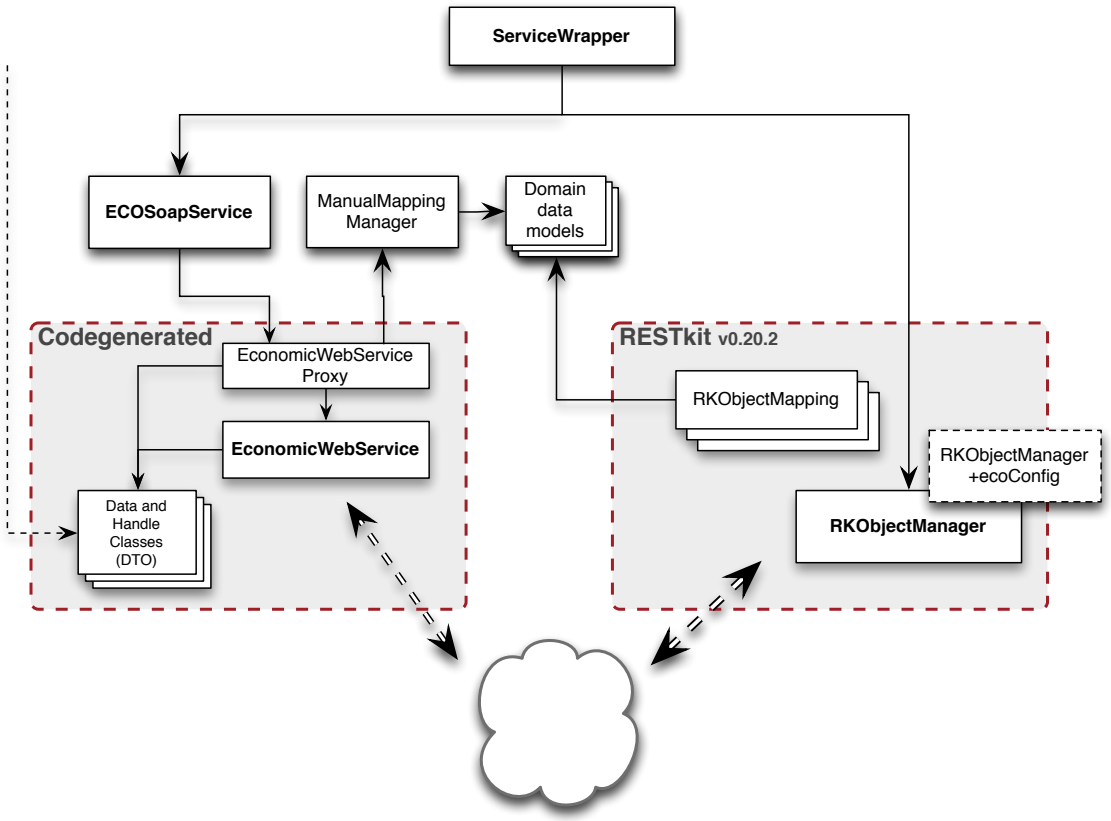


Figure 9: Diagram over servicelagets opbygning i e-conomic mobile.

3.2.5 Systeminteraktion & Systemflow

At opbygge et system der understøtter to vidt forskellige API teknologier har behov for en stærk afkobling fra service-laget. Derfor oprettes en service container-klasse der sørger for kommunikationen med både SOAP og REST, således at der kun er behov for at kalde én klasse fra forretningslogikken. Dette opfylder desuden "indirection" indenfor GRASP, for hvilket man bruger containerklassen som "mellem"-objekt imellem forretningslaget og servicelaget,

således man forhindrer direkte kobling. Containerklassen kender til domænets data-model så der let kan hives relevante data ud, og pakke dem forskelligt alt efter hvilket API der benyttes.

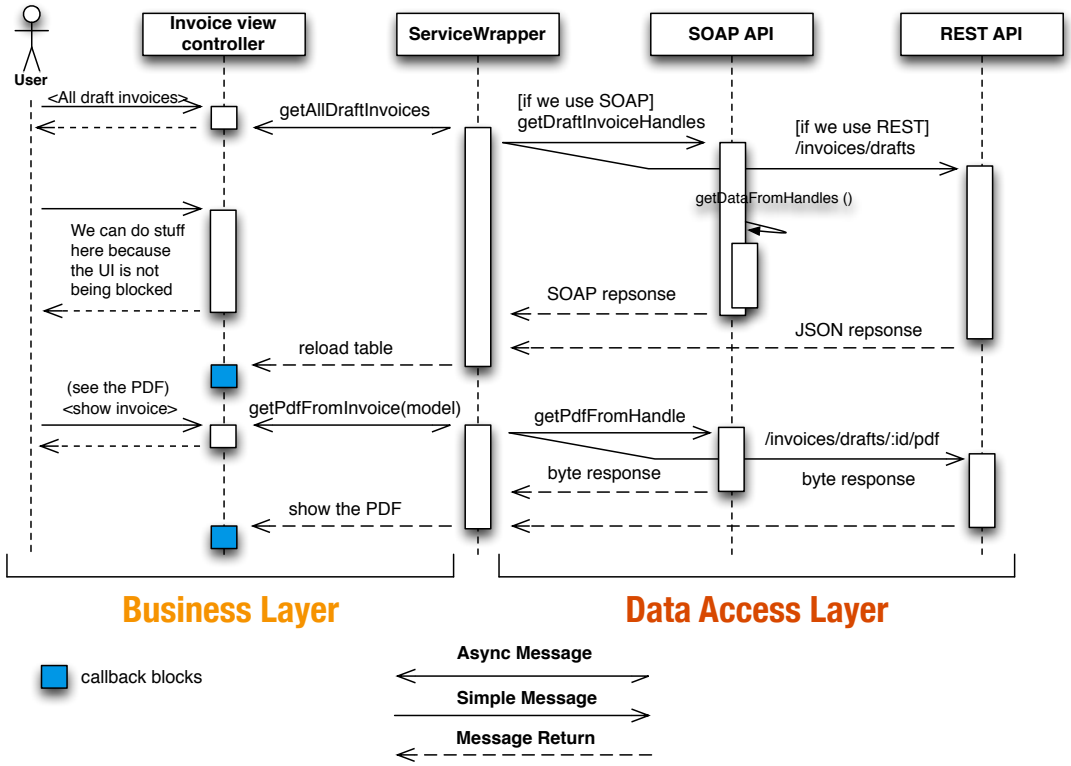


Figure 10: Sekvensdiagram der viser hvordan en fælles serviceklasse omslutter begge API'er. Brugeren beder først om en liste af kladde fakturaer, for dernæst at vælge en ud til PDF-visning. *Diagrammet er simplificeret.*

Sekvensdiagrammet på figur 10 viser denne afkobling af forretningslogikken fra datalaget. Diagrammet viser også gode eksempler på UC:In6 (list alle fakturaer) og UC:In4 (visning af faktura). Bemærk at de to API'er skal bruge forskellig information afhængig af hvilken operation de skal udføre. Bemærk også hvordan retur data er af forskellig format. Dette påvirker ikke controlleren - eller hvad der kalder metoden - som blot kalder de samme metoder, ligegyldigt hvilket API der benyttes. De blå sekvenser der kan ses er *blocks* af kode der registeres til senere eksekvering, når svar modtagers fra API'et. Denne *funktions programmering* er med til at opnå et ikke-blokerende UI, da al API trafik bliver behandlet i seperate tråde.

3.2.6 Systemstruktur

Som omtalt tidligere, bygger iOS' standard biblioteker på en stramt MVC kobling, og dikterer dermed også udvikleren til at kode med dette udviklingsmønster for øje. Dette kan for alvor ses i figur 11. Efter gennemgangen af de forskellige dele af systemet, giver figur 11 tilmed et godt helheds billede af strukturen, og sammenspilet imellem.

Man kan diskutere hvorvidt en given model skal have adgang til service laget. Det vil sige

at man vil kunne bede modellen om selv at udføre alle handlinger, som lige nu er pålagt den tilhørende controller. Praktisk hænger det sammen at hver model selv kan sørge for at persistere sig selv, hvis den vel at mærke skal dette. Derved behøver en controller ingen kendskab til data kilden, hvad end den er en webservice, lokal database eller noget helt tredje. Dette ville tilmed underbygge pricippet om et "Creator"-mønster i GRASP, da servicen stortset kun skal bruges i samspil med en datamodel.

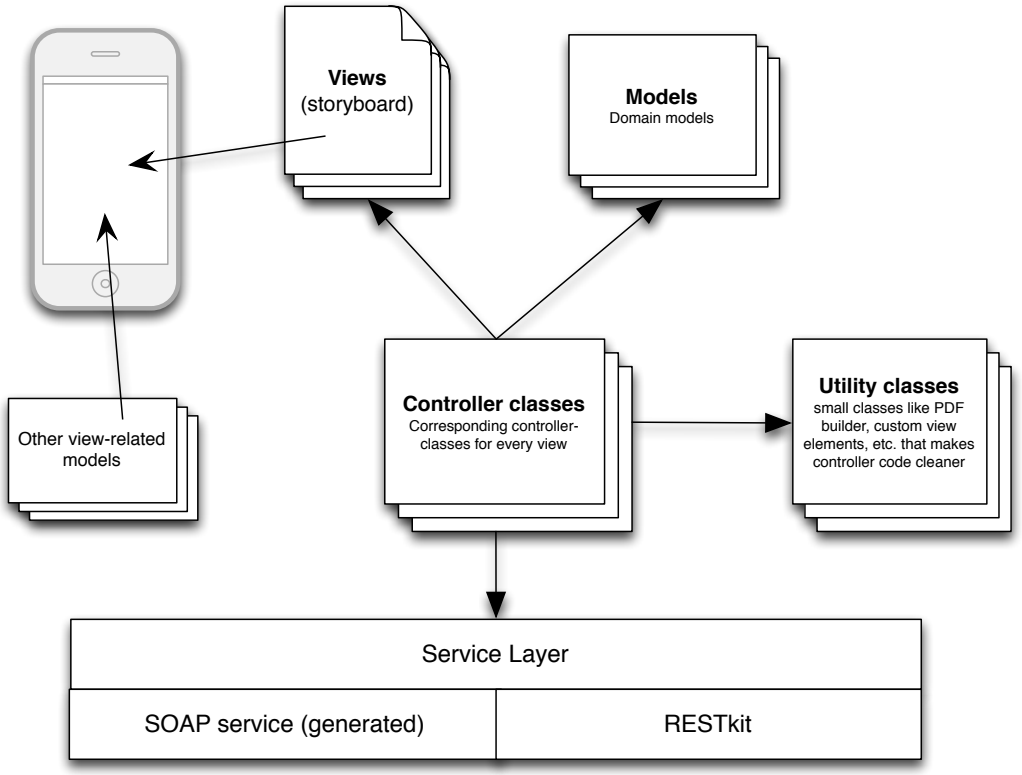


Figure 11: MVC + Servicelag i i iOS.

3.3 Brugergrænsefladedesign

iPhone og iPad er blevet meget populære indenfor de sidste fem år. En af årsagerne er iOS og den måde systemet er bygget op på. Som udvikler kan man relativt hurtigt komme op med et design der er let forståeligt for alle mennesker i alle aldersgrupper, hvis man følger Apples design-retningslinjer[18]. Dog er "standard-designet" efterhånden en noget forslidt fornøjelse at kigge på, hvorfor Apple dog også har annonceret et helt nyt standard design i iOS 7, som kommer til efteråret. Skønt der i dette projekt er adgang til beta udgaver af iOS 7 og andet eksperimentelt materiale fra Apple, er der dog ikke planlagt nogen tilpasning til denne udgave på nuværende tidpunkt. Desuden vil det højst sandsynligt ikke kræve et særlig stort re-design, hvis man netop gør brug af standard UI elementer.

[18] `https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html`

Brugerfladen og det generelle grafiske udtryk vil lægge sig op af det grafiske materiale, som lægger til grund for e-conomics nye grafiske linje i deres webapplikation. Det hedder sig i farver, ikoner og andet særligt genkendeligt i e-conomic økosystem.

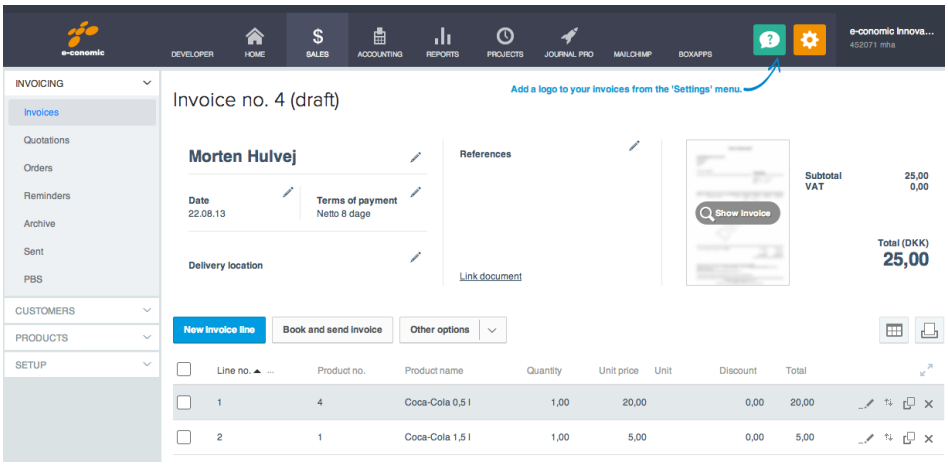


Figure 12: e-conomic browser applikation.

3.3.1 Opbygning af GUI

Siden iOS 5, er *storyboards* blevet tilgængelige i udviklingen af UI til iOS, hvilket gør design af workflows i iOS betydeligt nemmere. Dette projekt vil selvfølgelig også gøre brug af storyboards. Samtidig giver dette nemlig også mulighed for nem udvidelse indenfor UI til en senere iPad version af appen, hvor stortset hele kodebasen kan genbruges.

Kendte og velafprøvede designmønstre for den mobileplatform[19] vil også blive overvejet for at imødekomme projektets ikke-funktionelle krav.

[19] Tidwell, kap. 10

3.3.2 Navigation

Appen kommer generelt til at vise data. Meget data. Derfor skal der visse steder tænkes særligt kreativt, når data skal præsenteres for brugeren, således at brugeren har et godt overblik, men stadig har så meget data tilgængeligt som muligt på en lille skærm som på en iPhone. Da man i dette projekt ikke besidder de store grafiske evner, er der derfor undersøgt, og søgt inspiration i andre apps, der har lignende datamængder.

Efter adskillige prøvedesigns[20], var det tydeligt at navigationen imellem de forskellige komponenter såsom faktura, produkter og kunder ville blive et problem. Med den velkendte *tabbar* ville der hurtig mangle "pladser" og tabbaren i sig selv vil optage kostbar plads på skærmen. Med en navigationsbar som også er ret brugt i iOS, ville man ende op med dybe navigeringsruter, hvis en bruger skulle ønske at komme tilbage til hovedmenuen. Den rette navigation i appen vil blive essentiel for en succes, og derfor blev en speciel navigations model valgt, som her kaldes for *slide-menu-navigation*. Denne form for navigation er især blevet kendt fra Facebook appen og andre apps med meget forskelligt data, der skal kunne vises og kunne tilgås, på en hensigtsmæssig måde. Denne navigationsform kaldes også for en pyramide[21].

