Leads Companion

En Androidapp för säljarna på Telavox

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Institutionen för Elektro- och Informationsteknik

Examensarbete:
Johan Petersson

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2014

Sammanfattning

Den här rapporten beskriver ett examensarbete gjort som sista moment på högskoleingenjörsprogrammet Datateknik vid Lunds Tekniska Högskola, Campus Helsingborg. Examensarbetet genomfördes som ett utvecklings-projekt på företaget Telavox i Malmö som är verksamt inom IP-Telefoni.
Uppdraget var att utveckla en Android-app som skall användas internt av säljarna på företaget. Appen är en implementation av ett redan befintligt webgränssnitt som är en del av Telavox administrativa system. I projektet undersöktes även hur element av gameifiering kan implementeras i appen.

Då de tilltänkta användarna av appen ofta befinner sig i områden med endast tillgång till 2G-nätet har metoder för att minska negativ användarupplevelse på grund av långa laddningstider undersökts. Rapporten visar att genom att använda metoden *Lazy Load* kan mängden data som kommuniceras mellan server och klient, och således även laddningstider minskas. Slutsatsen är dock att om metoden skall kunna användas krävs en omfördelning av arbetet mellan server och klient. I den utvecklade appen har istället en cache-fil använts. Detta för att på så sätt minska upplevda laddningstider för användaren.

Rapporten visar också hur det är möjligt att med hjälp av ramverket *Fragments* utveckla en Android-app som fungerar likvärdigt på flera skärmstorlekar utan att utvecklaren behöver skapa flera layoutfiler anpassade för specifika skärmstorlekar.

Nyckelord: JSON, Android, Lazy Load, Android fragmentation, Dashboard.

Abstract

This report describes a bachelor thesis project realized as the final exercise in the Computer Science program at Lunds Tekniska Högskola, Campus Helsingborg. The thesis work was done as an Android app-development project at Telavox in Malmö, a company active in the IP-Telephony business.
The purpose of the project was to develop an Android application to be used by the sellers at the company. The application is an implementation of an already existing web interface which is part of the company's administrative system. An investigation about how elements of game-ification can be implemented in the application was also part of the project.

The proposed users of the app are often located in areas with only 2G-mobile network coverage. Because of this methods for diminishing negative user experience due to long load times have been studied. The report shows that by using *Lazy Load* the amount of data communicated, and as a result load times, between server and client can be lowered. The conclusion is that to be able to use *Lazy Load* a restructuring of how the work is divided between server and client has to be made. In the developed app a different approach has been taken by the use of a cache file which hides load times from the user.

The report also shows how it is possible by the use of the framework *Fragments* from *Android API* to develop an Android app targeted for devices with a broad range of screen sizes without the need to create several layout files for specific devices.

Keywords: JSON, Android, Lazy Load, Android fragmentation, Dashboard.

Förord

Tack till Personer: Christin Lindholm, som hjälpt mig med handledning under framtagandet av den här rapporten. Christian Nyberg, min examinator på LTH. Henrik Thorvinger, min handledare på Telavox för att han gav mig möjligheten att utföra examensarbetet på företaget. Viktor Fogelberg för teknisk handledning samt alla andra på Telavox kontor i Malmö. Marlene Lövgren som varit ett stöd under arbetet med den här rapporten. Avin Botani som fick mig att påbörja den resa som mina studier inneburit vilken med det här examensarbetet nu är vid slutdestinationen. Min familj för att alltid ha stöttat mig. Börje Gustavsson och Joel Brindefalk, två personer som har betytt mycket för mig men som båda tyvärr hastigt gick bort under min studietid vid LTH, vila ifrid mina vänner.

Johan Petersson
Malmö September 2014

Innehållsförteckning

1 Inledning

Den här rapporten beskriver ett examensarbete genomfört som sista moment på högskoleingenjörsprogrammet Datateknik vid Lunds Tekniska Högskola. Examensarbetet genomfördes som ett utvecklingsprojekt av Johan Petersson under våren/sommaren 2014 och är utfört på uppdrag av företaget Telavox i Malmö.

1.1 Bakgrund

Telavox är ett företag verksamt som operatör inom IP-telefoni. Företaget har egenägd infrastruktur i form av geografiskt spridda växelenheter samt verkar som virtuell mobil operatör i Telias mobilnät. Bolaget har sedan starten år 2000 vuxit till att idag ha över 150.000 företagsanvändare [1].

I nuläget har Telavox ett *dashboard,* det vill säga grafisk presentation av data och information, i ett webbgränssnitt kallat ***Leads***. ***Leads*** är den del av företagets affärssystem som används av företagets säljare. Det finns önskemål om att göra en mobilapp som ett komplement till det befintliga webb-gränssnittet. Önskemål finns även om att utforska eventuella element av *gameifiering*, exempelvis att alla enheter notifieras då en säljare har uppnått ett visst säljmål, och hur dessa skulle kunna implementeras.

1.2 Syfte

Syftet med examensarbetet har varit att utveckla en prototyp/app för *Android*-plattformen baserat på Telavox redan befintliga webbgränsnitt ***Leads*** och att i samband med utvecklingsarbetet:

- Undersöka om det med enkla medel går att utveckla användar-gränssnitt som visas bra på flera skärmstorlekar. Detta utan att utvecklaren behöver skapa flera komplexa layoutfiler anpassade för specifika skärmstorlekar.

- Undersöka metoder för att komma runt de problem som låga över-föringshastigheter kan medföra då användaren befinner sig i områden med endast *2G*-uppkoppling.

1.3 Målsättning

Målsättningen för examensarbetet var att utveckla en prototyp/app baserat på vad som framkommit i den inledande undersökningsprocessen. Grundläggande krav för examensarbetet var att appen som minst skall kunna hantera:

- Login till Telavox server.

- *JSON* för kommunikation mellan *klient* och *server*.

- Grafisk presentation av *data* och *information*.

- Att användaren har tillgång till information även då denne befinner sig på en plats där endast *2G*-nät finns tillgängligt.

- Två enheter, en telefon och en tablet skall väljas ut som målenheter, appen skall kunna uppvisa likvärdig funktion på dessa två.

1.4 Problemformuleringar

Androidenheters begränsade skärmstorlekar kan vara en utmaning vid presentering av information och data. Detta och att många androidenheter har olika skärmstorlekar och upplösningar kan komplicera vid utveckling. Då säljare i fält ofta befinner sig där det endast finns *2G*-nät krävs att storleken på den data som skall hämtas är liten. Med grund i detta syftar den här rapporten till att försöka besvara följande två frågeställningar:

- Hur kan man säkerställa att en användare har tillgång till den information som den behöver, även då denne befinner sig i områden med täckning av endast 2G-nät? Finns det metoder som kan minska mängden data som behöver kommuniceras mellan server och klient?

- Hur kan man på ett enkelt sätt utveckla till flera androidenheter utan att använda kod specifik för varje enskild enhets skärmstorlek?

1.5 Avgränsningar

För att begränsa omfånget på examensarbetet fattades i samråd med handledaren på Telavox, Henrik Thorvinger, beslut om följande avgränsningar.

- Ingen säkerhet mer än inloggning kommer att implementeras.
- Appen skall fungera som ett komplement och inte en ersättare till det befintliga systemet.
- Utveckling kommer ske med *Android* 4.0 som minst kompatibla version. Detta för att inte behöva begränsa användningen av tillgängliga *klassbibliotek* från *Android API* vid utveckling av appen. Det kan även förutsättas att de tilltänkta användarna alla har enheter som kan köra applikationer med *Android* 4.0 eller högre.

1.6 Rapportens upplägg

Här följer några beskrivningar av vad rapportens olika delar handlar om vad som finns hittas i dem.

Metod
Här hittas information om under vilka former examensarbetet genomförts och vilka metoder som använts.

Analys
Analyskapitlet är uppdelat i fyra delkapitel:

Först kommer en korfattad beskrivning av det nuvarande systemet samt lite om hur säljorganisationen på företaget ser ut.

Det andra delkapitlet beskriver arbetet med att ta fram en grundläggande systemdesign och datamodell.

Sedan kommer ett kapitel som handlar om det undersökningsarbete som gjordes för att hitta innehåll och funktionalitet hos appen.

Analyskapitlet avslutas sedan med att beskriva hur arbetet med att ta fram ett koncept för appen förtlöpte och visar skisser på det framtagna konceptet.

Teknisk bakgrund
I kapitlet teknisk bakgrund ges information som rör Android-utveckling, specifika klasser och klass-bibliotek som har använts vid utvecklingen av appen samt viss teori för metoder och teknologier som har använts eller studerats under examensarbetet.

Implementering
Kapitlet implementering beskriver övergripande det som har utvecklats och innehåller beskrivningar av den resulterande systemdesignen och systemets viktigaste komponenter.

Resultat
I kapitlet resultat visas den utvecklade appen och beskrivs lösningar på de problem som examensarbetet skulle undersöka. Här beskrivs även sådant som var planerat att finnas i appen men som sedan inte blivit implementerat samt orsakerna till det.

Slutsats
I det här kapitlet hittas slutsatserna som är dragna av examensarbetet samt svar på frågeställningarna.

Framtida utvecklingsmöjligheter
Med bakgrund i erfarenheter gjorda i samband med examensarbetets genomförande ges här förslag på hur vidareutveckling av appen skulle kunna ske.

I texten är vissa ord skrivna *kursivt*, det innebär att om ordet inte finns beskrivet i texten så finns en beskrivning av ordet med i terminologin i avsnitt 9 sist i rapporten. Ord skrivna ***kursivt*** och i ***fetstil*** betyder att de antingen är komponenter i Telavox befintliga system eller i den utvecklade appen. Detta gäller genom hela rapporten men inte för överskrifter eller i stycken där allt är skrivet kursivt.

2 Metod

Det här kapitlet beskriver de metoder som har använts i examensarbetet och hur upplägg och arbetsformer har sett ut.

2.1 Upplägg

Arbetet genomfördes i två övergripande faser, analys och implementering.

2.1.1 Analysfasen

Analysfasen påbörjades med att visst utvecklingsarbete gjordes, detta för att ge kunskap om det befintliga systemet, dess komponenter samt kommunikationen mellan dem. I samband med detta genomfördes studier av hur det nuvarande systemet ser ut. En *fokusgrupp* sattes ihop och intervjuer genomfördes med dess medlemmar för att bestämma appens innehåll och funktionalitet. Som förberedelse för intervjuerna gjordes litteraturstudier för att ge kunskap om intervjumetodik och arbete med *fokusgrupper*. Fasen avslutades med att ett koncept för appen togs fram och presenterades för *fokusgruppen*.

2.1.2 Implementeringsfasen

I implementeringsfasen genomfördes parallellt med utvecklingsarbetet studier av olika metoder med koppling till examensarbetets fråge-ställningar. Informationsinsamlingen skedde i den här fasen som en del i utvecklingsarbetet. Även testning av funktionalitet gjordes som en del i utvecklingsarbetet.

2.2 Handledning

Företaget bistod med två handledare under examensarbetet.

Henrik Thorvinger, fungerade under arbetets gång som ett bollplank för ideer om appens utformning och hjälpte till med prioritering av appens innehåll och funktionalitet.

Viktor Fogelberg, den utvecklare på företaget som tagit fram webb-gränsnittet med tillhörande *API* för ***Leads***, utsågs till teknisk handledare. Han bistod bland annat med granskning av kod och fungerade som informationskälla under hela examensarbetet. Företaget har varannan måndag ett stående möte där alla utvecklare kortfattat berättar vad de gjort sedan senaste mötet, om de stött på några problem samt vad de planerar att göra. Examensarbetet togs även upp under dessa möten.

2.3 Projektmodell

Examensarbetet genomfördes utan en förutbestämd *projektmodell* då det är svårt att hitta en modell anpassad för en person. Istället delades arbetet upp i ett antal moment. För att få upp en övergripande styrning av examensarbetet gjordes en tidsplan med ett antal milstolpar och tiden som bedömdes behövas för varje del i examensarbetet estimerades. Under arbetets gång har tidsplanen uppdaterats då det visade sig att den estimerade tiden inte räckte till.

De olika momenten som examensarbetet bestod av var:

- ***Kartläggning och studier av befintligt system och säljorganisation.*** Det första steget i examensarbetet var att skapa förståelse för hur det befintliga systemet och säljorganisationen ser ut. Beskrivning av momentet hittas i kapitel 3.1.

- ***Utveckling av grundläggande funktionalitet och preliminär systemdesign***

För att ge underlag till fortsatta studier utvecklades grundläggande funktionalitet och en preliminär systemdesign togs fram. Detta finns beskrivet i kapitel 3.2.

- ***Arbete med fokusgrupp för att kartlägga processer och nuvarande arbetssätt***
  En fokusgrupp bestående av tilltänkta användare sattes ihop med vilken intervjuer genomfördes. Hur det arbetet gick till beskrivs i kapitel 3.3.

- ***Arbete med att bestämma innehåll och funktionalitet för appen***
  Med underlag i vad som framkom i arbetet med fokusgruppen bestämdes innehåll och funktionalitet för appen. Detta beskrivs i kapitel 3.4

- ***Framtagning av koncept för appen.***
  Ett koncept för appen togs fram och presenterades för fokusgruppen. Även detta finns beskrivet i kapitel 3.4

- ***Implementering och testning***
  Efter att ett koncept för appen blivit framtaget och presenterat för fokusgruppen påbörjades implementering av appen. Testning av det som utvecklades gjordes här som en integrerad del i utvecklingsarbetet. Arbetet planerades genom att den funktionalitet och de sidor i appen som bestämts i tidigare moment prioriterades tillsammans med handledaren på företaget. Implementeringen gjordes sedan en funktionalitet eller sida i taget enligt prioriteringsordningen.

- ***Rapportskrivning***
  Under examensarbetets gång har kontinuerlig dokumentation skett över vad som har gjorts, beslut som tagits och problem som

uppkommit. Denna dokumentation har sedan sammanställts och använts som underlag till den här rapporten.