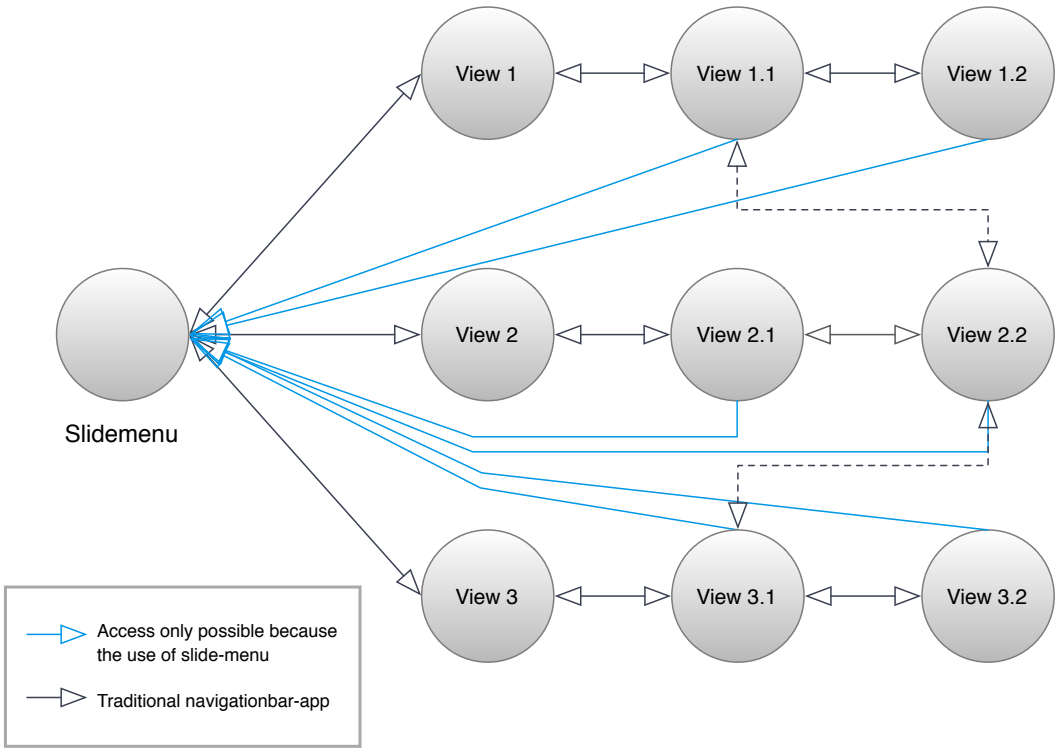


Figure 13: Navigationsdiagram der viser forskellen mellem traditionel navigationsbar-mønster og slide-menu navigations-mønster.

[20] Se bilag D
[21] Tidwell, s. 94

Figur 13 illustrerer hvordan brugeren ved bruge af en slide-menu har adgang til alle dele af appen, uanset hvor brugeren befinder sig. Med denne navigations type, kan brugeren altså altid komme tilbage, samt der spares kostbar plads på skærmen.

Figur 13 viser i øvrigt også at slide-menuen egentlig er en traditionel navigationcontroller-app, bare med flere fordele. Alle iPhone-brugere vil nikke genkendende til dette navigationsmønster. Med sin karakteristiske navigationsbar i toppen, der altid viser en "tilbage" knap, guider denne type af navigation brugeren tilbage den vej han kom fra. Dette kan godt blive et irritationsmoment for brugeren, hvis han er dybt inde i en bestemt visning og bare gerne vil tilbage til udgangspunktet. Ved brug af slide-menuen kan han nu nemt komme helt tilbage ved at swipe menuen frem fra venstre.

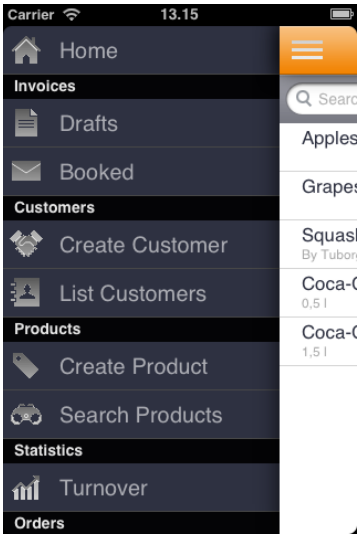


Figure 14: En åben slidemenu i e-conomic mobile.

3.4 Delkonklusion

I dette kapitel er der blevet gennemgået hvilke design elementer der skal benyttes i appen. En oversigt over et systemdesign er blevet fremlagt og ud fra bl.a. sekvensdiagram er der blevet præsenteret et systemflow, der kan bygges videre på. På baggrund af disse diagrammer kan man se at SOAP- og REST API'et vil blive implementeret gennem et interface-modul, der skal agere som en fælles indgang til hele servicelaget. Dette er med til at sikre en hel uafhængig implementering af app'en. Servicelaget kan således nemt skiftes ud, og selve app'ens implementering kan nemmere porteres til andre platforme hvis dette ønskes. Dette besvarer spørgsmål 10 i problemformuleringen.

Der er blevet fastlagt hvilken form for UI der skal implementeres, og nærmere hvilke elementer der skal bruges til dette, samt hvilken navigation. En special-designet *slide-menu* navigation er blevet fremlagt som kernen i app'en GUI. Denne navigationsform gør navigationsmulighederne i app'en mere fleksible, og samtidig udnytter velkendte UI-design på iOS platformen, herunder navigationsbar. Dette besvarer spørgsmål 13 i problemformuleringen.

Meget vigtigt er der også fastsat hvilke teknologier der skal bruges og hvordan, når både SOAP og REST API'et skal implementeres. RESTkit skal bruges til REST og *wsdl2code*[22]'s kodegenererings værktøj skal bruges til SOAP. Dette besvarer spørgsmål 9 i problemformuleringen.

Det er nu med sikkerhed konkluderet at CoreData ikke skal bruges i appen i denne omgang. CoreData vil have værdi for appen, men først når der udelukkende bruges REST API og dermed RESTkit. I første omgang benyttes der object-mappers til datahåndtering: Dels bruges RESTkits indbyggede mapper og dels den til formålet konstruerede SOAP object-mapper. Der er på nuværende tidspunkt ikke set nødvendigt at gemme forrentningsdata lokalt på enheden. Dette besvarer spørgsmål 11 og 12 i problemformuleringen.

Iøvrigt er der diskuteret om hvorvidt ansvaret for API kommunikation skulle ligge hos controllere eller hos modellerne. Der blev fundet gode argumenter for at ligge ansvaret hos den enkelte datamodel, der på sin vis "ved bedst" omhvad der skal sendes og modtages. Dette er dog ikke inkluderet i det nuværende design, og vil derfor være en af de første ting at kigge på, i et videre forløb med dette projekt.

[22] `http://www.wsdl2code.com`

4 Implementering

I dette kapitel vil der dokumenteres e-conomic Mobiles konkrete implementering samt vise eksempler i koden af særlig karakter, der underbygger design-beslutningerne foretaget tidligere i processen.

4.1 Cocoa Touch

På figur 15 ses et klasse diagram over et udsnit af appens implementation. Igen tages der udgangspunkt i faktureringsdelen.

På figuren ser man en nedarving fra klasserne `EI_BaseTableViewController` og `EI_SalesBaseTableViewController`, hvilket bliver brugt som fælles base for alle lister, og desuden er abstrakte. `EI_SalesBaseTableVC` er dog kun til faktura (og ordrer) og deres respektive viewcontrollers.

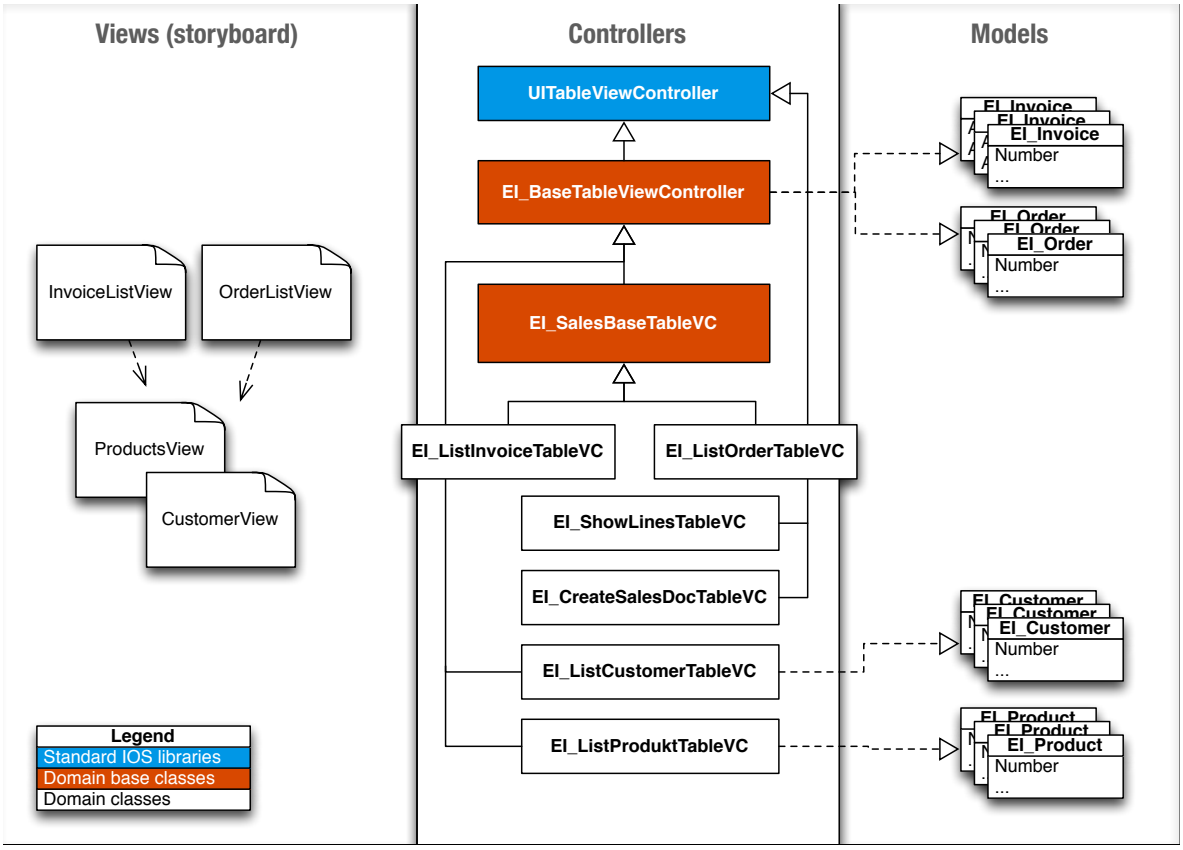


Figure 15: Klassediagram over fakturerings logik, der viser en MVC-opdeling.

Til venstre på figur 15 ses de forskellige views, som ikke skal forstås som klasser, men som elementer i UI-storyboard der beskriver alt hvad der har med UI at gøre. UI udvikling i iOS kan også sagtens opbygges programmelt, men dette synes tåbeligt når man først har set styrken i Xcode's Interface Builder.

Controller-klasserne der ikke nedarver fra baseklasserne, kunne man argumentere for skulle gøre dette. Funktionaliteten er primært den samme, ligesom den er det samme hos ordrer- og fakturaklasserne, og grunden for en manglende baseklasse er en prioriteringssag. Med stigning

i mængden af redundans i koden, stiger behovet dermed også for at udtrække en baseklasse for de respektive klasser, hvis antal jo også bliver større efterhånden som appen udvikler sig.

4.1.1 Xcode

Xcode v4.6.3 bliver brugt til udvikling af e-conomic Mobile. iOS version, eller det specifikke *target* er sat til iOS v6.1. Generelt giver Xcode både et primitivt og meget stærkt udviklingsmiljø. Primitivt, fordi man til forskel fra andre miljøer på andre platforme, ikke finder de store analyse-værktøjer eller andre større tillægs programmer bygget ind i Xcode, som man kender det fra f.eks. MS Visuel Studio pakken. Der findes dog udemærkede tillægsprogrammer til Xcode som hedder *Instruments* som kan bruges til dette formål. På den anden side er Xcode et stærkt miljø, pga. dets integrerede design til test, profiling og normal kørsel af projekter på tværs af enheder, samt selvfølgelig Interface Builder.

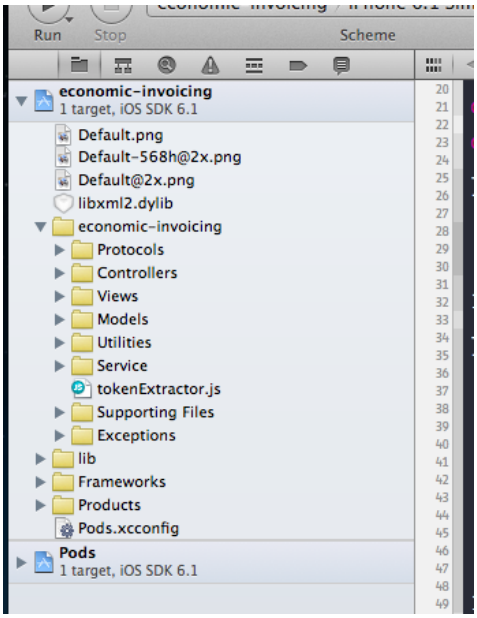


Figure 16: Mappe struktur i Xcode.

Billedet i figur 16 illustrerer hvordan projekt strukturen er sat i Xcode. Projekt-træet vidner om en stærk opbygning efter MVC mønstret.

4.2 Services

Servicelaget, eller Data Access Layer om man vil, udgør al kommunikation med de to API'er vi benytter i e-conomic Mobile. `ServiceWrapper` er en klasse der som navnet antyder *wrapper* begge API'er. På denne måde får vi afkoblet hele vores forretningsmodel, så vi ikke er bundet op på en bestemt data kilde. Service laget har ikke ændret sig siden designet. Der henvises til figur 9 på side 34.

4.2.1 SOAP

Proxyklassen og serviceklassen, hvilket er blevet kodegenereret, er i sagens natur ikke modificeret. Det har dog løbende været nødvendigt at rette nogle fejl opstået under kodegenererin-

gen, hvorpå der var tilføjet ekstra elementer i det færdige soap+xml output, der forårsager et fejl-svar fra e-conomics servere. Disse ændringer er implementeret ad hoc.

Desuden var fejlhåndtering i proxyklassen ikke særlig informativ, idet der blot returneres samme fejlbesked for alle service kald, hvis der ingen data kom retur fra API'et. Ved debug af en til formålet fremprovokeret fejl, blev det fundet at der kommer udemærkede fejlbeskeder fra API'et, som blot bliver ignoreret i proxyklassen.

Der er i den forbindelse lavet en *Exception parser*, der trækker relevante fejl information ud af fejlbeskeden fra API'et. Her iblandt en fejlbesked direkte fra serveren samt en fejlkode, der kan bruges til evt. en fremtidig validering af om brugere har adgang til diverse tillægsmoduler, jvf. et fejlkode skema fra e-conomic. Bemærk at der skulle tilføjes et kald til parseren i hvert af de godt 1500 service metoder. Dette kunne heldigvis gøres nemt med Xcode's find/erstat-funktion.

4.2.2 REST

Implementeringen af RESTkit er forløbet uden nogen problemer. Med tilføjelsen af konfigurationsklassen `RKObjectManager+ecoConfiguration`, som også indeholder mapping indstillingerne for domænet, kan REST API'et bruges.

For dette projekt, er en udviklingsserver blevet stillet til rådighed. I skrivende stund understøtter e-conmics nye REST API dog kun funktionalitet for hvad der svarer til v0.2 jvf. udviklingensplanen for dette projekt. Skønt dette er passende i forhold til projektet nuværende status, er REST API'et endnu ikke i et produktionsmiljø og kan derfor ikke bruges af slutbrugere.

4.3 Model

Designet af domænets datamodeller har hovedsageligt ikke ændret sig under projektet indtil videre. Der er tilføjet flere attributter og i øvrigt tilføjet flere nedarvinger, men det grundlæggende design er der ikke pillet ved.

Grundstenen i datamodellen er `EI_Entity`, der beskriver en entitet som brugeren aktivt er i interaktion med. Det kan være en faktura-linje eller en kunde. Alle databærende entiteter, der afspejler forretningsmodellen og dermed ligner API modeller, er præfikset med "`EI`", således der ikke kan opstå forvirring imellem klassenavnene. Udover disse entiteter er der også platform-specifikke modeller, som f.eks. `AppSettings`, der er en beskrivelse af nogle globale indstillinger der kan sættes i appen. Derudover er der også søge-modeller, nogle modeller til fejlhåndtering og ikke mindst modellen `AppUser` der indeholder information omkring brugerens session og adgang i appen. Se mere omkring adgang i afsnit 4.7. `Appuser` og `AppSettings` er de eneste modeller hvis data bliver gemt på iOS enheden. Alt andet data bliver hentet *on-the-fly*, og bliver bevaret i hukommelsen de steder det er nødvendigt. Jvf. udviklingenplanen eksisterer der på nuværende tidspunkt ikke flere modeller end hvad der er højst nødvendigt.

På længere sigt kommer der flere modeller til, som allerede er kendte, men som alligevel ikke ville gøre nogen nytte at tilføje på nuværende tidspunkt.

4.4 Object Mapping

Dette afsnit vil redegøre for object-mapping for hhv. SOAP- og REST-API'et.

4.4.1 SOAP Mapping

Adgang til datakilde eller ej, skal modellerne stadig fyldes med data. Når der kaldes data fra API'et, hvad enten man bruger SOAP eller REST, skal det modtaget data *mappes*, således der bruges domænets egne modeller. Ved brug af SOAP API'et, kræver det at der genkendes hvilke data der skal i hvilke modeller.

Den genererede proxyklasse giver kun én parameter med tilbage; en tekststreng med et servicekald navn, der fortæller hvilket service kald der hænger sammen med til returnerede data. Denne parameter kan man ikke umiddelbart oversætte til nogen tilhørende modelklasse, med mindre der implementeres en betingelsesalgoritme, der afhang af hvilken streng der skulle resultere i en given model. Dette ville resultere i mange `if-else` udtryk; en for hver af API'et funktioner, hvilket vil sige godt 1500.

At skulle implementere 1500 endpoints samt alle datamodeller som skal bruges i samme forbindelse, er selvfølgelig udelukket, da arbejdet ville være alt for tidskrævende og i øvrigt et *rigtig* dårligt design. I stedet er der lavet en selvstændig klasse der gør arbejdet mere effektivt. Klassen `ManualMappingManager` sørger for at registrere servicekald navnet sammen med en modelklasse. Når der senere modtages data, sammenlignes den model der er modtaget, med dette register. Hvis findes et match, kaldes en bestemt metode på *domæne* datamodellen, der starter en mapping mellem de rigtige værdier i date-transfer-objektet til de rigtige attributter i data klassen. Listing 1 viser denne metode, med fakturaklassen `EI_Invoice` som eksempel.

```
1  ...
2  - (void)fillWithDataFromTransferObject:(id)model
3  {
4      // map all baseclass properties
5      [super fillWithDataFromTransferObject:model];
6
7      if ([model isKindOfClass:[CurrentInvoiceData class]])
8      {
9          CurrentInvoiceData *m = model;
10         self.number = m.iId;
11         self.handle = m.handle.iId;
12
13         self.booked = NO;
14     }
15     else if ([model isKindOfClass:[InvoiceData class]])
16     {
17         InvoiceData *m = model;
18         self.number = m.number;
19         self.handle = m.handle.number;
20
21         self.booked = YES;
22     }
23     else {
24         [self raiseException:model];
25     }
26 }
27 ...
```

Listing 1: Klassen `EI_Invoice` der modtager data fra et API-objekt.

Bemærk at der i netop `EI_Invoice` sker flere ting; først ser man om der kan mappes attributter i superklassen, hvilket der ofte vil være i en faktura i denne domænemodel. Dernæst kommer der et særtilfælde hvor to transfer objekter skal mappes til én modelklasse. `CurrentInvoiceData` og `InvoiceData` beskriver begge en faktura, men hhv. for en kladdefaktura og en bogført faktura i e-conomic's regi. For at gøre domænemodellen så enkel som mulig, er der her valgt at samle disse data.

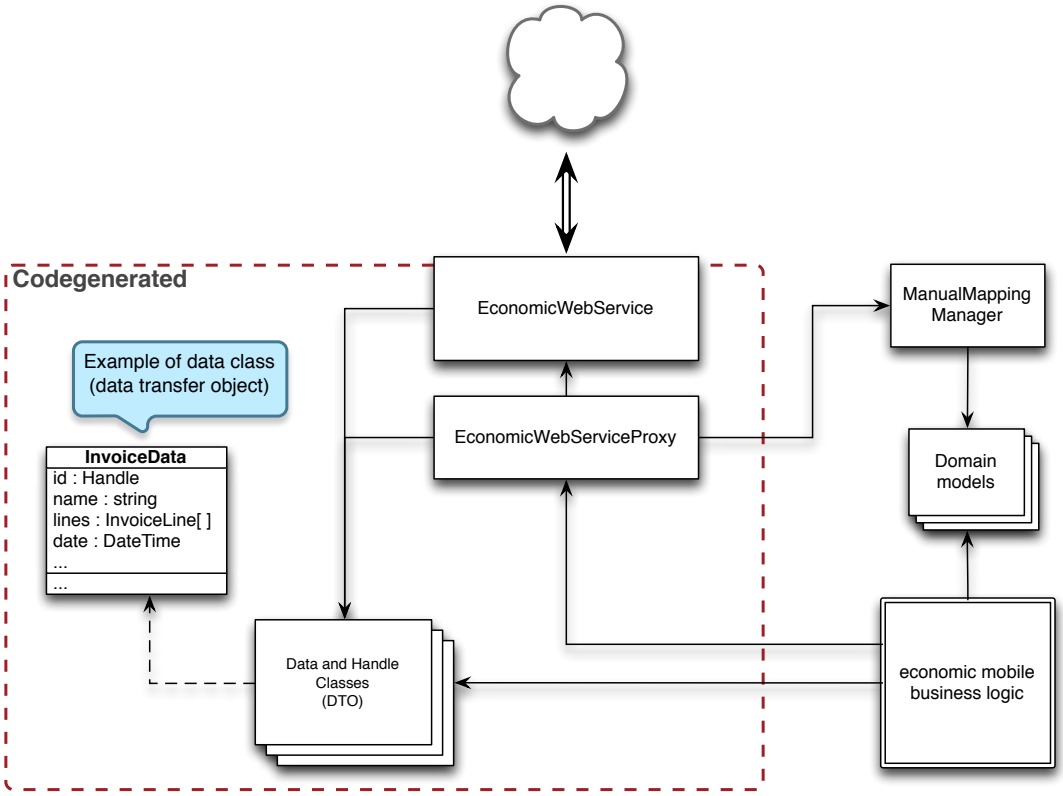


Figure 17: SOAP wrapper med tilføjet object mapper.

Koblingen af disse transfer objekter og modelklasser, sker som sagt i `ManualMappingManager`, hvor to simple arrays, `dataModels` og `domainModels` indeholder typer der skal mappes. Disse to arrays udgør tilsammen det føromtalte mapping-register. F.eks. er første element i `dataModels` er typen `CurrentInvoiceData`, der skal mappes med EI_Invoice fra før. EI_Invoice udgør så dermed den første plads i `domainModels`. Det videre flow sammenligninger så blot `dataModels` med den specifikke type der er modtaget gennem API'et og findes denne i arrayet, tages dens modpart i `domainModels` og fylder den med data ved at kalde mapping metoden `- (void)fillWithDataFromTransferObject:(id)model` direkte på det nyskabte objekt. Da det er typer der vedligeholdes i disse arrays, er der allerede derfor en type der kan skabes et objekt ud fra.

Da der kun er lavet en mapping den ene vej, dvs. når data modtages, bliver der nødt til også at være en "omvendt" mapping, hvor processen går baglæns.

Al denne manuelle mapping er trivielt arbejde, og der findes forskellige *auto-mappers* der ville kunne lette arbejdet, men grundet appens indtil nu beskedne data maængde og fordi

SOAP API'et alligevel er på vej til udfasning, bedømmes dette til at være en tilfredsstillende løsning.

Fortsætter man baglæns i mapping processen, ender man i `servicewrapper`-klassen, hvor data modtages, og hvor den implementerede service logik findes. På listing 2 kan man se koden der sørger for en mapping af indkommende data. Man kan også se at hvis mappingen mislykkedes, forsættes der blot med de oprindelige data. Dette er til for at primitive datatyper kan modtages. F.eks. returneres der blot en tekststreng og ikke et array når man kalder metoden `connect` på API'et. Dette behøver ikke at mappes og skal blot returneres råt. Se listing 2.

```
1  ...
2  - (void)processCallback:(NSString *)method withData:(id)data orError:(NSString
       *)reason
3  {
4      ...
5      id callback = [self.registeredCallbacks objectForKey:method];
6      ServiceArrayCallback serviceCallback = (ServiceArrayCallback)callback;
7
8      // map the objects to domain models
9      id mappedData = [ManualMappingManager findDomainModelByTransferModel:data
           ];
10     if (mappedData){
11         serviceCallback(mappedData, reason);
12     }
13     else     // try to return the original data, if mapping failed (the data
           maybe is not to be mapped)
14     {
15         serviceCallback(data, reason);
16     }
17     // dereg the callback
18     [self.registeredCallbacks removeObjectForKey:method];
19         ...
20 }
```

Listing 2: `ECOService` sørger for både at sende, men også modtage og behandle svaret fra SOAP API'et.

I listing 2 ses i øvrigt også variabelen `registredCallbacks`. Denne er essentiel for hele SOAP service laget, da denne indeholder kodeblokke til senere eksekvering, når vi modtager data. Disse kodeblokke er gemt med metodenavnet som nøgle. Som før beskrevet modtages metodenavnet sammen med data, og dermed kan der nu kobles forskellige callbacks på de rigtige data. Dette er vigtigt hvis man fortager flere servicekald efter hinanden, hvor disse kald ikke nødvendigvis returnerer i samme rækkefølge som de blev kaldt.

4.4.2 REST Mapping

Noget helt andet er når man bruger REST API'et. Bruger man RESTkit, tager implementeringen af en mapping-mekanisme væsentlig kortere tid, sammenlignet med SOAP. RESTkit har en indbygget object-mapper og når man har konfigureret denne mapper, er API'et klar til brug. Der er til formålet lavet en `category`, som kan sidestilles med extensions i .NET. `RKObjectManager+ecoConfiguration` konfigurerer RESTkit til at bruge det rigtige endpoint, samt indeholder nogle header indstillinger og selvfølgelig alle objectmappings.

```
{
    ...
    RKObjectMapping *invoiceMapping = [RKObjectMapping mappingForClass:[
           EI_Invoice class]];
    [invoiceMapping addAttributeMappingsFromArray:@[
     @"date",
         @"dueDate",
         ...
     // all properties are mapped here
     ]];
    ...
    // the mapping is registred to a specific url-path
    RKResponseDescriptor *invoiceResponse = [RKResponseDescriptor
           responseDescriptorWithMapping:invoiceMapping
        pathPattern:@"/invoices"
            keyPath:nil
        statusCodes:[NSIndexSet indexSetWithIndex:200]];
    [self addResponseDescriptorsFromArray:invoiceResponse]
    ...
}
```

Listing 3: Objectmapping for `EI_Invoice` for REST servicen.

Listing 3 er et eksempel på en mapping. Denne mapping sættes direkte på domæne modellen, og sammenfattes med et url-endpoint, således at RESTkit ved at hvis vi tilgår `/invoices`, skal data der modtages, mappes til klassen `EI_Invoice`.

4.5 View

Alle views er konstrueret i Xcode Interface Builder, og er linket sammen med de tilhørende view-controllers. I dette afsnit beskrives hvordan de forskellige views er sammensat ved hjælp af *storyboards* i iOS.

4.5.1 Interface Designer

Ved hvert view man opretter i Interface Builder (IB), følger der en ViewController klasse med som er dens "kodebase", og den er som navnet antyder, dens controller. Et view skal være inden i en viewcontroller-container *eller* inden i et andet view. Hvis ikke andet angives bliver `UIViewController` sat som viewcontroller. Det er så meningen at man skal nedarve denne i egne viewcontrollers og henvise til denne klasse i IB.

Man kan udnytte de såkaldte *storyboards* til at trække *segues* imellem ens viewcontrollers, således man mindsker behovet for at kode en transaktion til et andet view eller en anden handling. Skønt dette også nemt kan gøres rent programmelt, er der så vidt det er muligt benyttet denne metode, hvilket resulterer i renere viewcontroller-kode.