2.4 Källkritik

Under examensarbetet har mycket information hämtats från Internet. Alla de webbsidor som använts är dock av typen instruktioner eller beskrivningar av *API* eller protokoll samt standards. Dessa kommer direkt från tillverkaren/utfärdaren och kan således bedömmas som trovärdiga. Den enda källa från Internet förutom de tidigare nämnda är en rapport som beskriver Android Fragmentation och är inget som bedöms kontroversiellt. All övrig information har hämtats från tryckta källor av meriterade författare utgivna på betrodda förlag och bedöms således som trovärdiga. Nedan följer en kort beskrivning av varje källa som använts, fullständiga källhänvisningar finns i kapitel 10.

2.4.1 Källor från internet

http://androidplot.com
Hemsidan för AndroidPlot, ett open source-API till för att skapa olika sorters diagram på Androidplattformen.

http://cometd.org
Hemsidan för CometD, en implementering av Bayeuxprotokollet vilket tillhandahålls av Dojo Foundation. Dojo Foundation är en ideell organisation verksam inom open source-utveckling.

http://developer.android.com
Den officiella sidan riktad till utvecklare för android-plattformen. Tillhandahålls som en del av Android Open Source Project.

https://www.eclipse.org
Eclipse Foundations hemsida. Eclipse Foundation står bakom utvecklingsmiljön Eclipse, en av de mest använda utvecklingsmiljöerna för Java.

http://www.ecma-international.org
Ecma International är en organisation som sedan 1961 jobbat med standardisering av informations- och kommunikations-teknologi samt konsumentelektronik.

https://github.com
Onlinetjänst som tillhandahåller repositories för versionshantering av kod.

http://git-scm.com
Hemsidan för versionshanteringssystemet GIT.

http://www.ietf.org/rfc.html
IETF, Internet Engineering Task Force är en internationell sammanslutning av experter. IETF har som mål att förbättra Internet genom att producera högkvalitativa dokument och sätta standards i form av RFC (Request for comment).

http://opensignal.com
Open Signal är en organisation som kartlägger och dokumenterar täckning av mobila- och Wi-Fi nätverk över hela jorden. Organisationen samlar in och analyserar data genom användare av organisationens app för Android och IOS.

http://www.oracle.com
Oracle är ett företag som exempelvis utvecklar och tillhandahåller MySQL, en av världens mest använda databashanterare.

http://www.telavox.com
Hemsidan till företaget Telavox där examensarbetet utfördes.

http://apache.org
The Apache Software Foundation är en ideell organisation som tillhanda-håller fritt tillgängliga mjukvaruprodukter. Organisationen står bakom Apache License, en Open Source licens som gör det enkelt för alla, företag

som enskilda personer, att använda Apaches produkter i sina utvecklingsprojekt.

http://www.w3schools.com
W3C eller World Wide Web consortium är en internationellt organisation som tillhandahåller standards för internet.

http://www.gzip.org
Hemsidan för gzip, ett komprimeringsverktyg som tillhandahålls via The GNU Project, http://www.gnu.org/gnu/gnu.html.

http:// www.json.org
Sida som beskriver JSON-formatet.

http://office.microsoft.com
Microsofts sida för produkter i Office-paketet.

2.4.2 Litteraturkällor

Intervjumetodik, andra upplagan
Annika Lantz är professor i psykologi, framförallt organisations- och arbets- psykologi vid Stockholms universitet.

Design Patterns, Elements of Reuseable Object-Oriented Software
Författarna till boken "Design Patterns, Elements of Reusable Object-oriented Software", Erich Gamma, Richard Helm, Ralph Johnson och John Vlissides brukar kallas för Gang of Four. Boken var den första som presenterade idén om designmönster för mjukvaruutveckling.

Patterns of Enterprise Application Architecture
Martin Fowler är en erkänd författare som hittils har skrivit sju böcker om mjukvaruutveckling. Fowler är även redaktör för en serie böcker utgivna på förlaget Addison-Wesley. Boken beskriver designmönster som används i affärssystemsarkitektur.

Software Requirements, Styles and Tehniques
Soren Lauesen, som skrivit "Software Requirements, Styles and Techniques" har lång erfarenhet av mjukvarubranchen. Han har en lång rad av publikationer och böcker bakom sig och suttit i flera danska vetenskapsråd. Lauesen har forskat inom områden som design av användargränssnitt, människa-dator interaktion, kravhantering, objekt-orienterad design, kvalitetssäkring, systemutveckling, marknadsförings- och produkt-utveckling samt samarbete mellan forskning och industri.

2.4.3 Intervjuer

All information som framkom vid intervjuerna speglar respondenternas åsikter. Informationen har sedan verifierats för att säkerställa att en så korrekt analys som möjligt kunnat göras av intervjumaterialet.

3 Analys

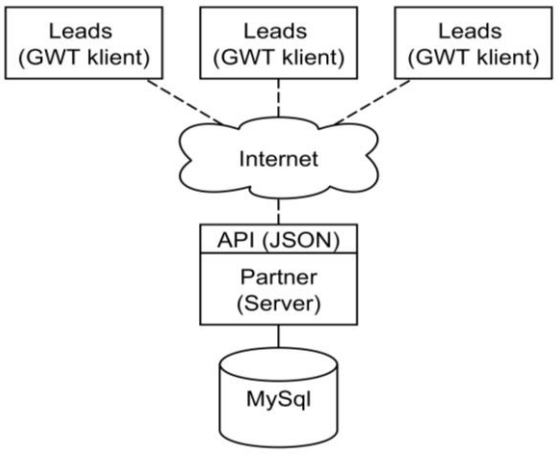
Det här kapitlet beskriver det inledande undersökningsarbetet med att hitta innehåll och funktionalitet i appen. Här beskrivs hur det befintliga systemet ser ut, hur intervjuer genomfördes och hur data från dessa behandlades. I slutet av kapitlet visas det app-koncept som togs fram och presenterades för de tilltänkta användarna innan implementeringsfasen påbörjades.

3.1 Befintligt system och säljorganistation

Innan utvecklingsarbetet kunde påbörjas behövdes en övergripande kunskap om hur det befintliga systemet ***Leads*** fungerade och om hur säljorganisationen ser ut. Informationen samlades in genom samtal med den tekniska handledaren och genomgångar av det befintliga systemet. Den användes sedan som underlag för arbetet med att hitta innehåll och funktionalitet i appen.

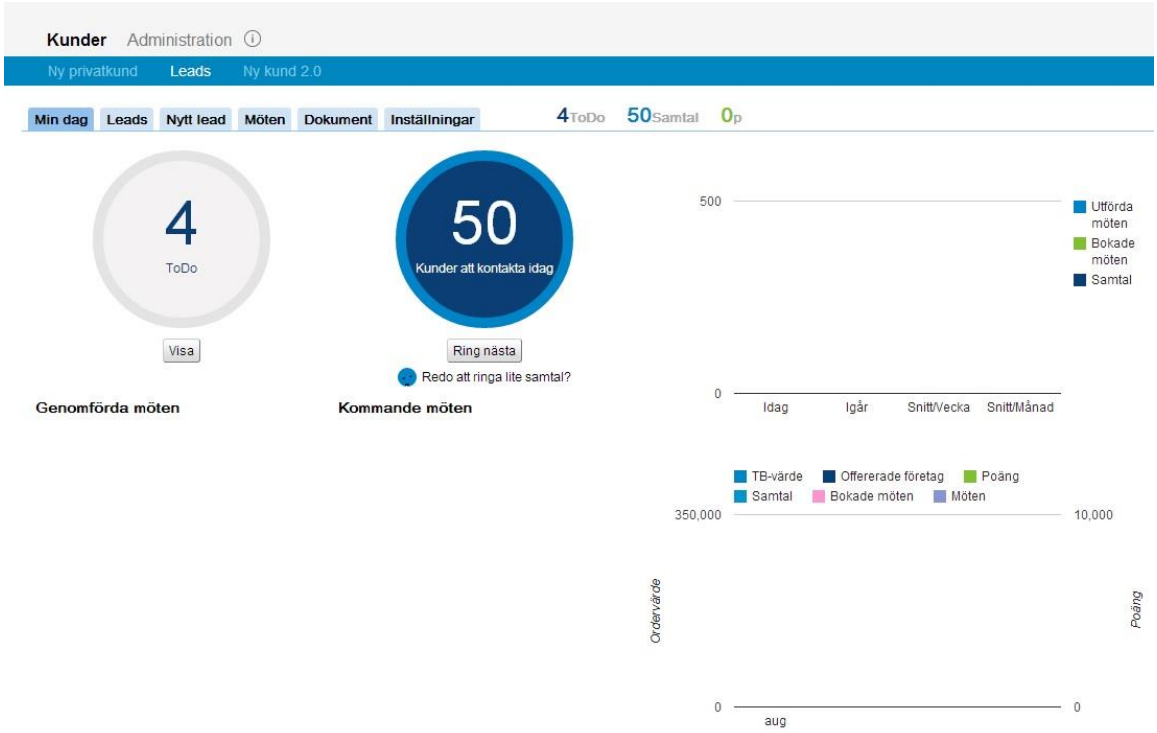
3.1.1 Leads

Leads är den del i Telavox affärssystem ***Partner*** där säljarna utför den dagliga administrativa delen av sitt arbete. ***Leads*** är en webbapplikation utvecklad med hjälp av *GWT* (Google Web Tools) och kommunicerar med ***Partner*** via ett *JSON-API, se* Figur 1.



Figur 1.
Leads består av en JavaScript-klient utvecklad i GWT och ett JSON-API för kommunikation mellan klient och server.

Leads används för styrning av säljprocessen och där går att hitta information om en säljares alla ***leads***, viss statistik över den egna säljprocessen, historik över gjorda affärer med mera.

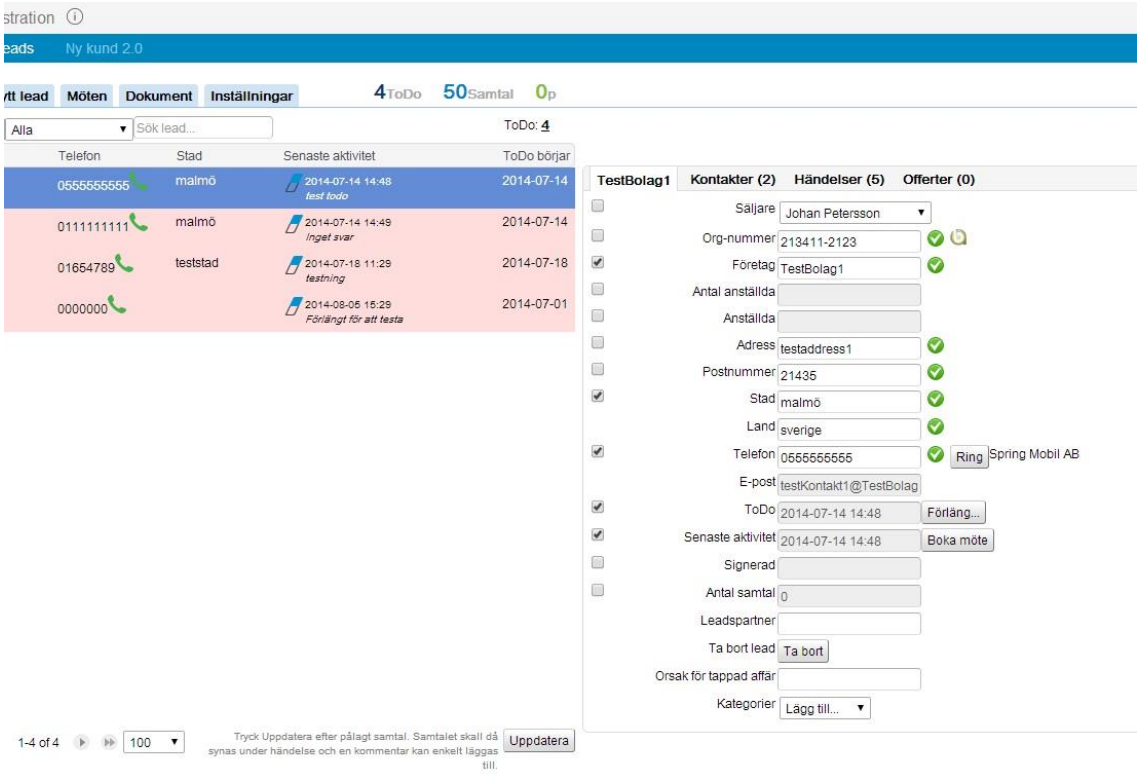


Figur 2
Sidan ”Min dag” vilket är den första sidan som användaren ser efter inloggning.

I Figur 2 visas sidan ”Min dag” vilket är den första sidan som användaren ser när denne har loggat in. Här visas information om:

- Hur många ***ToDos***, det vill säga aktiviteter som säljaren har liggande i planeringen.
- Hur många samtal av dagens kvot som är kvar att ringa.
- Listor över kommande och genomförda möten.
- Statistik över tagna poäng och ordervärden.

Figur 3 visar sidan i ***Leads*** där en säljare kan ta fram en lista över alla sina ***leads*** och ta fram detaljerad information om och även modifiera befintliga ***leads***.

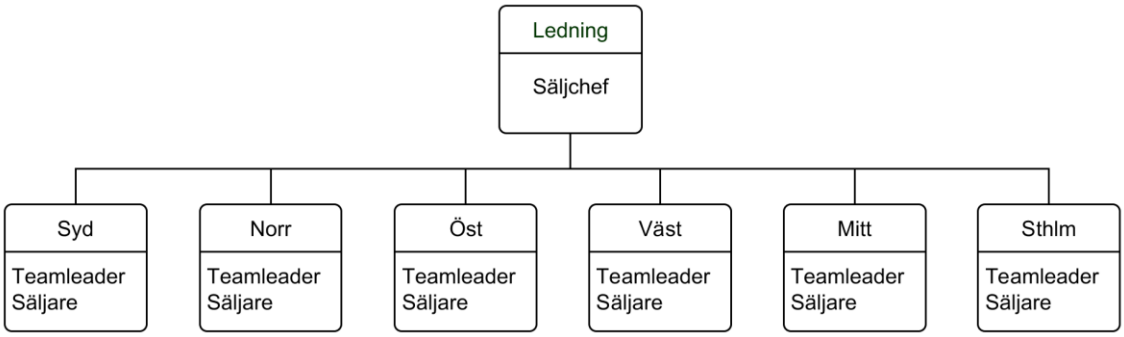


Figur 3
Sidan där en säljare se alla sina *leads* och ta fram detaljerad information om dessa.