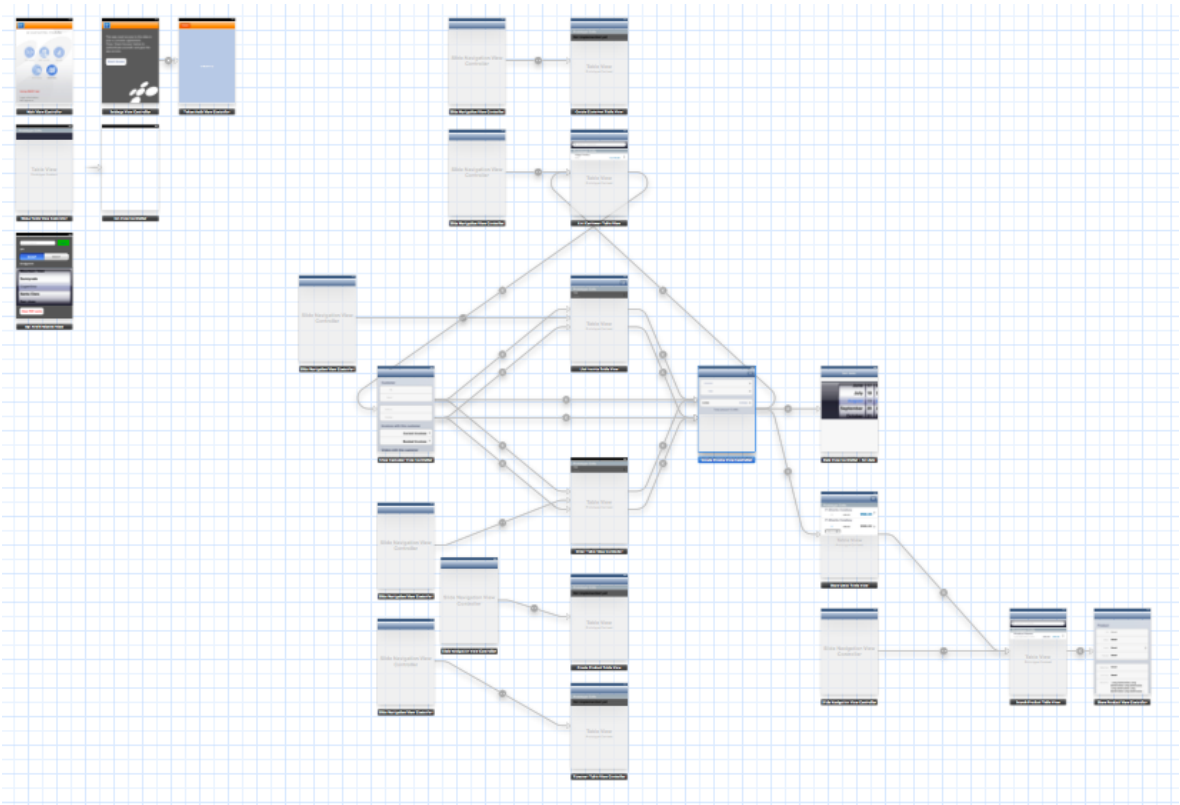


Figure 18: Dette fulde stroyboard set fra Xcode.

Figur 18 viser hele storyboardet der bruges i appen. Man kan se en gruppe af viewcontrollers der ligger isoleret fra de andre. Hvor de ligger i storyboadet er underordnet, men de er bevidst plantet her, fordi de ikke bruger den indbyggede segue-transaktion. I stedet bruges en tredjeparts viewcontroller kaldet `ECSlideViewController`, hvilket er den føromtalte slide-menu-navigation. Se figur 13.

Enkelte viewcontroller klasser bruges af flere views i IB. Dette er f.eks. `EI_CreateSalesDocTableVC` og `EI_ShowLinesTableVC`, der indeholder funktionalitet for både faktura og ordrer. Selvom funktionaliteten er den samme, kan det godt være at det skal se forskelligt ud i UI. Dette muliggør storyboards i IB. Det samme gør sig gældende, hvis man senere vil lave en iPad version af appen. Her vil man lave et nyt storyboard og lave nye views, men bruge samme kode.

I storyboardet nedarves der også fra `UINavigationController` og udbygger denne til projektets egne formål. Brugeren vil se den velkendte navigationsbar i toppen, samtidig med at der fås nogle meget brugbare værktøjer med "gratis". Bl.a. får man ved at bruge segues automatisk en "tilbage" knap, der fører brugeren tilbage til det forrige view(controller) der var i fokus. På den måde kan der opbygges en hel stak af viewcontrollers, der vedligeholdes, således man altid kan komme samme vej tilbage igen. Dette er særligt smart hvis man vil genbruge views som allerede eksisterer. Er man ved at oprette en faktura og skal tilføje en kunde fra ens kunde kartotek, ville det være dumt at bygge et nyt view, når der allerede eksisterer et view til at liste en kunder. Der trækkes blot endnu en segue.

På figur 19 kan man se rigtig mange segues der er trukket i forskellige retninger, men navigation controlleren skal nok holde styr på hvor brugeren kom fra. Hvis man først har lavet de rette views, kan man altså genbruge dem, alle steder i appen. På slide-navigation diagrammet (figur 13) viser de stiplede pile netop disse mulige genveje der kan laves i appen.

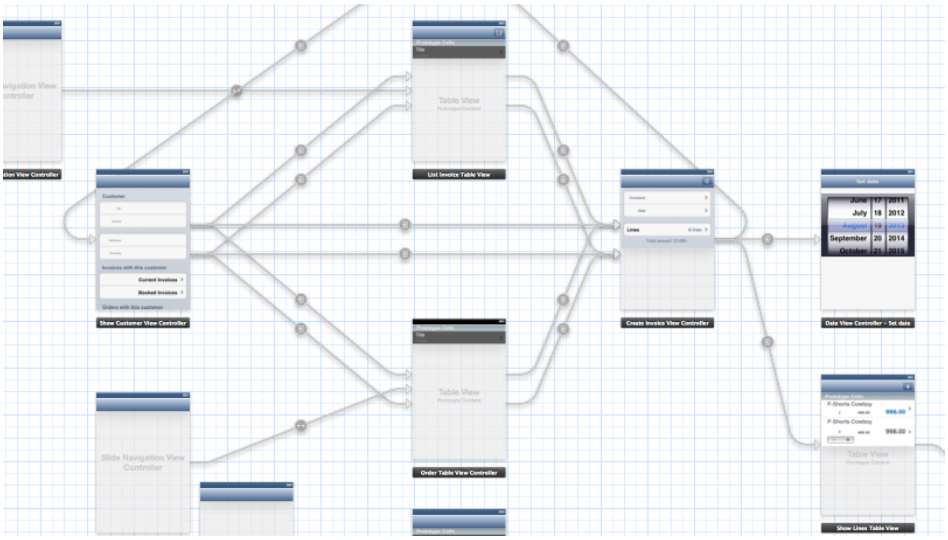


Figure 19: Del af storyboard der omhandler faktura.

4.5.2 Lister

Appen kommer til at skulle vise meget data, og derfor skal denne data også struktureres fornuftigt, til fordel for brugeren. Da appen i sine planlagte udgaver kommer til at skulle vise meget af datamængden i form af lister, er det vigtigt lister i sig selv ikke bliver en hæmsko for brugren. For at imødekomme uundgåelige udfordringer med den store datamængde tilføjes forskellige designmønstre som f.eks. *Infinite List*[23]. *Text Clear Button* og *Loading Indicators*[24]

[23] Tidwell, s. 462
[24] Tidwell, s. 467-468

er tilmed elementer der med sikkerhed vil blive implementeret.

Den uendelige liste er især god til at imødekomme nogle af de ikke-funktionelle krav omkring *responsiveness*, hvor langtidskørende operationer skal begrænses. Hvis data præsenteres på en liste, skal dette begrænses til hvad der bliver synligt. Hvis brugeren kræver mere - ved eksempelvis at "rulle" ned til enden af listen, hentes yderligere data automatisk. Se eksempel på figur 20. Dette kaldes ofte for *lazy-loading*.

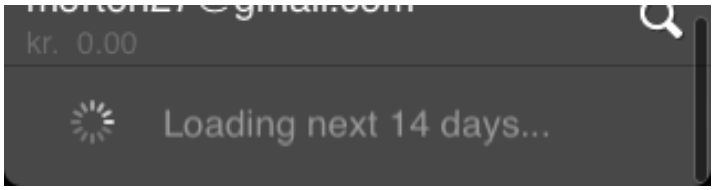


Figure 20: En "uendelig" liste der løbende udvides med data, efterhånden som det kræves.

4.6 Controller

ViewControllers er kernen i iOS og i Cocoa Touch framework. Generelt kan man kun lave meget begrænsede apps, uden at skulle skrive sine egne view controllers.

Fordi views er bundet tæt op på disse view controllers, behøver man ikke koncentrere sig om hvorvidt hvilket view der skal bruges hvornår. Det er alt sammen beskrevet i storyboardet og indgår i en viewcontrollers livscyklus. Der bruges som vist på figur 15, nedarving af viewcontrollers til at generalisere andre viewcontrollers i appen. På nuværende tidspunkt kan der refaktoreres en del mere, således at endnu mere kode kommer op i baseklasserne.

Det er disse viewcontrollers der har adgang til vores servicewrapper og derfor har kontrol over API kommunikationen. Som diskussionen tidligere i afsnit 3.2.6 er dette måske ikke hensigtsmæssigt, da controlleren reelt ikke har brug for denne reference.

Listing 4 viser et service kald foretaget fra controller. Dette fungerer godt, men for at imødekomme MVC og generel godt design, ville et kald til modellen selv, give bedre kode. Viewcontrollers kobling til servicelaget ville helt forsvinde, men dog skal callbacks stadig kunne defineres i viewcontrollers, da det returerede data stadig hører hjemme i controllerne.

```
1  ...
2  [self.service createSalesModel:self.salesModel :^(id data, NSString *error) {
3          if (error){
4          // An error happened. Inform the user
5                  [ModalDialog showConnectionErrorSavingFailed];
6          } else {
7          // Call succeeded. Save reference to the received data.
8                  EI_SalesBase *savedInvoice = data;
9                  self.salesModel = savedInvoice;
10         }
11         [self updateView];
12 }];
13 ...
```

Listing 4: Kode i `EI_CreateInvoiceViewController` der udgør et API kald, hvor der er angivet et callback som bliver udført når API-kaldet returnerer. Her oprettes en ny faktura.

Ved hvert service-kald, kan der sendes et callback med som argument. Det skal være typen `ServiceArrayCallback` eller `ServiceCallback`, som er defineret for at for en blok-lignende

servicekalds procedure. Disse callbacks er næsten udelukkende defineret i viewcontrollerne, og er iøvrigt dem der lagres i `registredCallbacks` beskrevet tidligere.

4.7 Adgangskontrol

For at en e-conomic bruger kan få adgang til hans e-conomic konto data, skal han godkende appen, i et flow der meget ligner det man kender fra OAuth[25] standarden. Ved e-conomic får man dog et gyldigt token med det samme, og der behøves således ikke et out-of-band POST. En bruger bliver som det første henvist til en godkendelses side, hostet hos e-conomic. På denne side godkender brugeren at netop *denne* app, e-conomic mobile, får adgang til netop denne bruger. Denne procedure sikrer at brugere ikke skal give trejdepart (denne app eller anden app-partner) deres e-conomic login oplysninger, ved at de indtaster disse på e-conomics egne servere. En app-partner har derved ingen mulighed for at opbevare disse informationer, og dette højner sikkerheden for brugeren. Appen modtager, hvis en godkendelse altså lykkedes, en unik *access-token*, som tilgengæld skal gemmes af appen. Med denne token kan appen på vegne af brugeren nu tilgå data hos e-conomic. Brugeren kan afvise adgangen til appen på hvilket som helst tidspunkt. Hvis brugeren afviser adgangen, kan appen ikke længere tilgå data hos e-conomic, for netop *denne* bruger.

4.8 Delkonklusion

I dette kapitel har implementationen af API'er og brugerfladen domineret indholdet.

Håndteringen af data fra API'erne er beskrevet, og hvorpå data-modellen samler data fra to forskellige formater. De forskellige object-mappers vil transformere data frem og tilbage imellem app'ens data-model og hhv. SOAP API'ets data-transfer-objects og REST API'ets primitive json data. RESTkit leverer en indbygget object-mapper, tilforskel fra SOAP API'et der havde brug for en selvstændig object-mapper opbygget fra bunden. Tager man tidshorisonten for REST API'ets frigivelse i betragtning, kan der konkluderes at arbejdet med en manuel object-mapper er givet godt ud. Efterhånden som elementer af REST API'et kommer i produktion, vil der gradvis kunne overføres funktionalitet til REST API'et, og SOAP API'et kan langsomt udfases. Dette besvarer spørgsmål 11 i problemformuleringen.

Brugergrænsefladens sammensætning er præsenteret, og hvorvidt denne imødekommer de ikke-funktionelle krav for projektet. Specielt er der beskrevet brug af elementer som *lazy-loading* og *infinite-lists*, der kan bidrage til at håndteringen af rigtig store datamængder glider bedre igennem app'en. Dette besvarer spørgsmål 14 i problemformuleringen.

Adgangskontrol til brugerdata er også blevet fremlagt. Adgang til e-conomics systemer kan kun gives til allerede oprettede e-conomic brugere. Brugere skal give app'en adgang til deres egne data der ligger hos e-conomic. Dette gøres med et enkelt loginflow via e-conomic hjemmeside, igennem appen selv. Dette fritager app'en for at håndtere følsomme adgangsdata. Der er ikke blevet diskuteret om hvorvidt, der skal gives mulighed for at oprette konti igennem app'en. Dette besvarer spørgsmål 16 i problemformuleringen.

[25] `https://en.wikipedia.org/wiki/OAuth`

5 Test

I dette kapitel vil både manuelle tests, unit-tests og app-tests blive vist. Desuden vises hvordan man kan indsamle nyttig information og herunder test-brugers færden i appen ved hjælp af TestFlight SDK.

Der har i dette projekt ikke været den store erfaring med test i iOS. Derfor er der i starten af udviklingsforløbet kun benyttet manuelle tests gennem iOS Simulatoren samt på en rigtig enhed. Skønt at Test-Driven-Development (TDD) er en tiltalende udviklingsform, og man havde da også gerne set projektets udvikling drevet i denne retning. En TDD tilgang til udviklingen ville dog kræve at der i starten af implementeringen var det samme kendskab til iOS platformen og ikke mindst til *unit-tests* i iOS, som der er nu, hvor vi nærmer os afslutningen af dette indledende projekt. Det er klart at der er oparbejdet en solid hands-on erfaring med iOS-udvikling under dette projekt og vil derfor nu kunne ty til et mere testorienteret udviklingsforløb fremadrettet.

Ud over de manuelle tests, er unit-test for *nogle* af projektets use-cases implementeret.

5.1 Udførelse af test

Der er testet næsten konstant gennem udviklingen, f.eks. efter en ændring er foretaget i koden. Det har indtil nu udelukkende været manuelle test, hvor man fysisk sidder og tester en funktion efter eventuelle fejl og om funktionen giver det forventede resultat. Gennem en test følges der bl.a. nøje med i konsollen i Xcode, efter output der kan påvise en mulig fejl, eller uventet resultat.

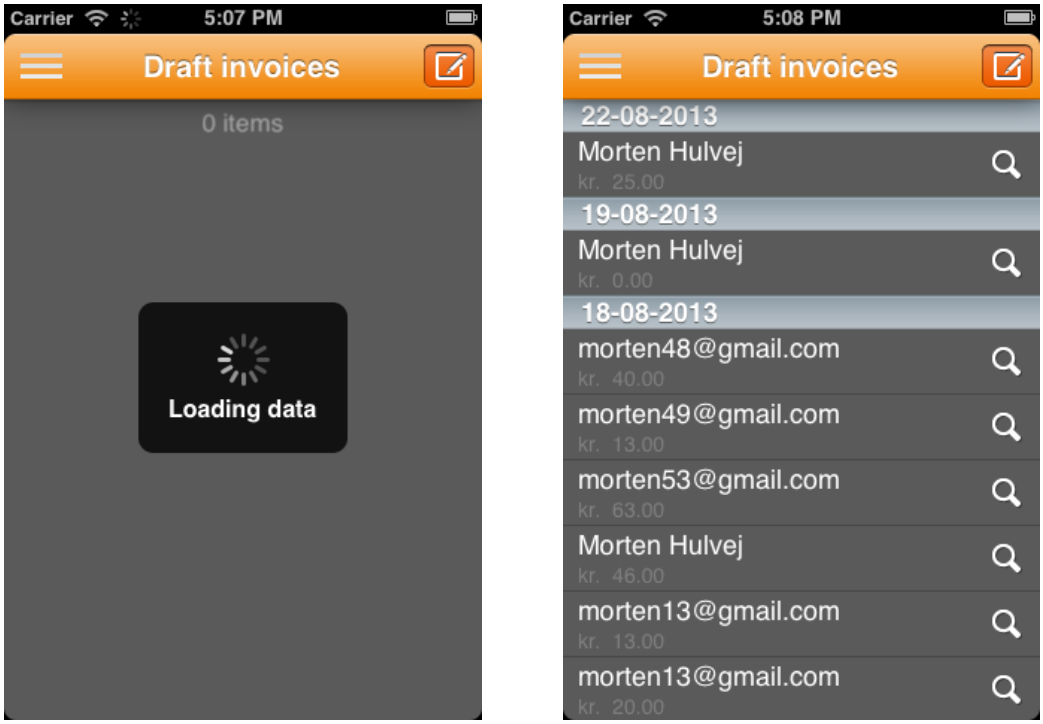
5.1.1 iOS iPhone Simulator

Helt fra iOS's start har Apple leveret en simulator sammen med Xcode. Denne simulator er uundværlig i forbindelse med udvikling af en iOS app. Simulatoren *simulerer* en nøjagtig opførsel, som kan sidestilles med en rigtig enhed. Denne simulator bruges konstant i projektet, da den giver et næsten perfekt billede af hvordan en app fungerer på en rigtig enhed.

Simulatoren har dog nogle begrænsninger. F.eks. kan den ikke bruge kameraet, hvilket kommer til at påvirke udviklingen på længere sigt, når der skal udvikles et modul til bilagshåndtering. Desuden har musen en begrænset gengivelse af *touch*: Når man skal rulle ned af en liste i simulatoren, skal listen ikke være meget lang før at man bruger meget tid på at rulle ned til bunden med en lang kombination af musetryk.

Til den basale navigation fungerer simulatoren dog udemærket og da simulatoren kører lokalt på udviklingsmaskinen, får man også mulighed for at teste og simulere forskellige tilstande på "enheden". Figur 21a viser simulatoren der er igang med at downloade en liste af fakturaer. Man kan her teste hvordan appen fungerer under dårlige netværksforhold, hvor det kan tage rigtig lang tid at modtage data. Med tredjeparts-programmer er der simuleret et fald i netværksbåndbredden, der svarer til en 2G data-forbindelse. På denne måde kan der fås et billede af om appen evt. skal give mulighed for at afbryde en igangværende handling, eller måske begrænse datamængden der skal downloades.

Specielt i dette tilfælde viste en test at brugere med rigtig mange fakturaer, ville opleve en uacceptabel lang ventetid. Derfor resulterede dette i en ændring, der kun indhentede de fakturaer som var redigeret inden for de sidste 14 dage. Ruller man ned til bunden af listen,



(a) Der hentes kladde fakturaer i simulatoren Denne handling kan tage lang tid, hvis der findes mange fakturaer i systemet.

(b) En færdig-hentet liste med kladde fakturaer.

Figure 21: iPhone Simulator

vil man nu automatisk hente fakturaer for de næste 14 dage, osv. På denne måde blev flere af de ikke-funktionelle krav opfyldt.

5.1.2 Test på iPhone

For at kunne teste en app på en rigtig enhed, kræver det at man er med i Apples *developer program*. I skrivende stund koster et års medlemskab $99,- dollars. Til projektet er der erhvervet et sådan medlemskab og derfor er også de rette certifikater sikret. Derfor kan appen testes på egen iPhone 5. I simulatoren får man fornemmelsen af, hvordan UI svarer på ens finger bevægelser, og man kan teste appens ydelse på et rigtigt mobilt netværk. Med iPhone tilsluttet computeren, kan der testes på præcis samme måde som i simulatoren. Debuggeren i Xcode kobler sig blot på processen på enheden i stedet for på simulatoren.

Det kan klart betale sig at køre ens test på en rigtig enhed, hvis der testes såsom batteriforbrug, hukommelseforbrug og andre hardware-relaterede test. Se mere i afsnit 5.6.

5.2 TestFlight

Der bruges TestFligt til at distribuere beta udgaver til testbrugere i projektet. TestFlight er en helt unik online platform der giver testbrugererne mulighed for at installere og opdatere appen direkte på deres smartphone, uanset hvor enheden befinder sig. Man kan overvåge deres færden i appen, og se om netop deres enhed giver problemer, hvis det f.eks. er en anden

Figure 22: iPhone 5 tilsluttet computer, mens Xcode kører en debug session direkte på enheden.

udgave end den der bruges i projektet.

TestFlight viser på denne måde en genvej igennem Apples noget kringlede distribution-proces.

5.3 TestFlight SDK

TestFlight tilbyder også et gratis SDK som kan benyttes i app'en, hvorpå stortset alt information omkring brug kan indsamles på en nem måde gennem en TestFlight konto. Dette SDK er implementeret i kodebasen, således at al logging der foretages til konsollen (standard-out) også bliver sendt direkte til en oversigt hos TestFlight. På denne oversigt bliver hver log sat sammen med brugeren der benytter appen, og derved kan man se præcis hvad en bruger foretager sig, og endnu vigtigere se hvis en bruger får en fejl.

Ud over fejllogging kan man også opstille bestemte *checkpoints*, som man kan brgue til f.eks. at registrere hvis en bruger gør brug af en bestemt funktion, og hvornår. Dette kan specielt være nyttigt ved videre udvikling af app'en til evt. drift. TestFlight SDK giver også mulighed for direkte feedback fra test-brugere, imens de bruger appen. Dette er dog ikke planlagt på nuværende tidspunkt.

5.4 Use-case test

Jvf. use-case diagrammet (se figur 32), kan der i appen laves flere test, der bekræfter de forskellige funktioners korrekthed. Som nævnt er der først skrevet tests i en relativ sen fase i udviklingen. Ikke destro mindre har det givet en fornuftig forståelse for det generelle test-miljø og muligheder i Xcode. I skrivende stund er kun en brøkdel af alle use-cases dækket med tests. Da test i iOS kræver særdeles skarp opdeling af ens kode, er det på nuværende tidspunkt kun muligt at teste use-cases på et *service* niveau, hvilket vil sige at samtlige servicekald til

API'et, vil blive testet. Skulle der testes funktionalitet i viewcontrollers ville dette kræve en refaktorering, da projektes tests ellers vil blive for afhængige af iOS, herunder bibliotektet `UIKit`.

5.4.1 Unit-test

Opbygningen af et test-miljø er påbegyndt, og indeholder unit-tests der dækker use-cases på et "service-niveau". Indtil videre er kun implementeret unit-tests for fakturering, men der er allerede forberedt tests for kunder og produkter. Alle tests er planlagt så de er selv-bærende og uafhængige, således at de ikke påvirker hinanden, hverken hvis en test fejler eller en anden ikke giver det forventede resultat.

Helt præcist er der lavet et *application-test target* i Xcode, der sørger for at køre alle unit-tests der er registreret i dette target. Da der her tales om service-tests, ville man nok oprette et *logic-test target* istedet, da dette ikke starter simulatoren og dermed app'en op samtidig. Dette kræver imidlertid at alle modeller og andre nødvendige filer, skal pakkes i et selvstædigt bibliotek. Fordi app'en ikke påvirker de eksisterende unit-tests ved opstart, ignoreres denne mulighed på nuværende tidspunkt.

```
1  ...
2  - (void)testCreateInvoice
3  {
4      __block EI_Invoice *invoice = [[EI_Invoice alloc] init];
5
6          // NB! expecting existing customer with id == 1
7          invoice.customer = [[EI_Customer alloc] init];
8          invoice.customer.number = 1;
9
10     // create the invoice
11         [service createSalesModel:invoice :^(id data, NSString *error) {
12                 if (error){
13                         STFail(error);
14                 }
15                 invoice = data;
16                 STAssertNotNil(invoice, @"data was nil");
17                 STAssertFalse(invoice.number == 0, @"invoice id was 0 when
                           returned");
18                 STAssertNotNil(invoice.customer, @"customer was nil");
19                 // if we end here the test was successful. End the test.
20                 ...
21         }];
22         ...
23 }
24 ...
```

Listing 5: Unit-test for UC:In1.

Listing 5 viser et eksempel på en unit-test. I dette tilfælde testes en oprettelse af en faktura-kladde (UC:In1). Initialiseringen af testen er ikke vist her, hvorend den blot opretter en instans af `ServiceWrapper` og gemmer den i variablen `service`. Hver test kører også tilslut en `tearDown` metode, der sørger for at rydde op efter hver test. F.eks. ville testen i listing 5 slette den netop oprettede faktura, samt lukke forbindelsen og serviceklassen ned. På denne måde får vi "selvbærende" tests, der ikke har brug for påklistret kode som krukke til sin kørsel.

5.5 Usability test

Enkelte test-brugere har været inde over projektet med kommentarer til appen. Èn test-bruger har afprøvet appen under opsyn, på egen enhed, med egen e-conomic konto. Her fandt man følgende udfordringer, som er interessante for de videre forløb:

En listning af fakturaer tog for lang tid, og brugeren blev hurtigt utålmodig, selvom en *loader* informerede brugeren om operationen. Især en hentning af allerede bogførte fakturaer tog rigtig lang tid, fordi forespørgslen ikke var afgrænset. Det er bl.a. på baggrund af disse tests, at *lazy-loading* blev prioriteret og tilføjet til projektets ikke-funktionelle krav.

5.6 App tests

Profilering og applikations test er også vigtigt, især platformen i dette projekt taget i betragtning. Derfor er der foretaget profillerings test mod en kørende app, med fokus på hukommelsesforbrug, netværksforbrug og batteriforbrug.

Til dette bruges *Instruments*, som er et selvstændigt udviklerværktøj fra Apple, men som også har en stærkt kobling til Xcode. Instruments kan foretage *rigtig* mange tests og målinger, hvorfor der kun er medtaget enkelte i denne rapport.

5.6.1 Hukommelsesbrug

Hukommelse er en altafgørende ressource på en iOS enhed, eller enhver anden smartphone for den sags skyld. Den er værdifuld, er begrænset, og skal deles med alle andre apps på enheden. I iOS rejses et flag, hvis en apps hukommelsesforbrug overstiger en bestemt grænseværdi, som er en flydende grænse bestemt af iOS selv i forhold til systemets aktuelle tilstand[26].

Hvis appen modtager en sådan notifikation, betyder det at systemet allerede har påbegyndt lukning af andre apps, som har holdt på hukommelse der kunne frigives. Dette er et forsøg på at holde den aktive app kørende så længe som muligt. Det er dog generelt inden for Apples design retningslinjer mht. hukommelse, at en app skal reagere prompte, på en sådan notifikation, og selv prøve at frigive hukommelse så vidt det er muligt. Det kunne f.eks. være referencer til billededata, eller andre store objekter der kan genskabes på et senere tidspunkt. I sidste ende lukkes appen uden varsel, hvis ikke hukommelsesforbruget er sænket til et tilpas niveau.

Ved de første testkørsler af e-conomic Mobile med *Intruments*, vises et udemærket hukommelsesbrug, der ikke overstiger nogen umiddelbare grænseværdier. På figur 23 ser man en faktura-listes hukommelses allokering målt over tid. Man kan her se de små "hakker" angiver hver gang man når til bunden af listen, hvor lazy-loading bliver aktiveret og downloader faktura data for næste periode. Netop fordi forbrugsniveau falder tilbage til udgangspunktet, giver dette et billede af ingen store *memory leaks*, hvilket er områder i hukommelsen som ikke bliver frigjort efter brug.

Endnu mere tydeligt bliver det på figur 24, hvor der hentes data til en kunde-liste. Fordi der på denne liste endnu ikke er implementeret lazy-loading som i forrige eksempel, kan der her ses et større og linært voksende hukommelsesforbrug, da hele listen med over 1000 kunder hentes på én gang. Dog er der heller ikke her grund til bekymring, da niveauet stabiliserer sig og hukommelsesområdet bliver frigivet kort tid efter.

[26] Se Apples Performance dokumentation

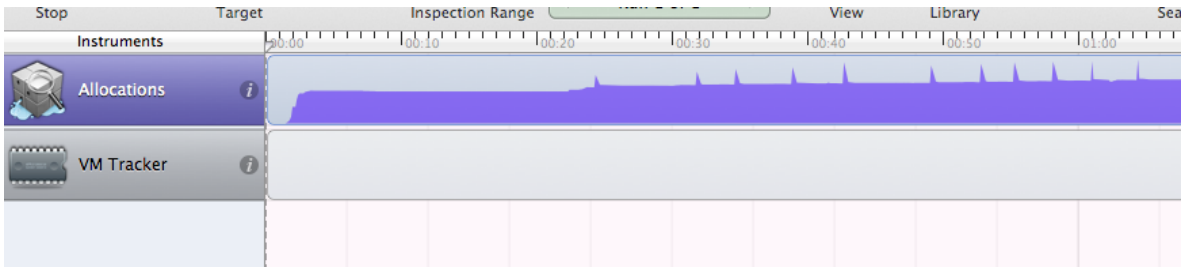


Figure 23: Oversigt over hukommelsesforbrug ved downloading af faktura-liste, for 14 dage adgangen.