I ***Leads*** finns även en flik med länkar till alla tänkbara dokument med information som en säljare kan behöva ha tillgång till vid kontakt med potentiella kunder.

3.1.2 Säljorganisationen

I säljorganisationen finns i toppen en säljchef, där under finns sex säljteam bestående av en Teamleader/Regionchef med ansvar för en grupp med mellan sex och elva säljare (se Figur 4). Säljchefen utför normalt inte själv något säljarbete förutom i väldigt stora affärer.



Figur 4
Telavox säljorganisation.

3.2 Utveckling av grundläggande funktionalitet

Som första steg i examensarbetet beslutades att utveckla appens grundläggande funktionalitet, det vill säga login mot servern och kommunikation med ***Leads API***. Detta för att skapa förståelse för systemets uppbyggnad samt hur kommunikation mellan klient och server fungerar.

3.2.1 Utvecklingsmiljö och använda verktyg

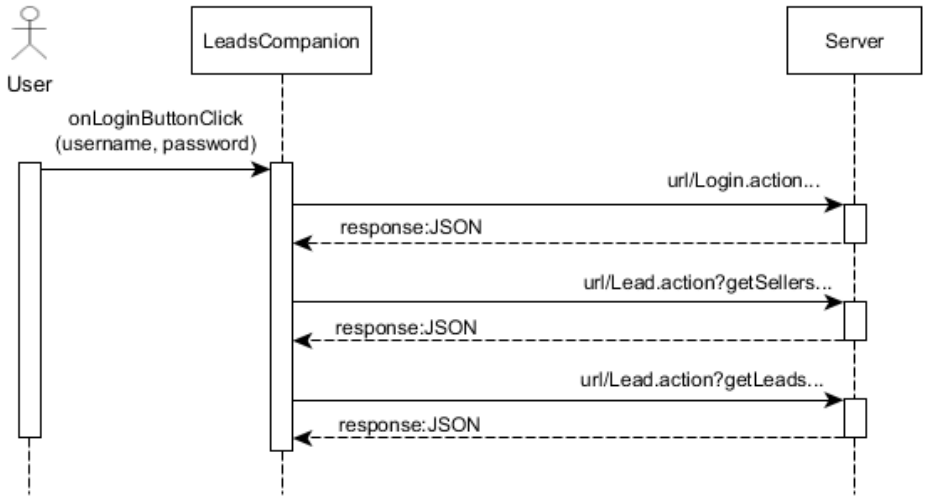
Till all utveckling användes utvecklingsmiljön *Eclipse* [2] med *ADT-plugin* (*Android Development Tools* [3]) och för versionhantering användes *Git* [4] vilket Telavox tillhandahöll ett *repository* till på onlinetjänsten *GitHub* [5].

3.2.2 Grundläggande funktionalitet

För att i det här skedet hålla utvecklingsarbetet så enkelt som möjligt skapades ett program i ren *Java* utan att blanda in något från *Android API*.

Detta kördes sedan mot en testserver. Tanken var att skapa förståelse för dels formatet *JSON* [6] vilket var ett okänt område, dels för att se till så att kopplingen mot servern fungerade, först okrypterat via *Http*- [7] och sedan krypterat via *Https*, det vill säga *Http* över *SSL/TLS* [8].

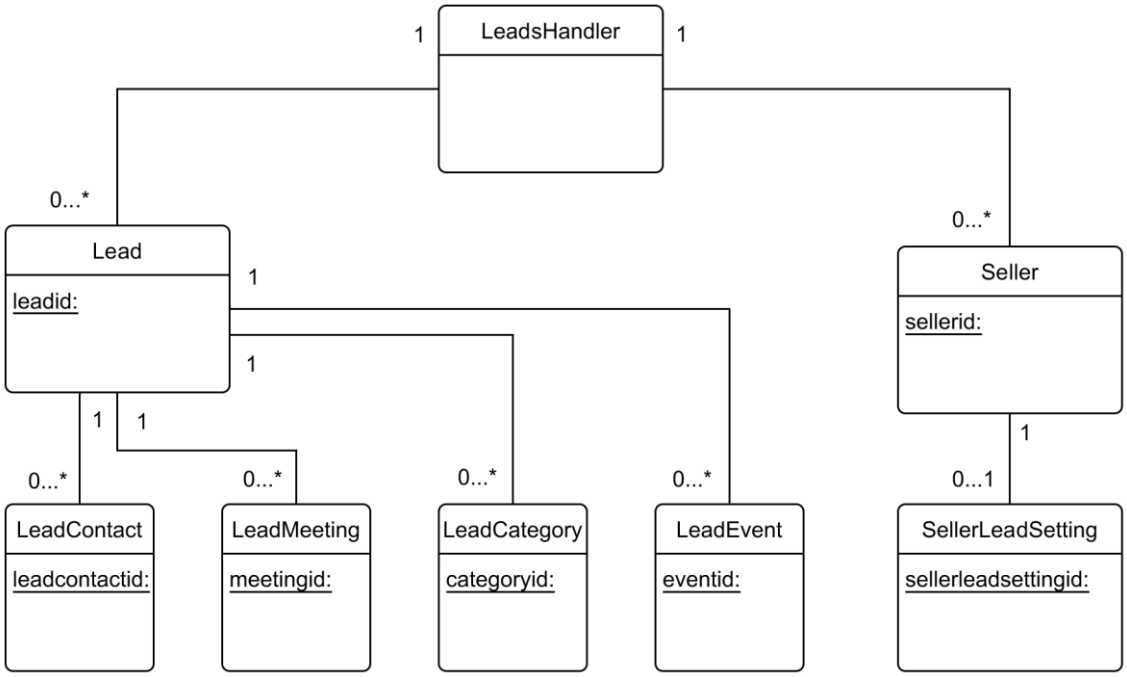
All kommunikation mot servern sker genom att man med *Http*-anrop ber servern om data via fördefinierade metoder i ***Leads API***. Servern returnerar data i form av *JSON-objekt*. Sekvensen för inloggning och datahämtning beskrivs i *UML*-sekvensdiagrammet i Figur 5.



Figur 5
UML-sekvensdiagram som beskriver sekvensen för inloggning.

Genom att studera informationen och strukturen i de hämtade *JSON-objekten* skapades sedan en grundläggande datamodell för appen. För att göra det så enkelt som möjligt beslöts att varje *JSON-objekt* som hämtas ut skall motsvaras av ett *Java-objekt* i den lokala datamodellen. Det togs beslut om att en *klass*, ***LeadsHandler***, skall ta hand om all hantering av och koppling mellan de olika *objekten* i datamodellen. Eftersom alla *JSON*-objekt i den befintliga datamodellen i ***Leads*** har ett unikt id som går att använda som nycklar var den enkel att implementera med datastrukturen *Map*. Utöver detta skall ***LeadsHandler*** även hantera all leverens av information till det grafiska gränssnittet. Ett *relationsdiagram* av den

preliminära datamodellen, där alla attribut förutom nycklarna utelämnats, kan ses i Figur 6. En mera djupgående beskrivning av *JSON*-formatet finns i avsnitt 4.3.3.



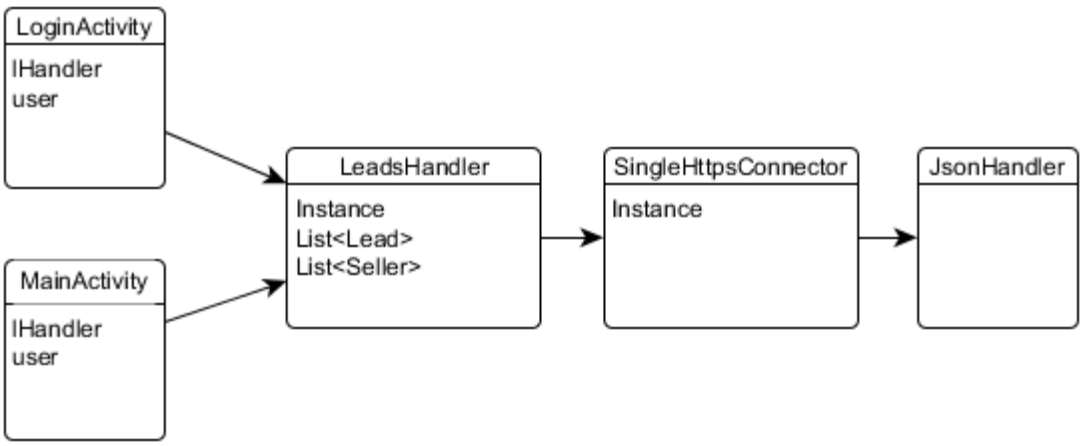
Figur 6
Relationsdiagram av datamodellen för Leads.

Det beslutades att ett antal *klasser* för hantering av specifika uppgifter i systemet skulle skapas. ***HttpsConnector*** för hantering av all internetkommunikation, ***FileHandler***, för eventuell filhantering (Fram-förallt för att kunna spara nedladdade *JSON*-objekt på fil för att kunna utveckla utan att ständigt vara uppkopplad mot en server), och ***JsonHandler***, en *klass* för hantering av *JSON-objekt*.

När allt fungerade i ren javakod så skapades ett *Android-projekt* i *Eclipse* till vilket koden importerades för att se till att allt fungerade där. Projektet som skapades bestod av två *aktiviteter*. ***LoginActivity*** vilket bara visar ett formulär där användaren anger användarnamn och lösenord och en knapp för att bekräfta inloggningsförsök. Vid lyckad inloggning startas

MainActivity som i detta läget endast bestod av en enda textruta där det nedladdade *JSON-objektet* skrevs ut som ren text.

Ett problem som uppstod var hur man skulle föra över en internetsession mellan aktiviteterna då varje aktivitet använde en egen *instantiering* av ***HttpsConnector***-*klass*en. I första implementeringen extraherades *session-cookien* och skickades vidare i ett *Intent* till *aktiviteten* som öppnades, där den stoppades in i en ny *instans* av ***HttpsConnector*** innan uppkoppling skedde. Det fungerade bra men för att ge en tydligare struktur i designen av systemet samt för att undvika manuell *cookie*-hantering skrevs istället ***HttpsConnector***-*klass*en om så att den fungerar som en *Singleton* [9]. Det vill säga en *klass* som håller en enda *instantiering* av sig själv som ett *statiskt objekt*. För att komma åt *instantieringen* har *klass*en en metod *getInstance()*, och *klass*ens *konstruktor* är gömd. Genom att göra så här löstes problemet då det är samma instans av ***HttpsConnector*** och således samma *session-cookie*. Ett *klass*-diagram över den preliminära systemdesignen kan ses i Figur 7.



Figur 7
Klassdiagram av preliminär systemdesign.

3.3 Kartläggning av appens Innehåll

Efter att den grundläggande funktionaliteten och preliminära system-designen var klar påbörjades ett mer djupgående arbete med att bestämma appens innehåll och funktionalitet.

3.3.1 En första konceptskiss

Som ett första steg i den här delen av examensarbetet togs en skiss av ett tänkbart appkoncept fram. Arbetet genomfördes genom att två skisser gjordes, en av författaren av den här rapporten, och en av handledaren på företaget. Skisserna jämfördes sedan och en ny skiss baserad på innehållet i de två första skisserna togs fram. Tanken bakom arbetet med skisserna var att försöka skapa en första vision om hur appen skulle kunna se ut, samt för att ge underlag till vidare undersökning i form av intervjuer och arbete med fokusgrupp.

3.3.2 Fokusgrupp

När den första preliminära skissen av appen blivit godkänd av handledaren sattes en fokusgrupp ihop. Gruppens deltagare representerade olika intressen i säljorganisationen, ledning och slutanvändare (säljare).
Det bör påpekas att i sammanhanget används ordet fokusgrupp annorlunda än definitionen i till exempel ”Software requirements, styles and techniques”, i vilken fokusgrupp beskrivs som en slags mer strukturerad form av brainstormingsessioner [10]. Istället definieras fokusgrupp i den här rapporten som:

En grupp bestående av intressenter och tilltänkta slutanvändare vilka innehar ingående domänkunskap och skall fungera som en kontinuerlig resurs av information, med åsikter om utformning samt som deltagare i pilottestning av den utvecklade produkten.

Gruppens syfte var att ge underlag för att hitta innehåll, funktionalitet och hantering av appen för att appen på bästa sätt skall fungera som ett verktyg i säljprocessen.

3.3.2.1 Gruppens sammansättning

Fokusgruppen bestod av tre personer med lite olika roller i säljorganisationen. En säljchef som tidigare jobbat som säljare under fem år, två säljare varav en har ett förflutet som teamleader och en är en av de säljare som vanligtvis säljer mest. Som direkta tilltänkta användare av appen förväntades de två säljarna bidra med insikt i det dagliga säljarbetet samt vilka behov som finns och problem som kan uppkomma. Säljchefen ingick i gruppen för att ge perspektiv från managementhåll och kunskap om styrning av säljprocessen.

3.3.3 Intervjuer

Intervjuer med personerna i fokusgruppen genomfördes för att ge kunskap om och insikt i hur det dagliga säljjobbet går till och vilka utmaningar som det kan innebära. En av grundpremisserna för appen var att den skall fungera som ett komplement och inte ersättare av det befintliga ***Leads***. Med det som utgångspunkt förväntades intervjuerna även ge underlag för att bestämma vilket innehåll i och vilken funktionalitet hos ***Leads*** som bör återfinnas i appen. Intervjuerna förväntades även ge upphov till idéer om hur tävlingselement ska kunna införas på bästa sätt, och hur appen skall kunna fungera som ett verktyg för ökad motivation hos säljarna.