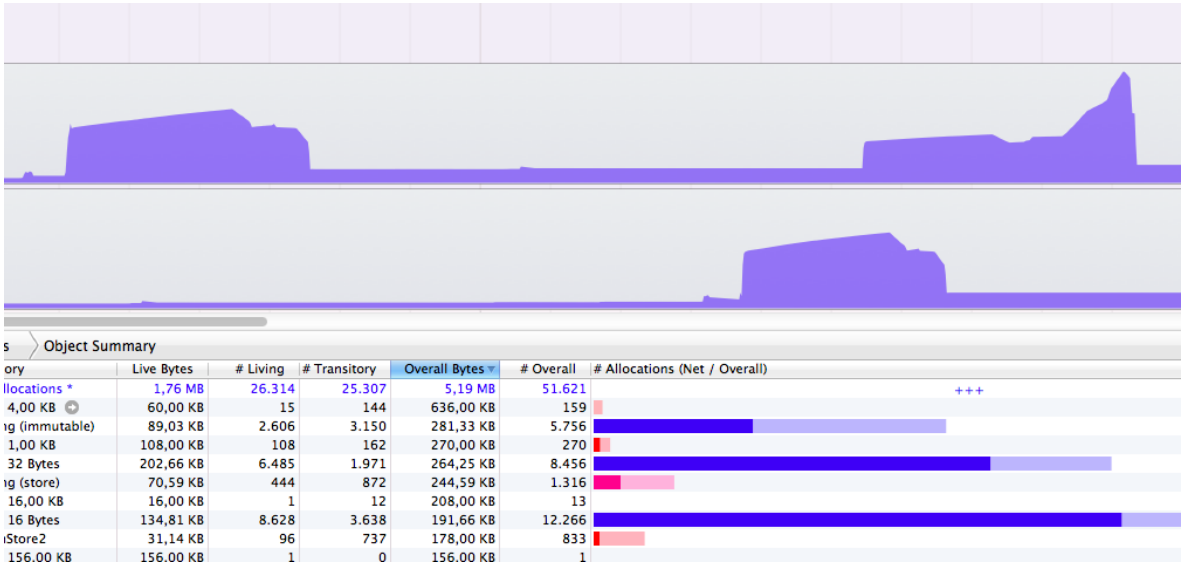


Figure 24: Samtlige kunder hentes på én gang hver gang kundelisten vises, hvilket koster dyrt i hukommelse.

5.6.2 Netværksforbrug

Når det drejer sig om netværksforbrug, er det overraskende nok ikke det store forbrug der registreres, når der udføres en såkaldt *Energy-Diagnostics test* med Instruments. Testen er udført med brug af SOAP API'et, som har en del overhead. Men på trods af dette rejser forbruget af netværkressourcer altså ingen til bekymring. Ved brug af REST API'et, vil forbruget sandsynligvis falde. Se figur 25

5.6.3 Batteriforbrug

Batteriforbrug er svart at teste, selvom Instruments dog også har værktøjer til dette. Det virker dog en smule tåbeligt, fordi for at man kan teste det reelle batteriforbrug, skal man for det første teste på en rigtig enhed, for at få et retsmæssigt billede af forbruget. For det andet skal enheden køre udelukkende på batteriet, og ikke være tilsluttet som ekstern strømkilde. For at teste med Instruments, *skal* enheden være tilsluttet computeren, og derved mister man de gunstige test forhold.

Figure 25: Lavt netværksforbrug gør app'en "billig" at bruge når wi-fi ikke er tilgængeligt.

5.6.4 UI Automation test

Automatiserede UI test er muligt i Instruments. Det fungerer ved at man ved hjælp af javascript, beskriver forskellige senarier hvorpå der testes UI i simulatoren, for effektivisere udviklingen. F.eks. kan man automatisere en bestemt handling såsom en visning af menuen, og derefter et klik på "List Customers". Ved at opbygge et test-senarie for hver use-case, kan man køre disse tests sideløbende med unit-tests og på denne måde gøre appen mere robust, da man hurtigt vil opdage evt. fejl ifm. med ændringer, enten i koden eller i UI. Automatiserede UI test er ikke prioriteret i dette projekt.

5.7 Delkonklusion

Dette kapitel har beskrevet test af forskellig karakter, der er kørt på app'ens implementation.

Der er i kapitlet fremlagt hvilke elementer der er testet. Bærende funktioner indenfor fakturerting er testet ved hjælp af unit-tests, herunder oprettelse af faktura, og downloading af samme. Selve testmiljøet er sat op i Xcode, men man havde flere udfordringer med dette, dels pga. manglende erfaringen med værktøjet og dels, pga. opbygningen af iOS.

Det er kun service funktioner der er testet/kan testes på nuværende tidspunkt. Dette tester dog en relativ stor mængde af implementationen, herunder begge API'er og object-mapping. Det konkluderes at dette er en tilfredsstillende dækning af koden i dette projekt, så vidt der ikke foretages forretningslogik i app'ens data-model. Det er dog også pointeret at et videre forløb med dette projekt, kræver en omstrukturering af koden, således at workflows også kan testes.

Resource-tests er foretaget og beskrevet i dette kapitel. Hhv. batteri-, netværks- og hukommelsesforbrug er blevet testet, ved hjælp af tillægs modulet Instruments. Især sidstnævnte test er værdifuld for udviklingen. Denne viste et stort hukommelseforbrug ved downloading af større datamængder, men viste samtidig at dette dog blev korrekt frigivet efter brug.

UI Automation tests ville kunne dække behovet for testing af UI, og dermed også til dels workflows i app'en. Der er dog ikke prioriteret denne form for test i dette projekt på nuværende tidspunkt. Et videre forløb ville klart kunne drage nytte af sådanne tests.

Usability tests har påvist vigtigheden af brugen af *infinite-list* mønsteret, hvor der ved en blev hentet *alle* fakturaer, der nogensinde var oprettet i e-conomic.

Alle typer af tests er blevet kørt på simulatoren og den fysiske enhed. Fordi REST API'et ikke er i en stabil nok tilstand, ville der ikke kunne dannes et fornuftigt grundlag at teste udfra, hvorfor de beskrevne tests kun er udført på SOAP API'et.

TestFlight og TestFlight SDK har vist at være et godt værktøj til fejllogning og registrering i app'en. Selv i et videre forløb vil man kunne drage stor nytte af TestFlight's *remote logging*, og kan give usability-testing en hjælpende hånd, med at samle, organisere og præsentere test-brugers færden i app'en.

Dette afsnit svarer samlet på spørgsmål 17 i problemformuleringen.

6 Konklusion

I dette sidste kapitel vil der blevet redegjort for alle dele af denne rapport. En opsummering af delkonklusionerne for hvert afsnit, samt en samlet konklusion for projektet vil kunne findes her.

Det eventuelle videre forløb for projektet, vil også blive beskrevet.

6.1 Opsummering af delkonklusioner

Der er blevet vendt mange spørgsmål, i forbindelse med dette projekt. I analysefasen er spørgsmål omkring hvilke API'er der skal bruges og implementeres blevet besvaret. Her blev der fundet at implementere alle API'er som e-conomic udbyder, herunder SOAP- og REST-API'et. Der er også fundet potentielle teknologier, der kan bruges til implementeringen af samme. Skønt REST API'et ikke er klar til produktion, er udviklingen så fremskreden at det selvfølgelig skal understøttes i app'en. Til det eksisterende SOAP API, bruges der en kode-genereret kodebase, som med minimal refaktorering bruges som hoved-API, og REST indtil videre kun på et udviklings niveau. REST API'et implementeres ved hjælp af biblioteket RESTkit.

Udviklingsmetoder er blevet endeligt fastlagt, hvorpå der bruges MVC-mønster, primært fordi iOS alligevel dikterer dette fra start. Der er også lagt et fokus på at tilgodese GRASP-principperne, som meget det er muligt. Der er sat nogle rammer for projektet, hvorpå der fokuseres på kernefunktioner i e-conomic. Der bliver i prototypen lagt ud med et fokus på fakturering, herunder oprettelse, redigering, sletning og bogføring af en faktura. Kunder og produkter tilhørende en faktura, vil i denne omgang skulle oprettes i det traditionelle e-conomic, før disse kan bruges i app'en. Der er også lagt fokus på hvilke brugere, der kunne bruge app'en. Hertil er der fundet flere e-conomic kundetyper, men dog har fokus været på slutkunder. Til brug for den videre udvikling, er udfordringer med den eksisterende app blevet fastlagt og undersøgt. Generelt er de største hastighed, og brugeroplevelsen gennem et ydelsesmæssigt dårligt UI. På baggrund af disse udfordringer er forskellige løsningsmodeller blevet fremlagt, bl.a. simple ting et ikke-blokerende UI og en bedre navigationsform.

Designfasen inkluderer en plan for hvordan API'erne skal implementeres, herunder især hvordan systemet gøres uafhængigt af, at der findes flere API'er. En afkobling er blevet beskrevet, hvorpå hele datalaget tolkes som ét API af resten af app'en. Meget vigtigt er også forskellige UI mønstre blevet undersøgt, og udvalgt. Der er lagt fokus på et navigations dygtigt UI, hvorpå der bruges en special designet version af iOS egen navigation-UI, med den velkendte navigationbar. Et design af service-implementationen hvorpå de to API'er bruges, er fremlagt sammen med en løsning på hvordan datatransformation vil fungere. Til og fra domænets datamodel, skal der transformeres meget data frem og tilbage. Mens der følger en veludbygget object-mapper sammen med RESTkit, er der designet en mapper specielt til den kode-genererede SOAP service.

I implementeringsfasen er fremlagt præcist hvordan de to API'er er implementeret, herunder også hvordan data *mappes*, frem og tilbage. Der er også beskrevet forskellige UI løsninger, bl.a. til de store datamængder brugt ved lister. Et infinite-list mønster er blevet brugt til at overkomme lange ventetider, ved forespørgsler på meget data på én gang. Desuden er adgangsmodellen fremlagt, samt hvordan den giver adgang til en brugers data hos e-conomics

servere.

Testfasen består af application-tests, unit-tests og enkelte usability-tests. Der lægges meget vægt på hukommelses-test der viser et jævnt forbrug af iPhones hukommelse. Men ved de lange lister, hvor der ikke endnu er benyttet et infinite-list mønster, ses et hurtigt stigende forbrug proportionalt med listens størrelse. Der bliver dog frigivet hukommelse korrekt, således der ikke registreres nogle umiddelbare *memory leaks*. Usability-testing har påvist vigtigheden af brugen af infinite-lists.

6.2 Projekt konklusion

Ser man på problemformuleringen, er alle spørgsmål i forbindelse med projektet besvaret, og resultatet er et proof-of-concept der *som minimum* giver e-conomic et bedre alternativ, til deres nuværende app. De vigtigste parameter så som ikke-blokerende UI, lange ventetider og et *native* miljø, er implementeret med success.

App'en er i en tilstand, hvor der kan dannes et fornuftigt grundlag for en videreudvikling og idriftsættelse af en færdig erstatningsapp. Projektet har dog været præget af denne udviklers erfaring med platformen, hvorfor også mere udbyggende funktioner ikke er prioriteret.

Der er heldigvis tilegnet stor indsigt i iOS-udvikling såvel som app-udvikling i al almindelighed. Det er især gjort tydeligt hvor meget UI og den grafiske linje, i langt højere grad spiller en rolle i denne form for udvikling, end traditionel softwareudvikling til desktops. Desværre er den grafiske linje endnu ikke på linje med den eksisterende. Hvis det ønskes kunne der hurtigt sættes grafiske ressourcer til rådighed, der ville kunne implementere den tilsvarende grafik brugt i e-conomics web-applikation, og derfor ses dette dog som et mindre problem.

Det har været rigtig værdi fuldt at have adgang til det kommende REST API. Dette kan sikre app'ens ultimative overlegenhed, når REST API'et kommer i produktion. e-conomic vil kunne tilbyde deres kunder en app der benytter deres nye API, fra første dag og samtidig ville de hurtigere kunne udfase SOAP API'et. Dette kan potentielt spare e-conomic for vedligeholdelsetimer i det gamle API og sikre deres kunder den nyeste teknologi.

App'en er selvfølgelig ikke klar til en større test-kørsel blandt slutkunder, dels fordi det er et PoC, men også fordi der stadig mangler ting der merkant adskiller denne app, fra den eksisterende. Godt nok er denne app, mere tidsvarende, veldesignet og ydelsesmæssigt overlegen, men dette er ikke nødvendigvis nok til at gøre kunderne glade. Der mangler flere funktioner, men er app'ens status er dog i en version 0.2, hvilket er planmæssigt tilfredsstillende for dette *dette* projekt.

6.3 Udtalelser fra e-conomic

[se udtagelse på næste side]

e-conomic

e-conomic international a/s
Wildersgade 10B, 3.
1408 København K

tlf. 88 20 4840

Til rette vedkommende

København, 9. September 2013

Udtagelse om diplombachelorprojekt e-conomic Mobile

e-conomics motivation for at få udviklet en ny app til vores hovedprodukt er flere. Dels har vi udfordringer med performance og stabilitet i vores nuværende mobile app. Pga. den nuværende implementering af appen, er vi desværre låst fast på teknologien og derfor ser vi med stor glæde på resultatet af Mortens arbejde i dette projekt.

Det er klart at Mortens PoC er den rette vej at gå når det kommer til den videre app understøttelse af vores produkt, især når det kommer til udnyttelse af funktioner til iPhone. Allervigtigst er at hastighed, og derved den generelle brugeroplevelse bliver forbedret.

Jeg kan sagtens se en videreudvikling af det PoC som Morten har lavet finde sted, med en efterfølgende release til vores kunder. Enten vha. en af vore egne udviklere eller af Morten selv, hvorfor han også er blevet tilbudt fastansættelse efter endt studie.

Med Venlig Hilsen

Jes T.S. Brinch
Director, Business Development

Figure 26: Udtagelse fra e-conomic.

6.4 Videre udvikling

Jvf. udviklingsplanen i afsnit 2.3 på side 22, vil det videre arbejde umiddelbart direkte kunne fortsættes. Man kunne her også udvikle flere funktioner til app'en. Udover elementer beskrevet i udviklingsplanen, kunne man tilmed udvide med flere *nice-to-have* funktioner såsom:

stregkode skanner - til oprettelse og søgning af produkter. Denne kunne også bruges som lagerregulering, ved en leverandør-levering.

Kunder fra kontakter - opret kunder fra kontakter.

Del faktura - med kontakter på enheden eller gem lokalt som billede på enheden.

Slide-menu søgebar - søgebar placeret i toppen af slide-menuen, som søger på tværs af entiteter. Således kan der søges på fakturaer, kunder, produkter, osv. via én søgebar.

Endvidere skal REST udvides, så alle nye muligheder bliver udnyttet så snart de bliver tilgængelige.

Hele test miljøet skal udbygges med unit-tests således at alle service-kald bliver dækket. Samtidig vil det være klogt at flytte ansvaret for API kommunikation over på modellerne, som noget af det første. Herefter bliver der større mulighed for at kunne teste controllers.

APPENDIX

A Bruger interviews

User research: David

Findings from interviews with e-conomic users

MWA • APRIL 02, 2013

MWA
APRIL 02, 2013

Primary tasks:

- Invoicing
- Booking expenses
- VAT reporting
- Forecasting income

MWA
APRIL 02, 2013

Bookkeeper

Uses a bookkeeper once a year to handle yearly tax (årsregnskab). They then sit together and add the right numbers from e-conomic into the right fields on SKATs self service website.