Intervjuerna genomfördes med en respondent i taget för att få varje enskild deltagares perspektiv och önskemål utan den påverkan som kan uppstå då man genomför gruppintervjuer. Informationsinsamling via intervjuer anses enligt ”Software requirements, styles and techniques” [11] passa bäst för att kartlägga befintliga arbetssätt och processer, därför lades fokus på det. Den sammanställda informationen som intervjuerna gav finns i avsnitten 3.3.3.2 - 3.3.3.7.

3.3.3.1 Frågor

Intervjumetoden som valdes var öppen riktad intervju [12] då det ansågs lämpligare än mera strukturerade intervjuformer på grund av att det ger respondenten utrymme att förklara företeelser mera ingående. Intervjuernas längd varierade mellan trettio och sextio minuter och spelades in för att transkriberas i efterhand. Transkriberingen gjordes inte ordagrant utan istället skrevs innehållet ned i en punktlista av påståenden. Påståendena kategoriserades och sorterades baserat på om de bedömdes som önskemål om funktionalitet eller innehåll i den blivande appen, beskrivning av ett upplevt problem, beskrivning av en funktion i det befintliga systemet eller beskrivning av en process i det dagliga säljarbetet. En andra sortering av materialet gjordes för att minimera antalet påståenden med likvärdig eller samma innebörd. Den listan av påståenden ställdes sedan mot frågorna för att kontrollera så att alla frågor blivit besvarade. När kontrollen gjorts användes påståendena som grund till preliminära krav i en presumptiv kravspecifikation. De områden som behandlades under intervjuerna samt de frågor som intervjuerna förväntades ge svar på listas här nedanför.

Säljprocessen

- Vilka steg ingår i säljprocessen och vad innebär de?
- Vilka av dessa steg bör ske på kontoret och vilka kan ske i fält?

Dashboard

- Vad kan varje säljare se i ***Leads*** idag?
- Skiljer sig vad användaren kan se beroende på position (säljare, teamleader) och bör skillnaden även finnas i appen?
- Vilken information är viktig för säljaren då denne befinner sig i fält?
- Vilken information bör visualiseras?
- Är det något som saknas i nuvarande ***Leads***?

Notifieringar/Händelser

- Vilka händelser bör trigga notifieringar?

Gameifiering

- Topplistan, brukar säljaren titta på den?
- Hur ser säljaren på *gameifiering*?
- På vilka sätt kan säljaren jämföra sig gentemot kollegor?

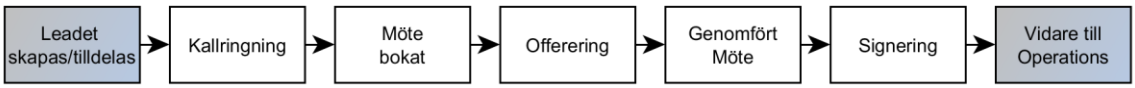
Övrigt

- Önskemål och övriga behov?
- Vilka mobilappar används ofta av säljaren då denne befinner sig i fält, om några?

Nedan följer en sammanfattning av den information som intervjuerna gav.

3.3.3.2 Säljprocessen

Säljprocessen består av ett antal förutbestämda processteg (Figur 8) : kallringning (första kontakt med en potentiell kund), bokat möte, genomfört möte, offerering och signering. Det förekommer även att säljaren knackar dörr, det vill säga tar kontakt med ett företag som ännu inte finns i systemet (***Leads***), då tillfälle ges. För tillfället ligger detta utanför styrningen av säljprocessen (Genererar inga poäng, se avs. 3.3.3.3, Processteg), men det är sagt att det skall ses över i framtiden. Då det i nuläget inte finns stöd för dörrknackning i ***Leads*** så ligger det utanför examensarbetets omfång. När ett ***lead*** är signat, det vill säga att ett kontrakt är underskrivet, går ärendet vidare till avdelningen Operations vilka ser till att det med kunden överenskomna abonnemanget blir igångsatt.



Figur 8
En säljares olika aktiviterer på ett Lead, från tilldelning till det att Leadet blivit signerat och vidarelämnat till Operations.

3.3.3.3 Processteg

Vissa led i säljprocessen ger säljaren ett visst antal poäng. Målet för varje dag är att säljaren skall ha kommit upp i 500p fördelat på kalla samtal, bokade möten och genomförda möten. Ett kallt samtal, ger 10 poäng, ett bokat möte ger 50 poäng och ett genomfört möte ger 125 poäng. Vidare så finns ett mål att ha 20 bokade möten varje månad. Säljarna brukar avsätta varje tisdag till att vara på kontoret för att sitta och kallringa.

Ett ***lead*** kan tilldelas en säljare på ett av två sätt:

1. Säljaren skapar själv ett nytt ***lead*** i webbgränssnittet och måste då skriva in tre stycken obligatoriska fält (Orgnummer, företagsnamn och företagets telefonnummer). Då ett nytt ***lead*** skapas görs en sökning på organisationsnumret mot alla redan existerande ***leads*** samt befintliga kunder för att kontrollera potentiella dubletter.

2. Ett ***lead*** tilldelas säljaren av teamleadern.

Då säljarna skriver offerter eller kallringer behövs det ofta att information om kostnader och olika avtal finns lätt tillgängligt. På grund av detta anses dessa steg som olämpliga att utföra i fält och således finns inget behov av stöd för dessa steg i appen. Det finns dock önskemål om att även nya ***leads*** skall kunna skapas i appen.

3.3.3.4 Dashboard

I ***Leads*** ligger idag en lång lista med poster (***ToDos***) vilket kan vara samtal, möten som säljaren har inbokade eller offerter som skall skrivas. Varje post är kopplad till ett ***lead***. Samtal registreras automatiskt då de utförs från en telefon registrerad i ***Partner*** (affärssystemet som ***Leads*** är en del av), både stationär och mobil. Då samtalet görs från den stationära telefonen så öppnas automatiskt ett dialogfönster där säljaren kan skriva in en kommentar. För varje post finns en detaljvy för det specifika ***leadet*** där information om företaget, kontaktkort och händelsehistorik finns

tillgänglig. Varje förändring av posten skapar automatiskt en händelse vilket kan vara att ett möte bokats in, ett samtal gjorts eller att tiden för en ***ToDo*** flyttats fram till ett senare tillfälle. Bokade möten registreras automatiskt via synkronisering med säljarens kalender, exempelvis *MS Outlook*.

I ***Leads*** visas grafiskt och i text hur mycket av dagens poängkvot som finns kvar och hur många samtal och möten som är utförda. Det går även att se viss historik över tagna poäng för den enskilde säljaren. En teamleader kan förutom sin egen statistik även se poängstatistik över säljarna i sitt team.

3.3.3.5 Notifieringar/Händelser

Det finns önskemål om att i appen ha en rullande vägg med händelser rörande företaget i stort liknande de händelseväggar som finns i sociala medier. Händelser som kan vara intressanta att visa är nya gjorda affärer och hur stora de är samt vilken säljare som står för den. Poster kan möjligtvis markeras på något sätt beroende på storlek av affären. Det finns även önskemål om att kunna kommentera eller klicka på något motsvarande en ”like”-knapp på en post. I nuläget finns inget sådant här implementerat i ***Leads*** utan säljarna uppmanas att använda sig av programmet/tjänsten *Microsoft Sharepoint* [13] för att interagera socialt.

3.3.3.6 Gameifiering

På Telavox kontor i Malmö sitter det en skärm med en topplista över de säljare som säljer bäst per vecka, samt vilket värde säljarna sålt för. I intervjuerna framkom att säljarna ofta tittar på tavlan för att se om de finns med bland toppsäljarna och att det uppfattas som sporrande att försöka ligga i topp. Enligt säljaren med bakgrund som teamleader så strävar man ur ett teamleaderperspektiv efter att ha ett lag som drar eller pushar varandra, gärna med en jämförelse med andra lag. Man vill att alla ska kämpa för att det egna laget skall vara bäst.

Vid intervjuerna framkom att det anses att det kan vara ganska felvisande att bara jämföra värdena på genomförda affärer. Detta då det kan finnas de säljare som arbetar väldigt hårt men inte kommer upp på topplistan. För att lyfta och uppmuntra dessa kan man istället jämföra tagna poäng och eventuellt vad som genererat dessa.

3.3.3.7 Övrigt

I intervjuerna framkom önskemål om att kunna prioritera olika ***ToDos*** som finns i ***Leads***. Man vill även kunna sätta en markering huruvida det är ett första, andra eller tredjemöte eller att man ringt ett samtal och inte fått något svar. I nuläget behöver man gå in på det specifika ***leadet*** och söka på senaste händelse, vilket uppfattas som omständligt. Man vill även på ett enkelt sätt kunna få upp en karta eller vägbeskrivning till ett företag utan att behöva kopiera addressen och klistra in den i en kartapp eller webbtjänst.
Förutom *Sharepoint* (se.3.3.3.5) så uppmanas även alla säljare att installera appen *Genious Scan* [14] med vilken man på ett enkelt sätt kan fotografera dokument som sedan konverteras till ett *PDF*-dokument och skickas som email. Önskemål om liknande funktionalitet alternativt att öppna *Genious Scan* inifrån appen finns.

3.4 Framtagning av app-koncept

Efter genomförda intervjuer med fokusgruppen lades det resulterande materialet in i ett *Microsoft Excel*-dokument och formulerades om till preliminära krav. Dokumentet var inte en regelrätt kravspecifikation utan användes mer som en checklista över innehåll och funktionalitet hos appen. De resulterande kraven gicks igenom och kontrollerades tillsammans med Viktor Fogelberg och Henrik Thorvinger. Detta för att utröna om de var uppfyllbara baserat på om de skulle kräva ingrepp i det befintliga systemet och i så fall hur stora dessa skulle vara, samt för att kontrollera kravens validitet.

Med utgångspunkt i *Excel*-dokumentet började arbetet med att ta fram ett mera konkret koncept för appen. När innehållet och funktionalitet var bestämd togs tillsammans med Sofia, grafisk designer på Telavox, en skiss fram och presenterades för fokusgruppen.

3.4.1 Innehåll i appen

Det beslutades att appen skall bestå av fem huvudsidor vilka är åtkomliga från huvudmenyn.

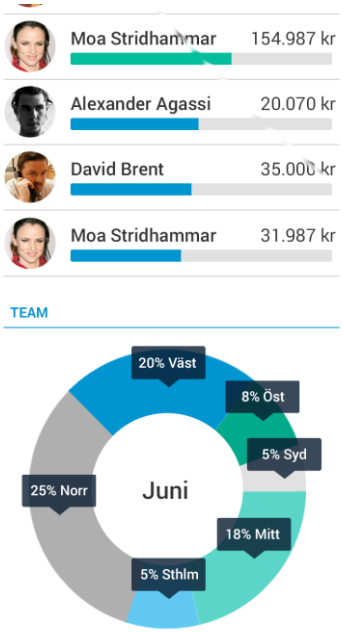
- En *DashBoard*-sida med visare som visar information om den inloggade användarens dagliga säljprocess (I menyn på skisserna kallas sidan Diagram).
- En sida med en lista över möten,
- En sida med en lista över alla ***ToDos***
- En sida med en anslagstavla, med information om händelser för företaget i stort och eventuellt möjlighet för säljarna att kommentera och interagera socialt.
- En sida med en listor och diagram över hur säljare och säljteam står i jämförelse med varandra.

I huvudmenyn skall även finnas länkar till eventuella andra appar som en säljare kan ha nytta av då denne befinner sig i fält, exempelvis *Genious scan* och *SharePoint*. Visarna på *dashboard*-sidan skall även återfinnas i fältet högst upp på skärmen i appen, området som kallas *actionBar* vilken även skall innehålla ett fält för möjlighet att söka på ett telefonnummer eller organisationsnummer. Detta för att kunna få upp information om ett företag på det viset. Vidare så beslutades att en användare genom att klicka på ett av elementen i en lista skall få upp en detaljsida med information om det specifika ***Leadet***. Detaljsidan skall bestå av två flikar:

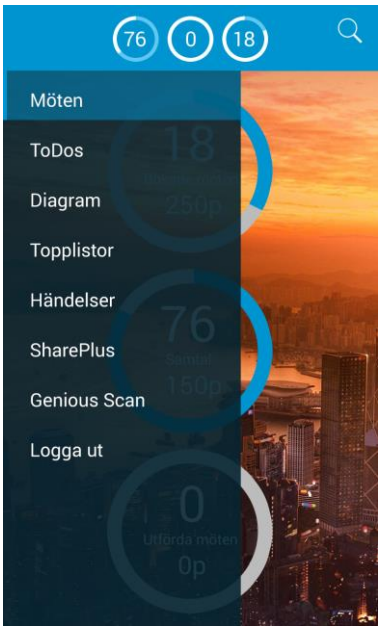
- Ett kontaktkort med information om företaget och eventuella kontaktpersoner.
- En lista med historik över alla händelser som finns registrerade på ***Leadet***.

NavigationDrawer [15] valdes som huvudupplägg för navigation i appen vilket beskrivs mer i detalj i avsnitt 4.2.2.1.

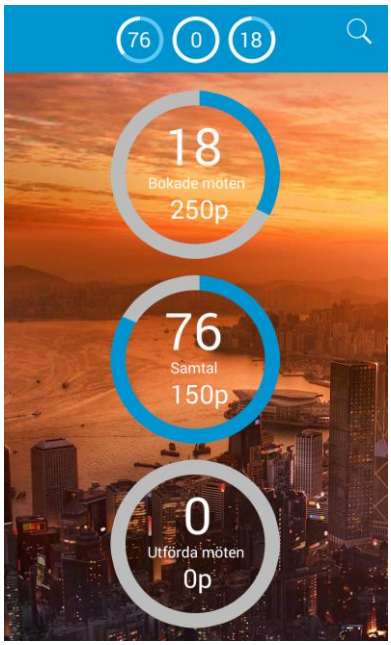
Nedan (I figur 9 - 12) visas bilder från skisserna som Telavox grafiska designer Sofia tog fram för app-konceptet.



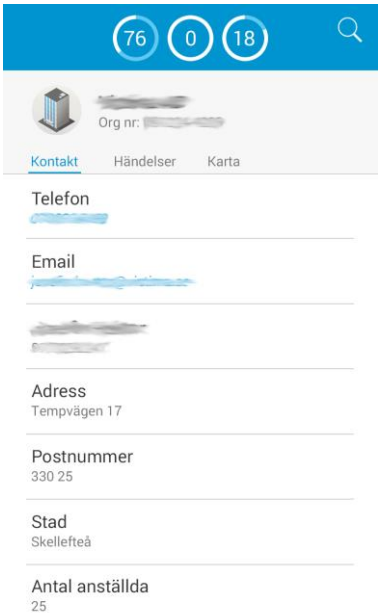
Figur 9
Sidan topplistor.



Figur 10
Utfälld meny.



Figur 11
Sidan Diagram (Dashboard).



Figur 12
Detaljsidan för ett Lead.

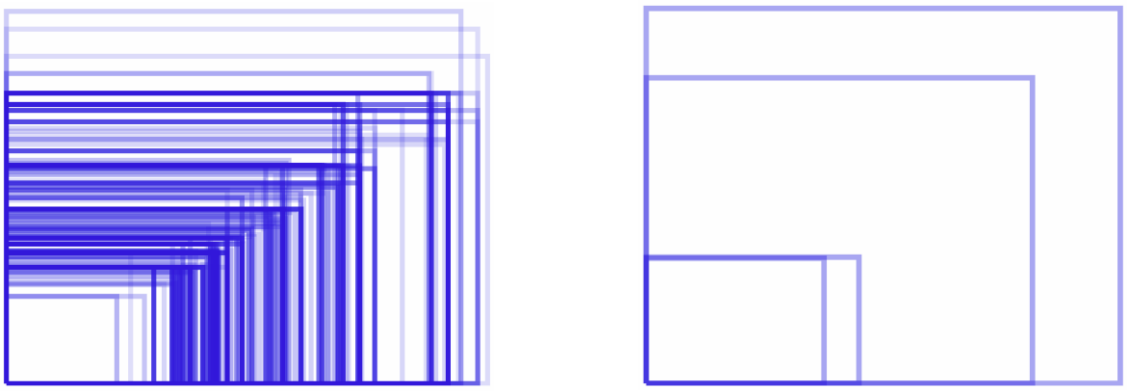
4 Teknisk bakgrund

Det här kapitlet går översiktligt igenom vad det innebär att utveckla mot Androidplattformen. Här ges även viss teoretisk bakgrund till de två frågeställningarna som examensarbetet syftat ge svar på. Slutligen ges en översiktlig genomgång av de *klassbibliotek* som har använts i utvecklingen av appen.

4.1 Att utveckla för Androidplattformen

Vid utveckling av programvara till mobila enheter krävs att utvecklaren tar hänsyn till saker som enhetens begränsade minne, tillgängliga skärmstorlekar med mera. Detta gäller oavsett mobilplattform som programvaran utvecklas mot.

Vid utveckling för *Android*-plattformen behövs dessutom tas hänsyn till den stora mängden olika tillverkare och enheter som finns på marknaden, man brukar kalla detta för fragmentering. En rapport [16] från en undersökning gjord av Open Signal [17] beskriver hur fragmenteringen ser ut men menar även att det kan ses som en styrka. Detta då den stora andelen enheter på marknaden gör att en användare förmodligen har lätt att finna en enhet som täcker dess exakta behov. Figur 13 visar bilder tagna från Open Signals rapport som beskriver skillnaden i fragmentering mellan plattformarna *Android* och *IOS* vad gäller skärmstorlek.



Figur 13
Skillnaden i skärmstorlekar hos androidplattformen (vänster) och IOS (höger). Bilderna är tagna från OpenSignal report 2013 och används med tillstånd.

4.2 Utveckling mot Android-enheter med olika skärmstorlekar

I guiden för att utveckla mot olika skärmstorlekar på *Androids* webbsida för utvecklare [18] ges fyra konkreta råd vad man bör tänka på, dessa är:

1. Att använda ***wrap_content***, **fill_parent** eller enheten *dp* vid specificering av dimensioner i en *XML-Layout*-fil.

2. Att inte använda *hårdkodade* pixelvärden i koden för applikationen.

3. Att inte använda *AbsoluteLayout*, vilket inte heller längre stöds i *Android API*.

4. Vid användning av *bitmaps*, se till att det finns versioner med olika storlek tillgängliga för olika skärmstorlekar. Det räcker med att lägga filerna i dess respektive folder vid utvecklingen. När applikationen körs så väljs rätt bild automatiskt av systemet.

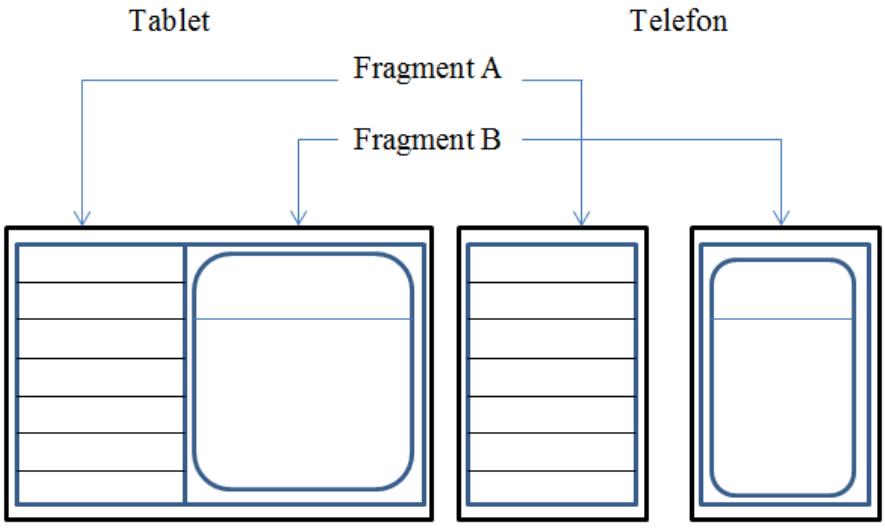
4.2.1 Densitets- och skalnings-oberoende pixlar

För att göra det enklare för utvecklare har man gjort en grov klassificering av *Android*-enheter med avseende på skärmstorlek och *pixel*-densitet. Skärmstorlekar är uppdelade i: liten, normal, stor och extrastor. *Pixel*-densitet (dpi, dots per inch) hos skärmar är uppdelade i: låg, medium, hög och extra hög. Då även en liten skärm kan ha en hög *pixel*-densitet används ytterligare två mått för att definiera *pixlar*; *densitetsoberoende pixlar(dp)* och *skalningsoberoende pixlar(sp)*. *Densitetsoberoende pixlar* är som namnet antyder oberoende av skärmens *pixel*-densitet, det vill säga skärmens upplösning. En *densitetsoberoende pixel* uppvisar samma fysiska storlek på skärmar med olika upplösning.

Skalningsoberoende pixlar (*sp*) används för att definiera storlek på text och de gör så att till exempel en 10 punkters text alltid har samma fysiska storlek oavsett storlek på skärm och densitet hos denna.

4.2.2 Fragment

Fragment [19] är en komponent införd i *Android 3.0 (API level 11)* och kan ses som utbytbara moduler för en *Androidaktivitet*. Varje *fragment* har sin egen *livscykel* och kommunicerar med *aktiviteten* och andra *fragment* via *callback-metoder*. I *Androids* riktlinjer för användning av *fragment* anges att varje *fragment* bör ha ett *Interface* med *callback-funktioner* vilka *aktiviteten* som använder *fragmentet* måste *implementera*. En av anledningarna till att *fragment* infördes i *Android API* var för att underlätta utveckling till olika enheter med skillnad i skärmstorlek, exempelvis mellan telefoner och tablets. *Fragment* ger möjlighet att vid *runtime* baserat på skärmstorlek skapa *dynamiska* layouts. En utvecklare kan definiera en layout att använda för telefoner och en layout att använda för tablets. I Figur 14 visas ett exempel på hur man kan göra med fragment. En app kan ha en listvy (Fragment A) och en detaljvy (Fragment B). Då appen körs på en tablet läggs de två fragmenten sida vid sida medan då appen körs på en telefon visas ett fragment i taget.

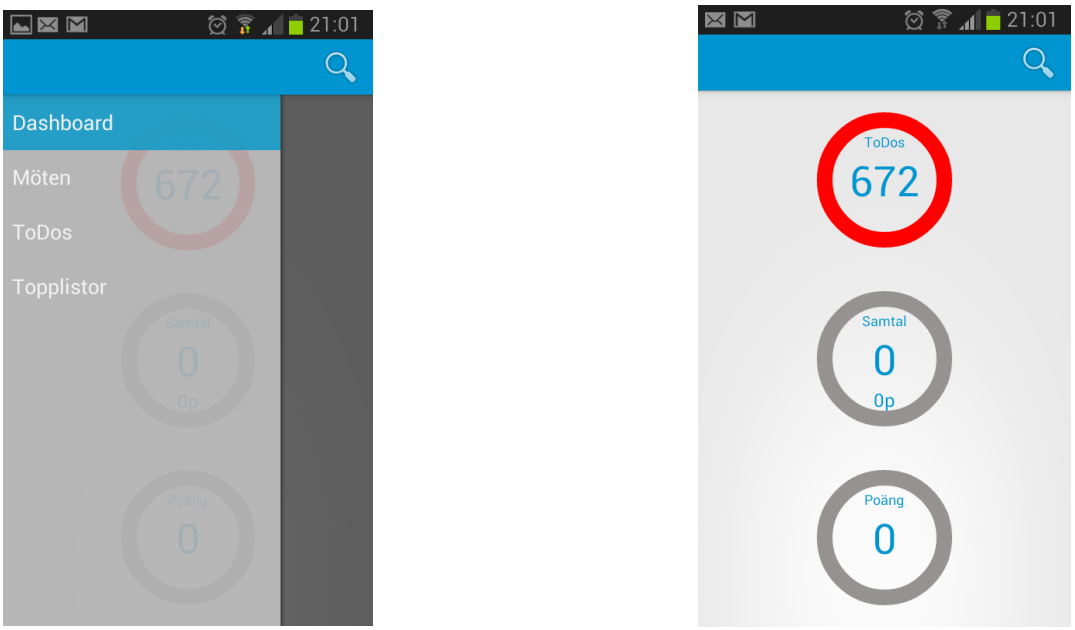


Figur 14
Exempel på användning av fragment i en Android-app

4.2.2.1 Navigationsmönstret NavigationDrawer.

Navigationsmönstret NavigationDrawer [20] är ett mönster där man från kanten av skärmen kan *swipa* in en meny till de olika sidorna i appen (se Figur 15). En fördel med detta *navigationsmönster* är att alla sidor i appen är tillgängliga vilken sida man än befinner sig på för tillfället. Istället för att starta en ny *aktivitet* varje gång man byter sida så byter man istället *fragment* i *aktiviteten*.

I *Eclipse ADT* finns det möjlighet att välja olika mallar då man skapar ett nytt Android-projekt. En av dessa mallar är ”Navigation Drawer Activity” vilken redan har ett *DrawerFragment* som är enkelt att utgå ifrån för att sedan modifiera till ens behov.



Figur 15
T.v. NavigationDrawer i utfällt läge. T.h. samma sida men med infälld Navigation-Drawer, båda bilderna är skärmdumpar från den utvecklade appen.

4.3 Lazy Load och Push notifications

I den nu befintliga implementeringen av ***Leads*** sker alla beräkningar och sammanställning av data i klienten. Detta gör att all data om en säljares ***leads*** som finns lagrade i den centrala databasen hämtas när användaren loggar in på klienten. Eftersom överföringshastigheter i det mobila nätet oftast är lägre än hos övriga Internetuppkopplingar (fiber/adsl/wifi) kan det påverka användarupplevelsen negativt. Genom att minska mängden data som kommuniceras mellan server och klient kan man även minska laddningstiderna.

I examensarbetet har koncepten *Lazy Load* och *Push notifications* studerats som potentiella lösningar och i det här avsnittet beskrivs dessa.

4.3.1 Lazy Load

Lazy Load innebär att en applikation inte laddar in en komponent eller data från den underliggande datastrukturen förrän just vid tillfället som det behövs. *Lazy Load* används normalt för att minska tiden för uppstart av program genom att begränsa mängden information som behöver laddas in i minnet. I ”Patterns of Enterprise Application Architecture” [21] diskuteras fyra sätt att implementera *Lazy Load* vilka beskrivs kortfattat här nedanför.

4.3.1.1 Lazy Initialization

Metoder som använder Lazy Initialization kontrollerar varje gång ett *objekt* eller värde skall returneras om det finns eller inte. Om *objektet* eller värdet inte finns, det vill säga är *null* så skapas *objektet* eller beräknas värdet innan det returneras av metoden.

4.3.1.2 Virtual proxy

En *Virtual Proxy* är ett *objekt* som för systemet ser ut som det *objekt* systemet förväntar på en viss plats. Ett *Virtual Proxy objekt* innehåller dock inte någon som helst data. Det egentliga *objektet* skapas först när systemet försöker göra *metodanrop* på det. Enligt ”Patterns of Enterprise

Architecture” lämpar sig *Virtual Proxy-objekt* bäst för klasser av typen *Collection*, till exempel olika typer av listor då systemet annars lätt kan bli väldigt komplext.

4.3.1.3 Value holder

En *Value Holder* är ett *objekt* som innesluter det egentliga informationsbärande objektet vilket innan det anropas för första gången är *null*. Då objektet efterfrågas skapas det genom anrop till datastrukturen och hålls sedan kvar i *Value Holder-objektet* för framtida åtkomst.