MWA
APRIL 02, 2013

"I get so tired of all that functionality and the advanced options that you never use"

MWA
APRIL 02, 2013

LARS HØJBERG, 36, CONSULTANT

Graphic Designer and web integrator in his own one man company

Uses e-conomic 3-5 times a week, since 2006

MWA
APRIL 02, 2013

Getting started

Thinks that there are way to many things that you dont use anyway. Would have made life easier for him if he didnt have to look at all the things that he never uses.

MWA
APRIL 02, 2013

Motivation

Is happy that he can do (almost) everything by himself in e-conomic so that he doesnt spend money on bookkeepers

It is important to him that he can avoid having to use other applications. e-conomic covers most of his needs

Figure 27: Bruger interviews, kilde: e-conomic UX Team

MWA
APRIL 02, 2013

Major challenges

Finds it very difficult to get an easy overview of his prospect income. He uses a calculator to add up the numbers on his list of 'current invoices' and 'offers'.These numbers are very important to him. Uses his current invoices for time registration as he charges by the hour. Would like to have 'real' time registration in e-conomic
He often invoices norwegian customers, but get paid in Danish kroner. To handle this he needs to do a workaround
Would make life easier for him if he could be notified of things he has to remember in relation to e.g. a customer - now he uses notepad on the side and must remember to check it every time he invoices someone
Would like to use 'reports' more, but find them too difficult to overview - they are made for bookkeepers...
Does not understand the bookkeeping terms like 'Vend fortegn' - why is it all in negative by default? That doesnt make sense to an ordinary user like me

---

MWA
APRIL 02, 2013

Never uses support, finds solutions to problems himself. Has phoned support once in 7 years.

MWA
APRIL 02, 2013

Handles his expenses by himself and books them as they come in (no bulk handling)

MWA
APRIL 02, 2013

Figure 28: Bruger interviews, kilde: e-conomic UX Team

MWA
APRIL 02, 2013

Motivation

Finds e-conomic a necessary and efficient tool.

Does not find accounting interesting, but it has to be done. He will rather spend his time selling and talking to his customers

MWA
APRIL 02, 2013

Getting Started

An accountant set up the application for him. The rest he fairly quickly figured out by himself.

MWA
APRIL 02, 2013

Major challenges

Is very dependent on the stability of the application as he often creates offers and orders together with the customer. Feels that it has been too unstable for a while now.
Finds it very annoying that you cannot see a saved product description on the final offer/invoice.

For collaborating with employees there is not a lot of options. He would find it very useful if you could somehow mark or flag things, so that others can see the e.

Figure 29: Bruger interviews, kilde: e-conomic UX Team

status, notes or other useful information related to e.g. a customer

---

MWA
APRIL 02, 2013

Collaboration

There are several employees in his company who are also using e-conomic.

MWA
APRIL 02, 2013

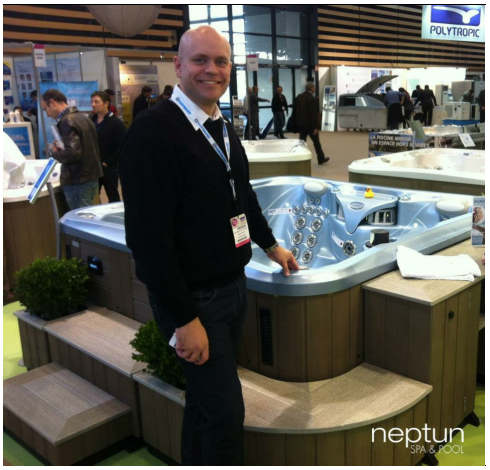
Bookkeeper

Has a bookkeeper employed who receives a bunch of 'bilag' from him every other week

She handles all his expenses and 'bilag', sometimes from home, other times she works at the office

---

MWA
APRIL 02, 2013



MWA
APRIL 02, 2013

Never uses support, has used Google a few times when in doubt

---

MWA
APRIL 02, 2013

Primary tasks:

- Create a lot of offers and orders

- Invoicing

- Compare budget

- Checking goods availability

- Checking incoming payments before delivering goods

Figure 30: Bruger interviews, kilde: e-conomic UX Team

MWA
APRIL 02, 2013

RASMUS, 33, DIRECTOR OF SPA SHOP

Owns a physical spa shop selling goods over the counter

Uses e-conomic on a daily basis, since 2009

MWA
APRIL 02, 2013

MWA
APRIL 02, 2013

Primary tasks:

- Invoicing
- Overview of his accounts
- Sending reminders

MWA
APRIL 02, 2013

Bookkeeper

Has a bookkeeper employed several days a week

She handles all his expenses and 'bilag'

MWA
APRIL 02, 2013

Major challenges

Selling service deals means that a deal usually runs over a long period of time. He cannot create a 'running abonnement' (which would save him a lot of time), but

Figure 31: Bruger interviews, kilde: e-conomic UX Team

has to manually create all invoices per month.

Besides he has to remember to send them each month.

MWA
APRIL 02, 2013

Getting started

An accountant set up the application for him. The rest he fairly quickly figured out by himself.

MWA
APRIL 02, 2013

Motivation

Finds e-conomic and easy and efficient tool.

MWA
APRIL 02, 2013

Collaboration

There are several employees in his company who are also using e-conomic

---

MWA
APRIL 02, 2013

Never uses support, has used Google a few times when in doubt

---

MWA
APRIL 02, 2013

"e-conomic is easy and manageable, it is just a thing you use. Generally I am a happy customer"

MWA
APRIL 02, 2013

PHILIP, 31, DIRECTOR OF IT BUSINESS

Owns an IT business mainly selling service deals

Uses e-conomic on a daily basis, since 2009

---

http://padlet.com/wall/userresearch

B Jes Brinchs udtalelser

Interview med Jes Brinch, Director for Business Development, hvor han svarer på følgende spørgsmål:

Hvad er de største udfordringer ved vores nuværende mobil-app (til iOS)?

"De største udfordringer med vores nuværende iPhone app er hastighed, responsivness og stabilitet. Dette skyldes til dels valget af HTML5 samt e-conomic's API."

Hvordan er brugernes reaktion på app'en?

"Vi har desværre oplevet dårlige ratings af app'en. Dog mener jeg ikke, at disse giver et retmæssigt billede af e-conomic app'en, da vi hvert måned har mellem 5000 – 10.000 unikke kunder på app'en og ca. 80% er tilbagevendende kunder."

Hvordan udnyttes den mobile platforms muligheder i app'en?

"Vi har brugt den mobile brugergrænseflade til at begrænse os til at blot favne de mest anvendte funktioner i e-conomic, samt en optimering af workflows. Foruden dette gør vi brug af telefonens kamerafunktion, til at indscanne bilag til sit regnskab."

Nævn de tre områder der skulle lægges allermest vægt på, hvis der skulle laves en ny udgave af appen - udviklet native til iOS?

"Bedre responsiveness, brug af flere native funktioner så som 'Del via Email' og GPS og generelt højere hastighed."

C Use-cases

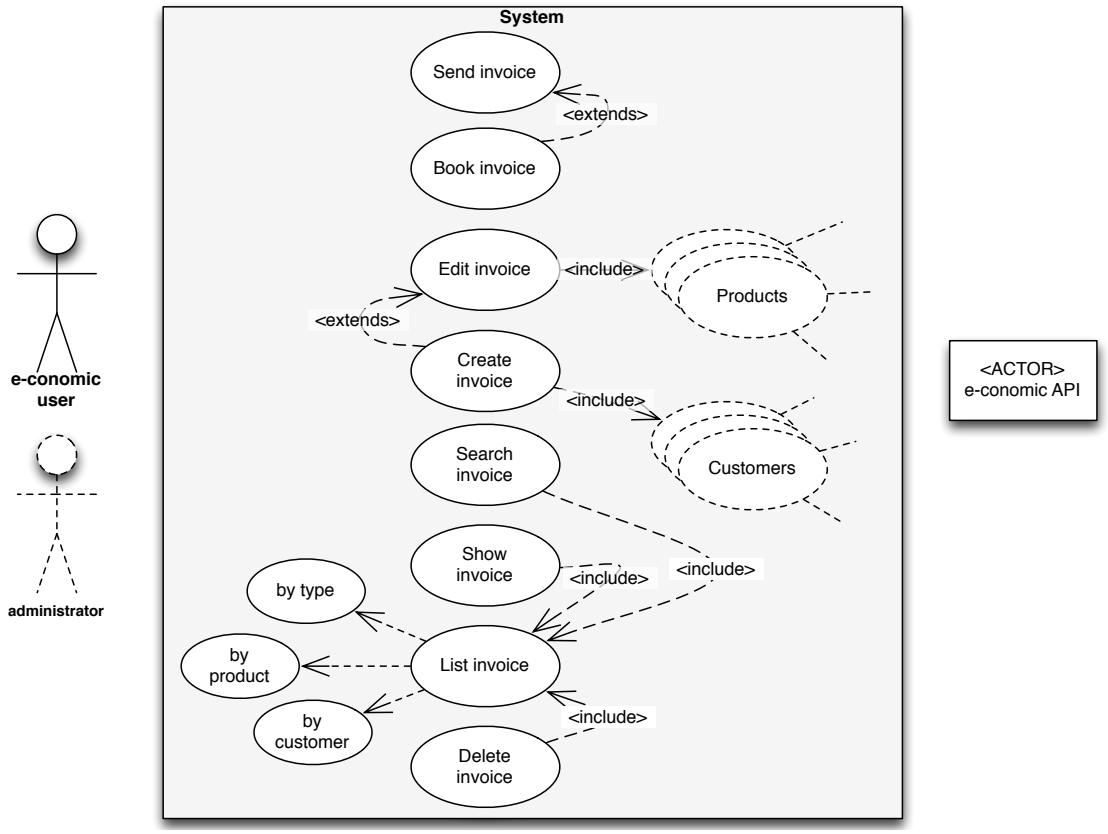


Figure 32: Fuldt use-case diagram over fakturerings funktionener.

UC: In1	
Use Case Title	Create invoice
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to create an invoice with only the absolute required attributes.
Main Success Scenario	1. user have a ongoing sale with a customer. 2. user creates new invoice. 3. user performs List customers. 4. user selects the customer from customer-list. 4. system signals that invoice is saved 5. user is now performing Edit invoice
Extensions	
Preconditions	The customer must exist in the e-conomic system. (prior to v0.3)
Success Guarantee	Invoice is created in users e-conomic account User can perform Show invoice and see output of invoice,

UC: In2	
Use Case Title	Edit invoice
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to edit a invoice, adding or removing content.
Main Success Scenario	1. user finds a particular invoice. a. user performs List invoice or Search invoice 2. user edits the data of the invoice. 3. user "exits" the invoice. 4. system detects the edit is finished. 5. system validates the new data. 6. system saves the changes in e-conomic system.
Extensions	2a. invoice is booked 1. system shows the invoice as read-only. 5a. system validation of data failed. 1. system notifies user about invalid data and cancels the save.

Figure 33: UC: Opret og redigér faktura.

	UC: In3
Use Case Title	Book invoice
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to book a draft invoice, so it is registered in the economic system, and it is included in all calculations of th ee-conomic system
Main Success Scenario	1. user finds a particular invoice. a. user performs List invoice or Search invoice 2. user books the invoice 3. system verifies that invoice can be booked. 4. system signals that the invoice has been booked.
Extensions	3a. system signals that invoice already is booked 1. booking canceled. 4a. user also want to send invoice. 1. user performs Send invoice
Preconditions	Invoice must have invoice lines
Success Guarantee	Invoice is booked in economic system. Invoice is now read-only

	UC: In4
Use Case Title	Show invoice
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to see the invoice in its full form, as the customer will see it.
Main Success Scenario	1. user finds a particular invoice. a. user performs List invoice or Search invoice 2. user asks the system for a visual presentation of invoice. 3. system fetches the invoice's visual presentation from e-conomic system and presents it.
Extensions	

Figure 34: UC: Bogfør og vis faktura.

	UC: In5
Use Case Title	Send invoice
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to send an booked invoice
Main Success Scenario	1. user finds a particular invoice a. user has just booked the invoice. b. user performs List invoice or Search invoice 2. user sends the invoice via email a. sends to the customer email b. sends to email from device contacts c. sends to a custom email
Extensions	
Preconditions	Invoice must be booked.

	UC: In6
Use Case Title	List invoice
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to get a list of invoices
Main Success Scenario	1. user signals to list invoices a. by type 2. system fetches all invoice by given type. 3. system shows all invoices on a condensed list
Extensions	

	UC: In7
Use Case Title	Search invoice
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to search among invoices, either by number, customer, product or date.
Main Success Scenario	1. user selects search criteria 2. user inputs search parameter 2. system fetches all invoice matching search criteria and parameter. 3. system shows the search result on a condensed list
Extensions	2a. search parameter is invalid with the chosen parameter. 1. system signals the user about wrong parameter.

Figure 35: UC: Send, list og søg fakturaer.

	UC: Cu1
Use Case Title	List customer
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to get a list of customers
Main Success Scenario	1. user signals to list customers. 2. system fetches all customers. 3. system shows all customers on a condensed list.
Extensions	

	UC: Cu2
Use Case Title	Show customer
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to see detailed info of a particular customer
Main Success Scenario	1. user finds a particular customer a. user performs List customer. 2. system is fetching all info on customer. 3. system shows all info of customer in detail.
Extensions	

Figure 36: UC: List og vis kunder.

	UC: Pr1
Use Case Title	List product
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to get a list of products
Main Success Scenario	1. user signals to list products. 2. system fetches all products. 3. system shows all products on a condensed list.
Extensions	

	UC: Pr2
Use Case Title	Show product
Scope	System use case
Level	User-goal level
Primary Actor	economic user
Stakeholders and interests list	user - wants to see detailed info of a particular product
Main Success Scenario	1. user finds a particular product a. user performs List product. 2. system is fetching all info on product. 3. system shows all info of product in detail.
Extensions	

Figure 37: UC: List og vis produkter.