4.3.1.4 Ghost

Ett objekt av typen *ghost* är det riktiga objektet men utan något mer innehåll än objektets id. Innehållet fylls i från den underliggande datastrukturen först när objektet behövs.

4.3.2 Push Notifications

Push Notifications innebär att servern i ett server-klient scenario initierar kontakt istället för tvärtom. Det finns lite olika teknologier för att åstadkomma push och i examensarbetet har tittats på två stycken, *CometD* och *GCM*, (*Google Cloud Messaging*) vilket beskrivs mer ingående i avsnitt 4.4.4.

CometD [22] är baserat på *Bayeux-protokollet* [23] och används för närvarande av Telavox för push till webb-klienten för ***Leads***. På grund av att *CometD* använder sig av *Long polling* [24] som metod för *push* finns det risk att det tar för mycket batteri. *Long polling* innebär att istället för att servern initierar en *push* till klienten så kontrollerar klienten i vissa förbestämda intervall om servern har något att skicka ut. I de fall servern inte har något att lämna ut till klienten så istället för att skicka ett tomt svar, vilket sker med ”*Short polling*”, så håller servern kanalen öppen tills den har information den vill pusha ut till klienten eller att en timeout inträffar.

CometD stödjer även protokollet *websockets* men då Telavox server-arkitektur inte gör det i dagsläget har detta inte tittats på i examensarbetet.

4.3.3 JSON

All data och information som kommuniceras mellan server och klient i ***Leads*** skickas i JSON-formatet.

JSON (*Java Script Object Notation*) är ett dataöverföringsformat baserat på *Javascript* och är definierat i Ecma standard-404 [25]. *JSON* är enkelt att läsa från och skriva till och är byggt kring två strukturer, *JsonObject* och *JsonArray*.

- Ett *JsonObjekt* består av ett antal fält i par om namn / värde.
- En *JsonArray* består av en ordnad lista av värden.

Ett värde kan vara något av typerna: *String*, *Number*, *True*, *False*, *Null*, *Object* eller *Array*.

Ett *JsonObjekt* består av ett antal namn/värde-par inneslutna inom måsvingeparenteser, medan en *JsonArray* består av ett antal värden inneslutna inom hakparenteser.

- Ett enkelt *JsonObjekt*: { ”namn” : ”Åke Bengtsson”, ”ålder” : 32 }
- En enkel *JsonArray*: [”namn_1” , ”namn_2” , ”namn_3”]

Dessa två grundstrukturer kan sedan kombineras för att skapa mer komplexa strukturer. Som ett exempel visas nedan strukturen hos ett *JsonObjekt* för ett *lead*.


```
{
   "postalcode":"223 63",
   "events":[["read"],[[
{
   "leadeventid":897494,
   "time":"1395242960000",
   "type":"read",
   "msg":"Leadet skapades av johan petersson",
   "leadid":108750
}]]],
```

```
   "street":"Stora varvsgatan 6",
   "regnr":"061705-5801",
   "sellerid":2573,
   "categoryids":[],
   "leadid":108750,
   "contacts":[],
   "country":"Sverige",
   "city":"Malmö",
   "stateEventid":897494,
   "company":"Testföretag Tilde-Nova AB",
   "deliverysellername":"",
   "sellername":"johan petersson",
   "meetings":[],
   "telephone":"040123456",
   "employees":""
}
```


4.3.4 JSON-Parsers

För att läsa från och skriva till *JsonObject* används *JSON-parsers*, kodbibliotek med *klasser* för läsning/skrivning och hantering av *JSON*. Det finns en mängd olika alternativ att välja från. I det här examensarbetet har använts två olika bibliotek, *org.json* och *android.util.JsonReader* vilka fungerar lite olika. Dessa båda bibliotek beskrivs mer ingående i nästa avsnitt.

4.4 Använda API och klassbibliotek

Här följer en beskrivning av de *API* och *klassbibliotek* som förutom *Android API* och *Javas* standardbiblioteket har använts i utvecklingsarbetet.

4.4.1 Org.json

Org.json [26] är ett *klassbibliotek* för hantering av *JSON* och användes tidigt i examensarbetet eftersom det är enkelt att använda. Biblioteket arbetar efter en *DOM*-liknande princip (Document Object Model [27]). Det vill säga att man läser in hela objektet som ett dokument innan parsning sker. I en miljö med begränsat arbetsminne som i mobila enheter kan detta bli ett problem om det är ett *JsonObject* med mycket data som skall behandlas.

4.4.2 Android.util.JsonReader

I *klassbiblioteket* util i *Android API* finns två *klasser* för hantering av *JSON*, *JsonReader* och *JsonWriter*. *JsonReader* är en enkel *parser* för *JSON* som istället för att läsa in ett helt *objekt* innan bearbetning arbetar på *dataströmmar*. Genom detta blir *JsonReader* många gånger mer minneseffektiv och kan hantera *parsning* snabbare än parsers som läser in ett helt dokument.

JsonReader kräver dock mer vid implementering i jämförelse med *org.json* då man måste implementera en *parser* själv med *metoderna* som tillhandahålls [28].

4.4.3 HttpsURLConnection

HttpsURLConnection [29] är en *klass* som används för att skicka och ta emot data över internet via *Https*-protokollet och är en *subklass* till *HttpURLConnection*. Klassen innehåller stöd för komprimering av data i formatet *GZIP* [30] vilket sker automatiskt om man inte uttryckligen säger till att *GZIP* inte skall användas. Ett alternativ till *HttpsURLConnection* är

”*Apache HTTP Client*” [31] vilken på webbsidan ”Android Developers Blog” [32] rekommenderas för utveckling mot android versioner upp till och med 2.2, medan *HttpsURLConnection* rekommenderas för version 2.3 och senare.

4.4.4 Google Cloud Messaging

Google Cloud Messaging (*GCM*) [33] är en tjänst som *Google* tillhandahåller för push notifications och består av *API* för implementation av både klient och server. För att använda *GCM* krävs att man registrerar tjänsten på *Google Play Services* och att även enheterna som *GCM* skall användas på har *Google Play Services API* installerat. Tjänsten fungerar genom att varje enhet kopplad till tjänsten får ett unikt id och *nyckel* som är kopplad till ett konto på *Google Play Services*. Servern som implementeras med *GCM server API* skickar informationen som skall pushas ut till enheterna till *Googles server* vilken levererar den vidare till de enheter som skall ha den.

4.4.5 Google Analytics

För att kunna spåra fel och oväntade händelser vid test av användande av applikationen i fält kommer *Google Analytics* [34] att användas. *Google Analytics* är ett gratis verktyg som tillhandahålls av Google och består av kodbibliotek för olika plattformar inklusive *Android*, *IOS* och Web, samt ett konfigurerbart webgränssnitt med *dashboard* för visning av spårad data.

4.4.6 AndroidPlot

AndroidPlot [35] är ett bibliotek och *API* till för att enkelt kunna skapa olika grafer och diagram i en *Android*-app. Biblioteket är släppt med open source licensen *Apache Licence 2.0* och har använts för att skapa tårtbitsdiagram och visare för presentation av data.

4.4.7 AsyncTask

För att undvika att *UI-tråden* i en *Android*-app utför arbete som kan ta lång tid, exempelvis hämta data från en *server* eller lokal databas, finns i *Android API* en speciell *klass* som heter *AsyncTask* [36]. *AsyncTask* startar upp en ny *tråd* i vilken en tidskrävande åtgärd kan köras utan att påverka *UI-tråden*. När åtgärden är slutförd signalerar den detta till *UI-tråden* via en *callback-metod* och stänger sedan ner sig själv. *AsyncTask* innehåller fyra förbestämda *metoder* vilka är:

- *onPreExecute*()
  Om det är något som behöver ställas in eller kontrolleras innan det egentliga arbetet skall utföras så sker det här. Det kan till exempel vara att kontrollera att en viss resurs är tillgänglig.

- *doInBackground*()
  Här sker det egentliga arbetet, exempelvis ett anrop mot en server, eller ett uppslag mot en databas lokalt i enheten.

- *onProgressUpdate*()
  Den här metoden används om man vill visa en processindikator, som till exempel hur mycket av en fil har laddats ner och hur mycket som återstår.

- *onPostExecute()*
  När arbetet är utfört kallar *AsyncTask* automatiskt den här funktionen med det definierade returvärdet. Härifrån kan sedan processen notifieras om att arbetet är utfört och agera efter det.

Som ett exempel visas i Figur 16 metoden ***fetchAllLeadsTask(...)*** i *klass*en ***AsyncTaskHandler*** (se avs. 5.1) vilken används för att hämta ut alla ***Leads*** från servern.

```
 * @throws IOException
 */
public static void fetchAllLeadsTask(
        final HttpsConnectorAsyncCallbacks activity, final boolean offline)
        throws IOException {
    new AsyncTask<Void, Void, List<Lead>>() {
        @Override
        protected void onPreExecute() {
            mConnector = SingleHttpsConnector.getInstance();
        };

        @Override
        protected List<Lead> doInBackground(Void... params) {
            List<Lead> leads = null;
                try {
                    leads = mConnector.getAllLeadsAction();
                } catch (IOException e) {
                    throw new IOException(
                            "Error getting data from server", e);
                }
            }
            return leads;
        }

        @Override
        protected void onPostExecute(List<Lead> leads) {
            activity.postFetchLeads(leads);
        }
    }.execute();
}
```

Figur 16
Metod i klassen AsyncTaskHandler vilken används för att hämta ut alla leads från servern.

5 Implementering av appen

Den här delen av rapporten beskriver övergripande vad som har utvecklats under examensarbetet och hur designen av appen ser ut.

5.1 Systemdesign och utvecklade klasser

För systemdesignen hos applikationen har en filosofi att en *klass* skall ha hand om en sak försökt hållas. Konkret har det inneburit att vid implementeringen så har minimalt med logik lagts i *Activity-klasserna* och dess tillhörande *fragment*, istället har ett antal *klasser* för hantering av specifika uppgifter använts.

Appen innehåller följande hanterings-*klasser* vilkas *metoder* är *statiska*, det vill säga att de finns tillgängliga utan att *objekt* av *klasser*na behöver *instantieras*.

FileHandler
Klass för hantering av läsning från och skrivning till fil.

GCMHandler
Klass för hantering av *Push notifications* via *Google Cloud Messaging*

IOStreamHandler
Klass med endast en *metod* för att läsa ut en *teckensträng* ur en ström.

JsonHandler
Klass för *parsning* av *JSON*.

SharedPrefHandler
Klass som används för att spara användardata mellan *inloggningssessioner*.

SingleHttpsConnector
Klass som hanterar all kommunikation med *servern*, använder sig av *HttpsURLConnection*.

SingleLeadsHandler
Klass som hanterar den lokala *datastrukturen* för appen. *Klassen* håller en *statisk instans* av sig själv för att bibehålla den data som *Klassen* hanterar. Den har även hand om att sammanställa och leverera data till användargränssnittet. Hur det går till beskrivs mer ingående i avsnitt 5.2.

SingleStatisticsHandler
Klass som hanterar, sammanställer och levererar informationen som visas på sidan ***TopListFragment*** i applikationen. Varje gång sidan öppnas hämtar den ny data från servern via asynkrona anrop genom ***AsyncTaskHandler***.

AsyncTaskHandler
Klass som innehåller metoder för asynkrona anrop via *AsyncTask*. *AsyncTask* är en komponent i *Android API* som beskrivs mer ingående i avsitt 4.4.7.

5.2 Koppling mellan lokal datastruktur och användargränssnitt

Här görs en genomgång av de *objekt* och *metoder* som används av *klass*en ***SingleLeadsHandler*** för att sammanställa informationen som visas på de olika sidorna i appen.

5.2.1 Lokal datastruktur

SingleLeadsHandler har *objekt* av typen ***Lead*** lagrade i en *datastruktur* av typen *Map<key, value>* där variabeln ***leadid*** är nyckel för det specifika ***leadet*** och ***leadet*** i sig är värdet. Varje ***Lead*** har objekt av typerna ***LeadEvent*** och ***LeadMeeting*** lagrade i listor.

5.2.1.1 Innehållet i ett Lead-objekt

Objekten ***Lead***, ***LeadEvent*** och ***LeadMeeting*** i den lokala *datastrukturen* innehåller följande information:

Lead

- Information om företaget ***leadet*** representerar, namn, address, orgnummer etc.
- Information om säljaren ***leadet*** tillhör: id, namn.
- Id-nummer.
- En lista med objekt av typen ***LeadEvents***, händelser för leadet.
- En lista med objekt av typen ***LeadMeeting***, bokade och genomförda möten.
- En lista med objekt av typen ***LeadContact***, kontakter på företaget.
- En lista med objekt av typen ***SimpleQuote***, skrivna offerter för leadet.
- En variabel ***hideUntil*** av typen *Calendar*, som används för att dölja ett lead fram till ett visst datum.

LeadEvent

- Tidsstämpel för när händelsen skapades.
- Id-nummer för händelsen.
- Id-nummer för det ***lead*** som händelsen tillhör.
- Information om typ av händelse, fält för meddelande och kommentar.

LeadMeeting

- Id-nummer för mötet.
- Id-nummer för det ***lead*** som mötet tillhör.
- Starttid för mötet
- Längd på det bokade mötet.

5.2.2 Leverans av information till det grafiska gränssnittet

För att leverera information till användaren via det grafiska gränssnittet skapar både ***SingleLeadsHandler*** och ***SingleStatisticsHandler*** *objekt* som implementerar *Interfacen* ***CommonItem*** och ***CommonListItem***. ***Interfacen*** är skapade för att kunna använda *objekt* med olika innehåll i samma listor, exempelvis för att dela av listvyn med möten i sektionerna, dagens möten, kommande möten samt genomförda möten. Varje ***Item****-objekt* är kopplade till en specifik *Layout_xml*-fil vilken beskriver hur informationen visas på enhetens skärm.

5.2.2.1 Implementeringar av CommonItem och CommonListItem

SingleLeadsHandler använder sig av följande *klasser* för att leverera information till användargränssnittet:

- **DashboardItem**
  Innehåller en sammanställning av informationen som visas på sidan ***DashBoardFragment***. Här finns de värden som skall presenteras i visarna.

- **LeadDetailItem**
  Innehåller information om ett ***lead*** som visas på detaljsidan vilken öppnas då användaren har klickat på en post på någon av sidorna ”***MeetingFragment***” eller ”***TodosFragment***”, eller gjort en sökning på ett telefon-nummer i sökfältet. I ***LeadDetail***-objektet finns även två listor med *objekt* av typen ***EventListItem*** samt ***ContactListItem***.

- **ContactListItem**
  Innehåller information om en kontakt på företaget.

- **DetailListItem**
  Används för att bygga upp en lista med detaljer för ett lead för visning på detaljsidan. Innehåller två *strängar*: title och value.

- **EventListItem**
  Innehåller informationen från en händelse i ett ***leads*** händelselista.

- **MeetingListItem**
  Innehåller information hämtat från ett *objekt* i ett ***leads*** möteslista: Företagsnamn, start- och slut-tid, datum samt eventuell kommentar.

- **TodosListItem**
  Innehåller information om en ***ToDo***-post: Vilken typ av aktivitet det handlar om, tid för start av aktiviteten, tid för senaste händelse på det aktuella leadet, företagsnamn samt kommentar från senaste händelse.

- **ListTitleItem**
  Innehåller endast en *textsträng*, används för att dela upp listor i olika sektioner.

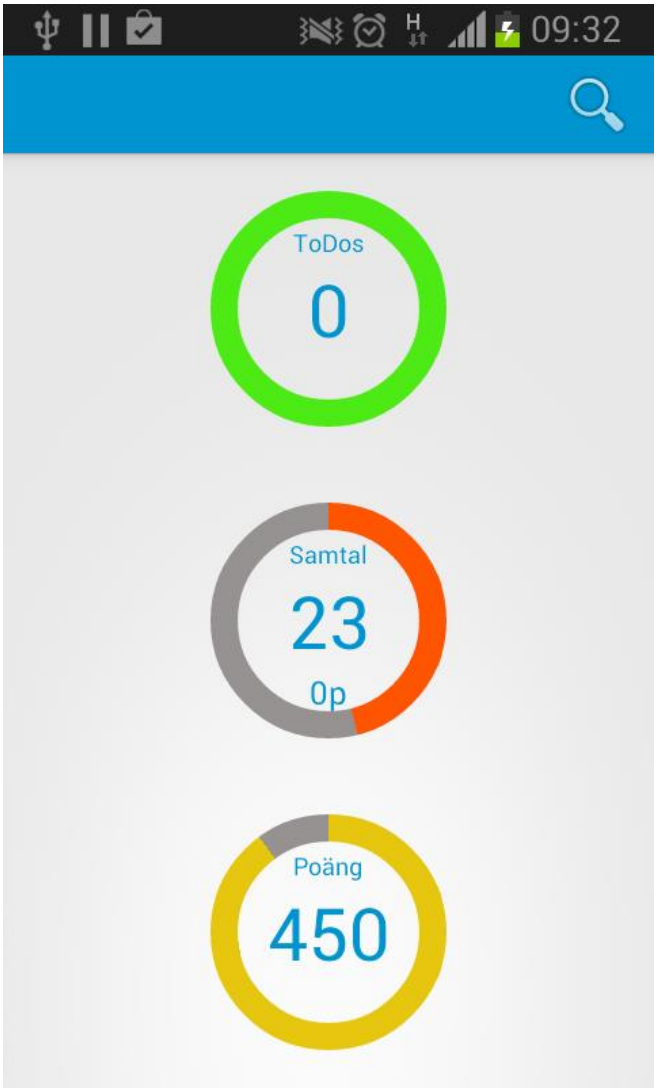
SingleStatisticsHandler använder sig av följande *klasser* för att leverera information till *användargränssnittet*.

- **SalesTeamItem**
  Innehåller namn på säljteamet, id för säljteamet, säljteamets sammanlagda värde på genomförda affärer samt antalet säljare som ingår i teamet.

- **TopListItem**
  Innehåller namnet på en säljare, säljarens nuvarande värde för den här månadens gjorda affärer, en heltalsvariabel för att spara ett referensid till en porträttbild av säljaren samt nuvarande poäng som säljaren genererat (de två sistnämnda fälten används i nuläget inte) .

5.3 Exempel på sammanställning av information som visas på skärmen

Här följer ett exempel som visar hur informationen sammanställs och levereras till sidan ***DashBoardFragment*** vilken visas i Figur 17.



Figur 17.
Sidan DashBoardFragment, den översta cirkeln visar antalet ToDos som säljaren har att göra. Cirkeln i mitten visar antal telefonsamtal somsäljaren har ringt på en dag. Den nedersta cirkeln visar antalet poäng som säljaren har genererat på en dag.

Sidan innehåller tre visare i form av cirklar:

- Den översta cirkeln beskriver antalet ***ToDos*** som säljaren har att göra. Baserat på antalet ***ToDos*** så ändrar cirkeln färg vid specifika tröskelvärden, från grönt via gult och orange upp till rött.

- Den mellersta cirkeln visar antalet ringda samtal för en dag som finns registrerade i systemet. Cirkeln består av två segment, ett grått och ett som ändrar färg beroende på det aktuella värdet, från rött, via orange till gult. När antalet samtal överstigit 50, vilket är det dagliga målet, byter cirkeln färger och har gult som bas och grönt som färg för visaren.

- Den nedersta cirkeln fungerar på samma sätt som den i mitten men visaren visar istället dagens tagna poäng.

När användaren väljer att visa sidan ***DashBoard*** från menyn byts *fragmentet* som för närvarande visas ut mot ***DashBoardFragment***. I ***DashBoardFragment*** anropas automatiskt metoden *onCreateView()*, vilket alltid görs vid byte av *fragment*. ***DashBoardFragment*** anropar i sin tur *metoden* ***getDashBoardItem()*** i ***SingleLeadsHandler*** vilken skapar ett *objekt* av typen ***DashBoardItem***, sammanställer och sparar informationen i avsedda fält i *objektet*, och sedan returnerar det till *fragmentet*. Figur 18 visar ett urdrag ur *metoden*.

```
for (Lead l : leads.values()) {
    if (l.getHideUntil() != null && checkIfTimePassed(l.getHideUntil())
            && checkIfValid(l)) {
        todos++;
    }
    for (LeadEvent le : l.getLeadEvents()) {
        if (le.getType().equals("callout")
                && checkIfToday(le.getTime())) {
            actDailyCalls++;
        }
        if (le.getType().equals("meeting")
                && checkIfToday(le.getTime())) {
            actBookedMeetings++;
        }
    }
    for (LeadMeeting lm : l.getMeetings()) {
        if (checkIfTimePassed(lm.getStart())
                && checkIfToday(lm.getStart())) {
            heldMeetings++;
        }
    }
}
```

Figur 18.
Utdrag från koden i metoden *getDashBoardItem()* i *SingleLeadsHandler*. Metoden går genom alla *Leads* som en säljare har och kontrollerar om villkor är uppfyllda för att tas med i informationen som skall visas på sidan. När genomgången är klar returnerar metoden informationen i ett objekt av typen *DashBoardItem*.

Sammanställningen sker genom att alla ***leads*** gås igenom och kontrolleras mot kriterier för att de skall räknas med i informationen som skall levereras till användargränssnittet.

Exemplet visar att den nuvarande implementeringen av systemet kräver att *klienten* alltid har en säljares alla ***leads*** tillgängliga lokalt för att appen skall kunna visa tillförlitlig information.

5.4 Möjlig tillämpning av Lazy load i appen

Som tidigare beskrivits (se avs 4.3) så bör man försöka minska mängden data som kommuniceras mellan server och klient i en mobil applikation. Detta för att undvika långa laddningstider vilket kan medföra försämrad användarupplevelse. Ett sätt att åstadkomma detta är att använda sig av *Lazy Load.*

En möjlig lösning som tittats på under examensarbetet är att använda en variant på *design mönstret ghost* (se avsnitt 4.3.1.4) för att först hämta ut en lista med alla ***leads***, fast innehållande minimalt med data, det vill säga endast informationen som visas i listan och objektens id. När detaljer sedan behövs, exempelvis när användaren klickar på ett element i en av listorna i appen för att ta fram detaljer om ett ***lead***, görs ett anrop till servern med leadets id vilken levererar ***leadet*** med fullständig information i ett nytt *JSON*-objekt till appen som sedan presenterar informationen på detaljsidan.

Ett enkelt test för att mäta tiden det tar att hämta data från servern genomfördes. I testet användes data från 1 säljare och omfattar omkring 3000 *leads.* Det genomfördes för att ge en uppskattning om hur stor skillnad det ger av att använda designmönstret *ghost*. Resultatet från testet kan ses i Tabell 1.

Mängd data i ett Lead	**Antal sekunder för hämtning av data**
Lead med endast id	2.46
Lead med möten och id	4.1
Lead med all data	8.0

Tabell 1.
Resultat av test att hämta ut en säljares alla *leads* från servern med olika mycket data . Testet är gjort på en testserver på det lokala nätverket och sker utan komprimering och utan kryptering.

Testet visar att det finns potential att minska mängden data som hämtas initialt med ungefär en tredjedel. Då storleken på ett ***lead*** skiljer sig från ett annat skall testresultatet ses endast som en fingervisning och mer tester krävs för att ge mera tillförlitlig data.
För närvarande sker all sammanställning av data från ***Leads*** i klienten vilket beskrevs i avsnitt 5.3. Detta gör att med den nuvarande implementeringen av systemet kräver att hela datastrukturen behöver finnas tillgänglig lokalt hos klienten och väldigt lite data kan hämtas med *Lazy Load*.

5.5 Alternativ lösning

Vad som har gjorts för att förhindra att användarupplevelsen försämras på grund av långa laddningstider är att all data som hämtas från servern sparas till en *cache*-fil. När appen öppnas laddas data från *cache*-filen in i den lokala datastrukturen samtidigt som ny data hämtas från servern. Den lokala datastrukturen uppdateras sedan med den nya informationen när hämtningen är klar. Genom att göra så här så gömmer man effektivt laddningstider från användaren.

5.6 Utveckling mot olika skärmstorlekar

I implementeringen av det grafiska gränssnittet skapades två grundlayouts, en för telefoner vilken alltid är låst i porträttläge, och en för tablets som alltid är låst i landskapsläge. Vid uppstart av ***MainActivity*** mäts skärmbredden för enheten och beroende på det uppmätta värdet laddas layouten anpassad för den enhetstypen. Grundlayouten består endast av en *FrameLayout* [37] med en eller flera ytor att stoppa *fragment* i. I fallet telefon så laddas en layout in där menyn ligger i en *NavigationDrawer* och endast ett *fragment* är synligt åt gången på skärmen (se Figur 26 på sidan 63). I layouten för en platta visas tre *fragment* samtidigt på skärmen (se Figur 27 på sidan 63). Längst till vänster ligger meny-*fragmentet* och i mitten visas *fragmentet* som valts i menyn. Längst till höger är *fragmentet* med visare av processinformation placerat, vilket även tagits bort som menyval i menyn.
För att åstadkomma det här skapades ett nytt *ListFragment* vilket innehåller menyn som visas i tablet-läge. *Fragmentet* innehåller endast en lista och använder sig av samma *ListAdaptor* [38] och *layout-XML* fil som drawer-*fragmentet* vilket gjorde implementeringen väldigt enkel.

5.6.1 Placering av fragment och skärmorientering

Då appen körs på en tablet används ej *navigation-drawer*. Utan istället läggs Meny-*fragmentet* längst till vänster, den i menyn valda sidan presenteras i mitten och sidan med visarna läggs längst till höger. Sättet att åstadkomma det här på är att vid uppstart av huvudaktiviteten, mäta skärmstorleken och att baserat på pixelbredd på skärmen välja layouttyp. Genom att kalla på enhetens konfiguration med anropen *getResources()* och *getConfiguration()* som finns tillgängliga från från *klassen Activity* går det att mäta skärmen bredd i antalet *densitetsoberoende pixlar* (*dp*) och på det viset avgöra vilken grundlayout som skall laddas av applikationen. Koden som ligger i metoden *onCreate(...)* i ***MainActivity*** vilket åstadkommer detta kan ses i Figur 19.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Configuration config = getResources().getConfiguration();
    if (config.smallestScreenWidthDp >= 600) {
        isTablet = true;
        // Device is a Tablet
        setContentView(R.layout.activity_main_tablet);

        // Make sure the application is always in landscape mode
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        mNavigationMenuFragment = (NavigationMenuFragment) getFragmentManager()
                .findFragmentById(R.id.navigation_menu);
    } else {
        isTablet = false;
        // Device is a phone
        setContentView(R.layout.activity_main_handheld);

        // Make sure the application is always in portrait mode
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

        mNavigationDrawerFragment = (NavigationDrawerFragment) getFragmentManager()
                .findFragmentById(R.id.navigation_drawer);

        // Set up the drawer.
        mNavigationDrawerFragment.setUp(R.id.navigation_drawer,
                (DrawerLayout) findViewById(R.id.drawer_layout));
    }
    // get instance of SingleLeadsHandler
    lHandler = SingleLeadsHandler.getInstance(this);
}
```

Figur 19.
Koden i metoden onCreate() i MainActivity. Genom att mäta skärmbredden med anropet smallestScreenWidthDp i config går det att avgöra om appen körs på en tablet eller telefon och rätt Layoutfil laddas in.

Figur 20 visar Layout-filen som används då applikationen körs på en telefon. Layouten består av en *FrameLayout*, i vilken de olika sidorna i applikationens *fragment* placeras *dynamiskt* vid *runtime*, och ett *statiskt definierat fragment* för *NavigationDrawer*.

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="se.telavox.leadscompanion.main.MainActivity" >

    <FrameLayout
        android:id="@+id/container"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <fragment
        android:id="@+id/navigation_drawer"
        android:name="se.telavox.leadscompanion.main.NavigationDrawerFragment"
        android:layout_width="@dimen/navigation_drawer_width"
        android:layout_height="match_parent"
        android:layout_gravity="start" />

</android.support.v4.widget.DrawerLayout>
```