D Tidlige designforslag

[*Se mockups på de næste sider*]

Figure 38: EPIC user-stories

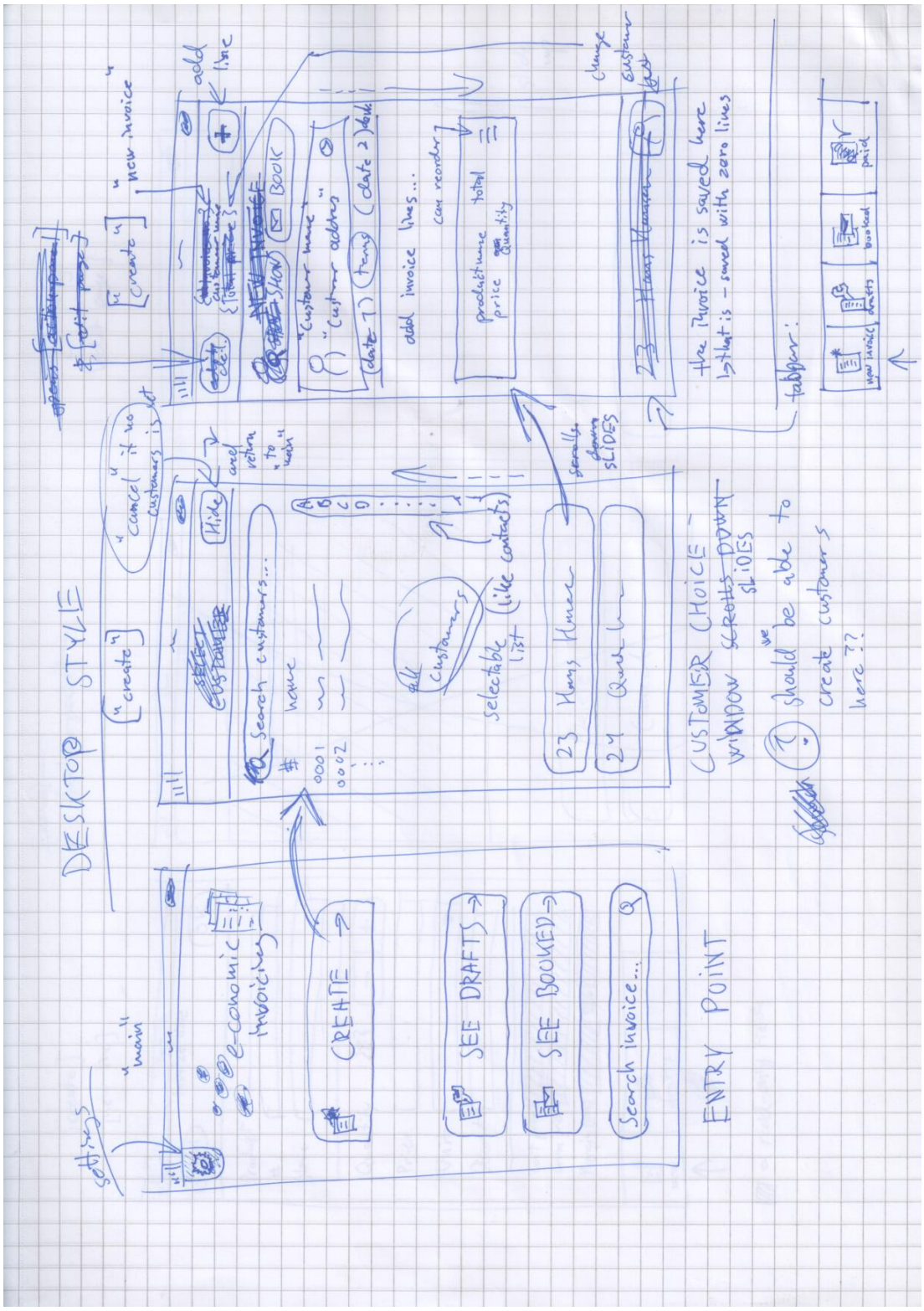


Figure 39: Desktop styled design

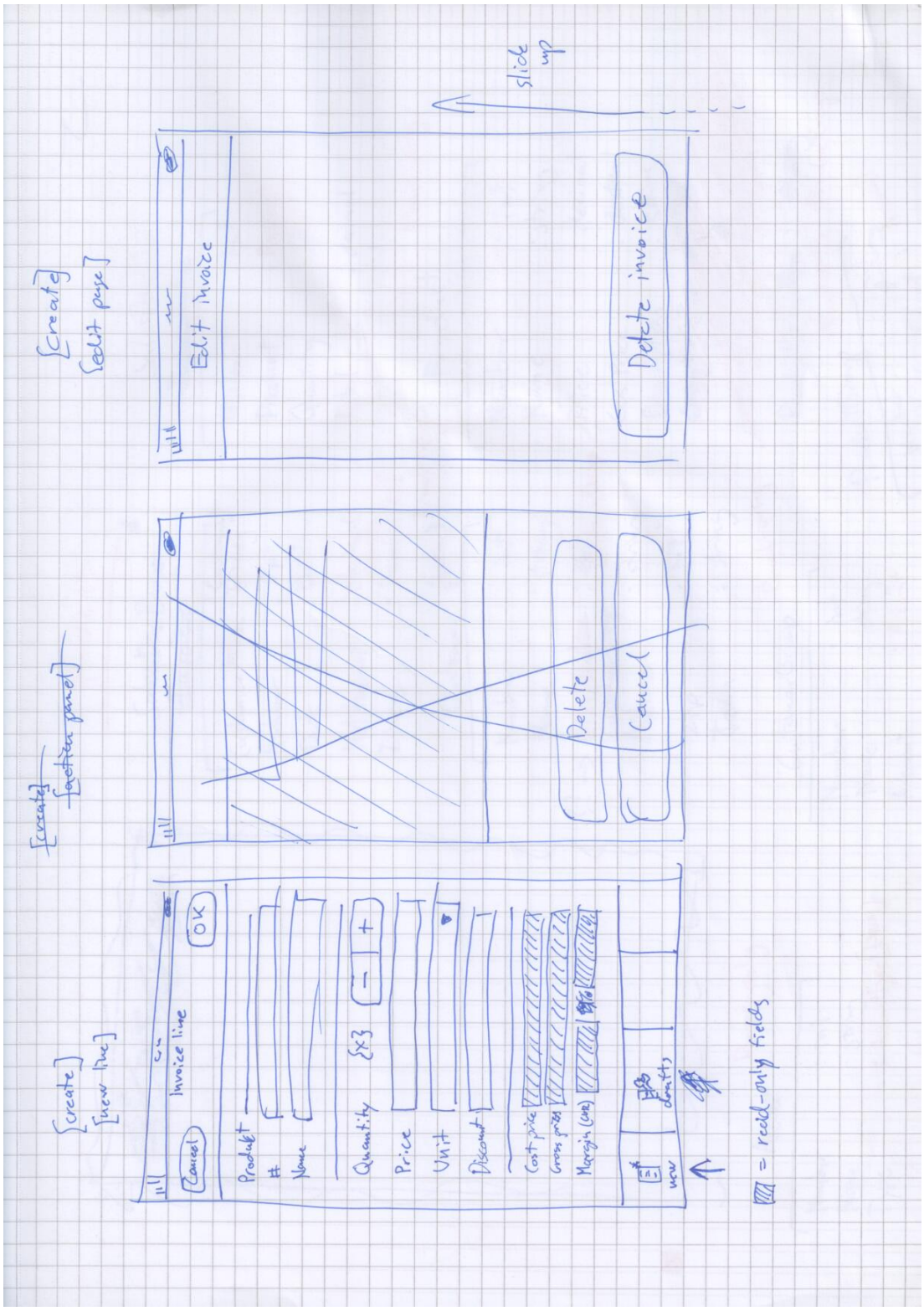


Figure 40: Form like creation view

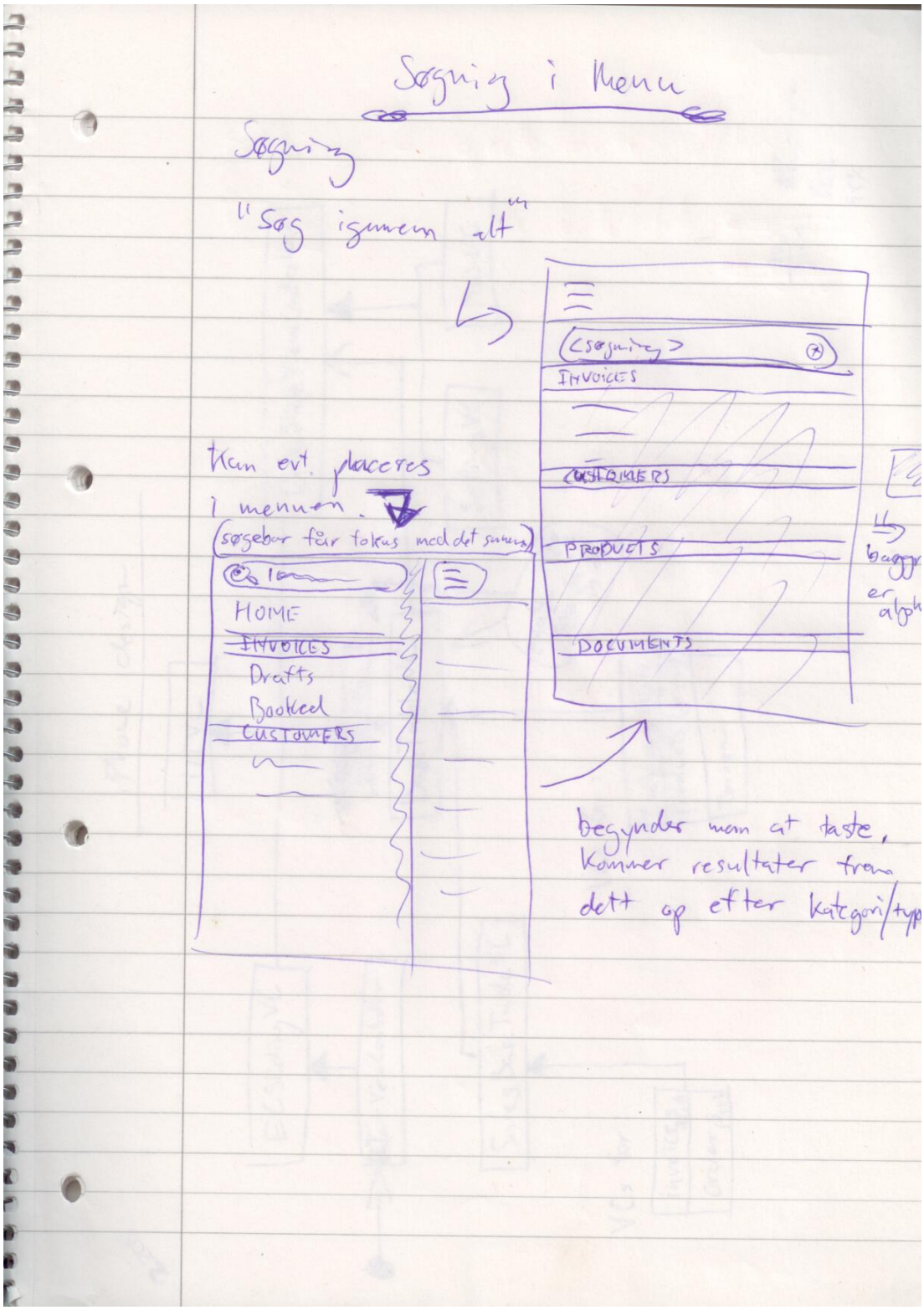


Figure 41: Searching in slide menu

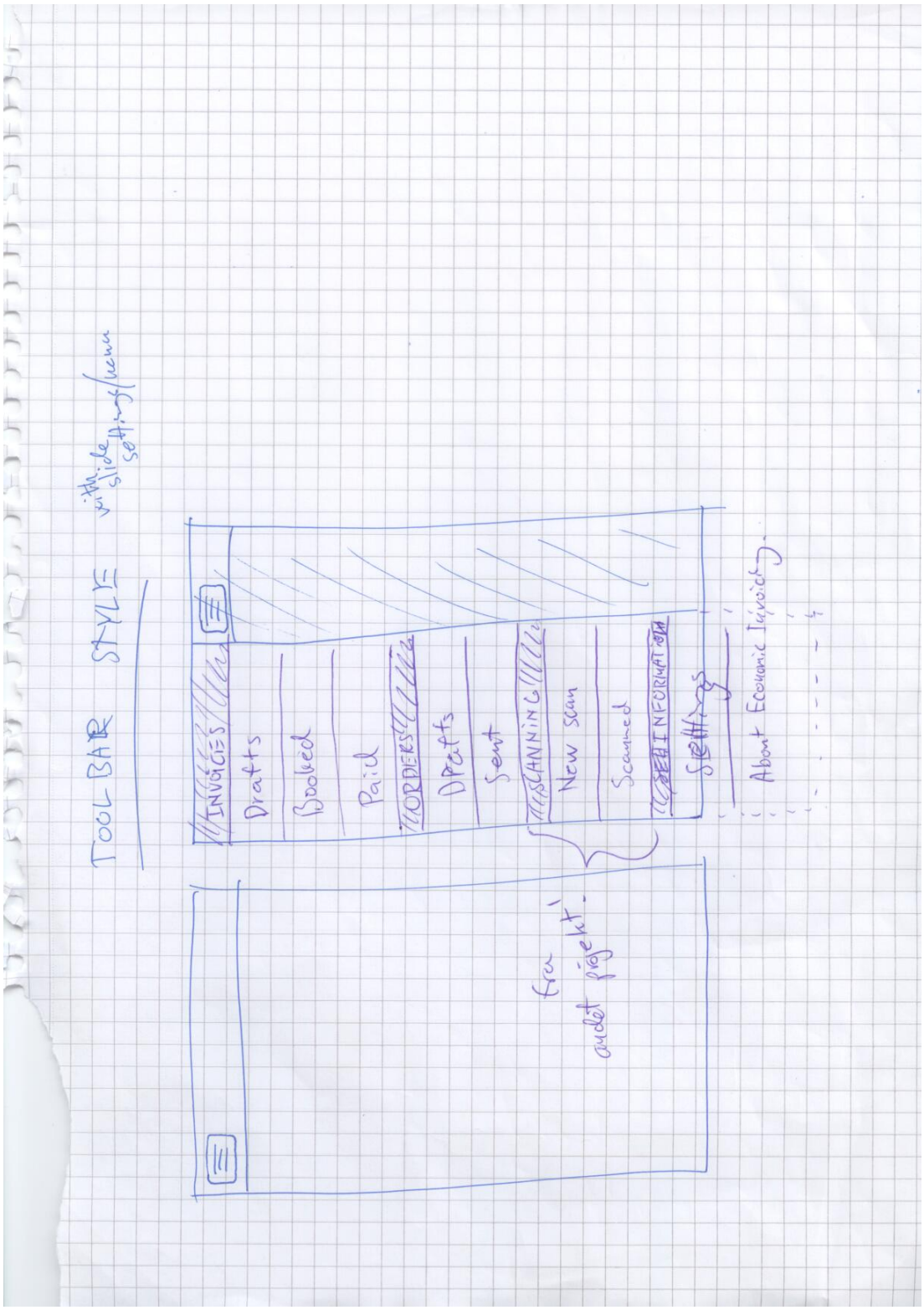


Figure 42: Toolbar design

Referencer & Litteratur

Books:

Larman, Craig - *Applying UML and patterns*, Prentice Hall, 2005

Tidwell, Jenifer - *Designing Interfaces*, O'Reilly, 2010

Web resources:

Wikipedia - `http://www.wikipedia.org/`

e-conomic - `http://www.e-conomic.net/`

GitHub - `http://www.github.com/`

Apple online dokumentation

Overall iOS documentation - `https://developer.apple.com/library/ios/`

Performance Tuning - `https://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/PerformanceTuning/PerformanceTuning.html`