Figur 20
Layoutfilen activity_main_handheld.xml

I Figur 21 visas Layout-filen som används när applikationen körs på en enhet med större skärm, här består Layouten istället av två *statiskt definierade fragment*, ett för menyn som ligger längst till vänster och ett för sidan med visare som ligger längst till höger. Mellan de båda *fragmenten* finns en *FrameLayout* i vilken den i menyn valda sidan visas.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:baselineAligned="false"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="se.telavox.leadscompanion.main.MainActivity" >

    <fragment
        class ="se.telavox.leadscompanion.main.NavigationMenuFragment"
        android:id="@+id/navigation_menu"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="2"/>

    <FrameLayout
        android:id="@+id/container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"/>

    <fragment
        class ="se.telavox.leadscompanion.views.dashboardview.DashboardFragment"
        android:id="@+id/dashboard"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="2"/>
</LinearLayout>
```

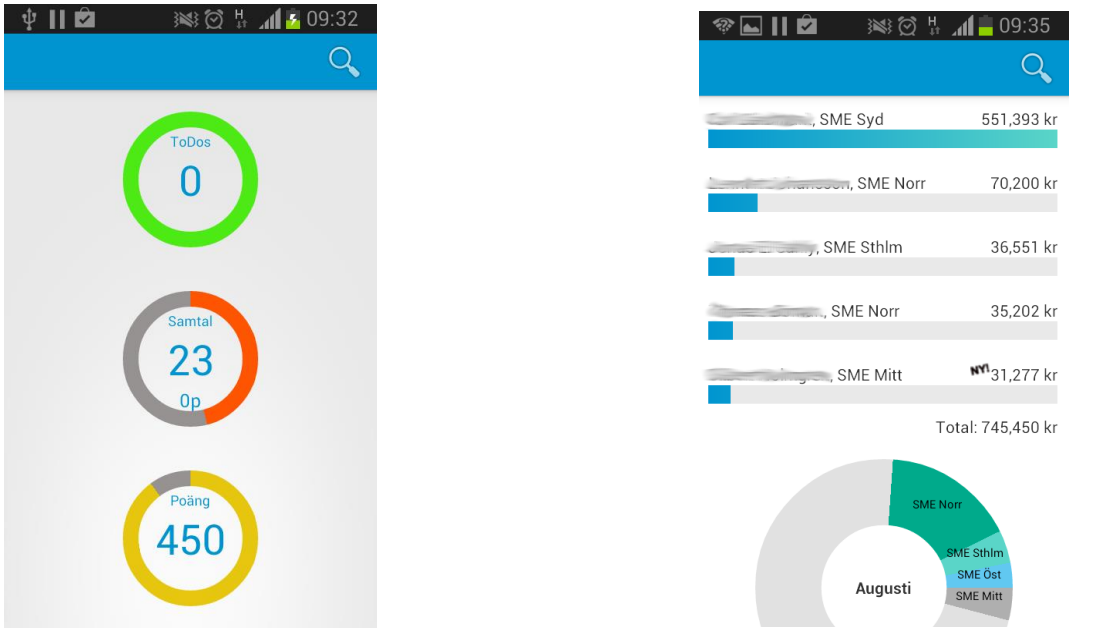
Figur 21
Layoutfilen activity_main_tablet.xml

6 Resultat

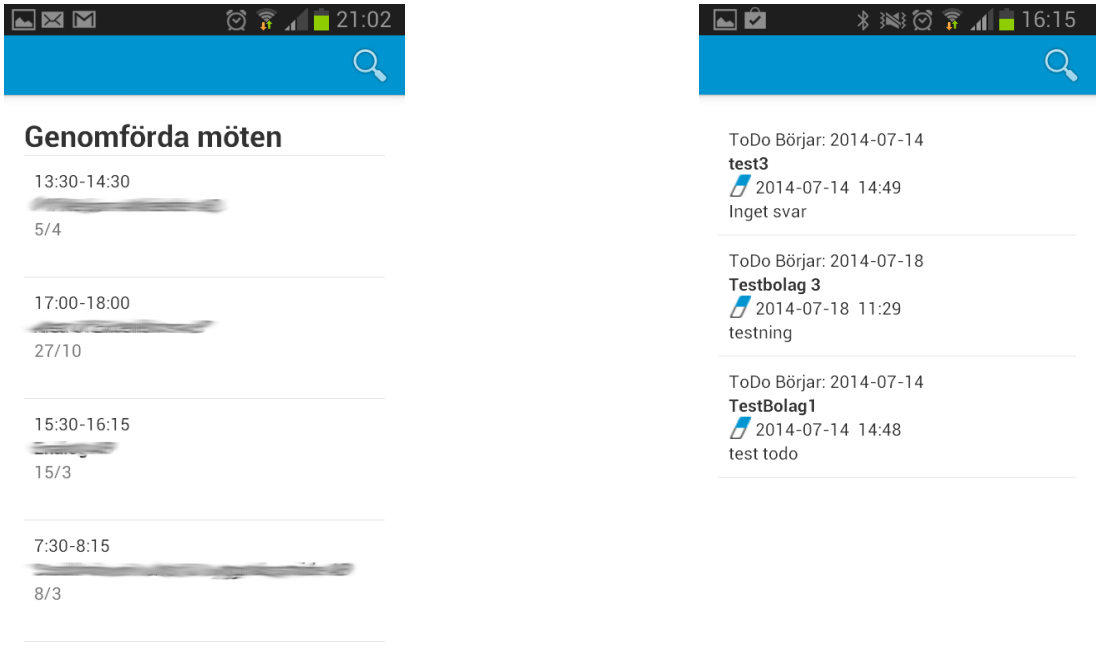
En fungerande app har blivit utvecklad som uppfyller alla grundläggande krav (se. 1.3). Innehåll och funktionalitet motsvarar vad som blev bestämt i undersökningsprocessen i början av examensarbetet. Under arbetet har metoder för att minska negativ användarupplevelse på grund av låga uppkopplingshastigheter samt möjligheter att enkelt utveckla för enheter med olika skärmstorlekar studerats.

6.1 Den utvecklade appen

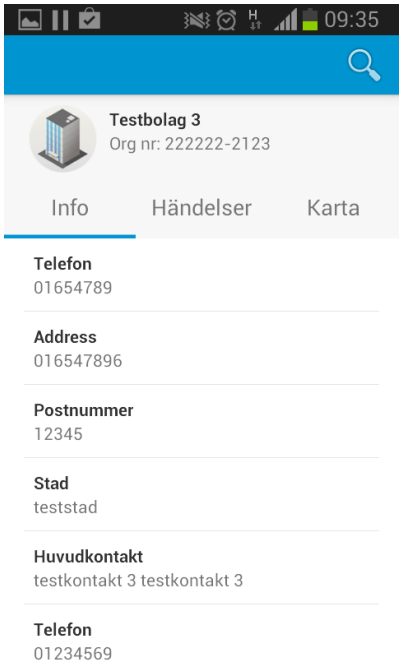
Appen består av två *Activities*, ***LoginActivity*** och ***MainActivity***. ***MainActivity*** har fyra huvudsidor vilka är: Dashboard, Möten, ToDos och Topplistor. Alla sidorna är tillgängliga från appens meny som finns i en *NavigationDrawer*. Sidorna Möten och ToDos visar båda listor över möten eller kommande aktiviteter som användaren har att göra. Då användaren klickar på ett element i en av listorna öppnas en ny sida med detaljer för ***leadet*** som elementet tillhör. På detaljsidan finns kontaktuppgifter till företaget samt historik över händelser för leadet. På detaljsidan finns också möjlighet att öppna Google Maps med det aktuella företagets address. Figur 22 - Figur 25 visar skärmdumpar av de olika sidorna från den utvecklade appen. Enheten som appen kördes på då skärmdumparna blev tagna är en Samsung Galaxy S2.

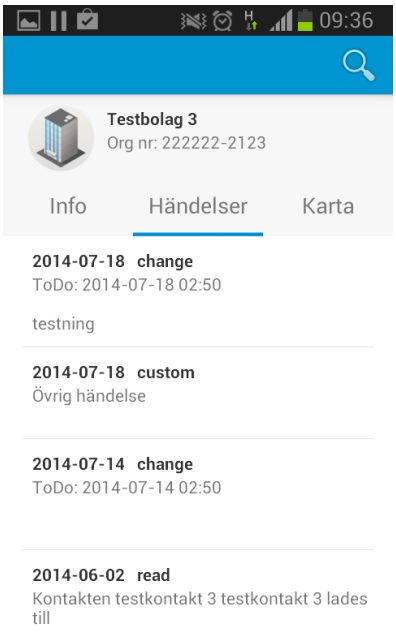


Figur 22
T.v. Sidan Dashboard som visar information om hur säljaren ligger till i sina dagliga mål. T.h. Sidan Topplistor, överst visas de fem just nu bäst säljande säljarna. Diagrammet under visar varje säljteams andel av den totala försäljningen.

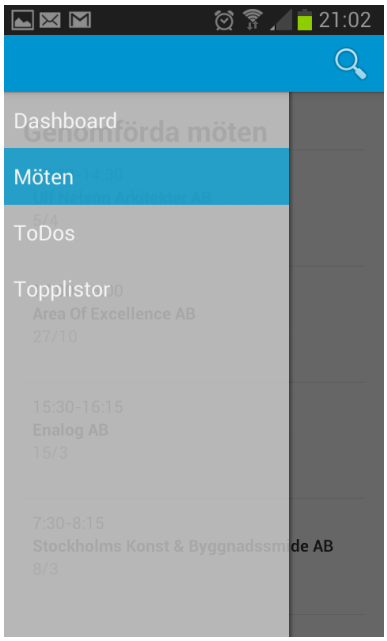


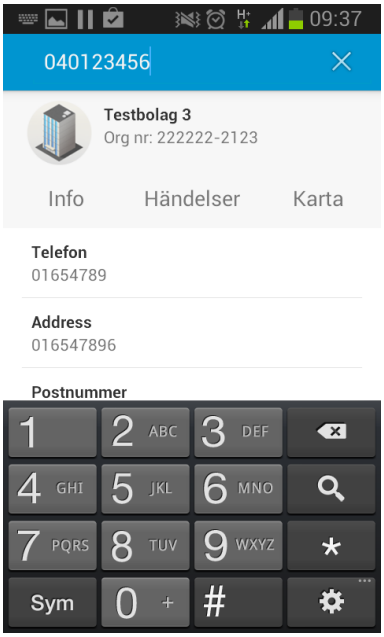
Figur 23
T.v Sidan med listan över möten. Då användaren klickar på ett listelement öppnas en detaljsida för elementet. T.h Sidan med ToDos, det vill säga planerade aktiviteter som användaren har att göra.





Figur 24
Detaljsida för ett *lead*. T.v detaljsidans kontaktkort. T.h lista på händelser för *leadet*. Klickar användaren på knappen ”Karta” så öppnas Google Maps med företagets address som sökargument.





Figur 25
T.v Menyn i en NavigationDrawer i utfällt läge. T.h Då sökfältet används visas den resulterande sökningen på en likadan sida som detaljsidan.

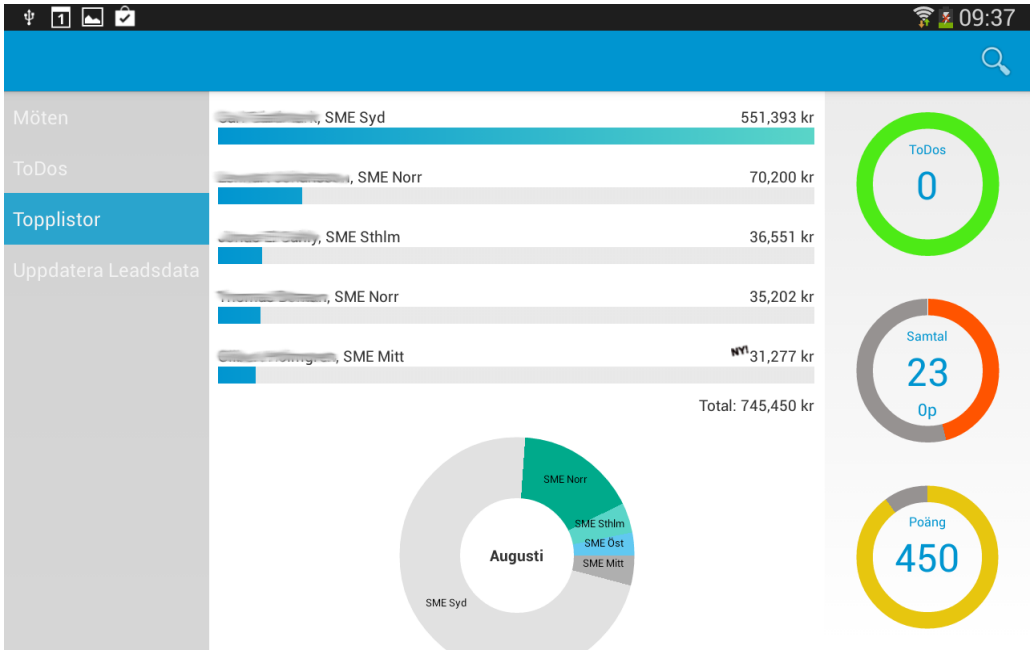
6.2 Metoder för att undvika att låga överföringshastigheter påverkar användarupplevelsen negativt

Lazy Load har studerats som en möjlig strategi för att minska mängd data som behöver kommuniceras mellan server och klient. Detta för att genom komma runt de problem som kan uppkomma då användaren befinner sig i ett område med endast *2G*-uppkoppling. Då alla beräkningar och all sammanställning av data sker i klienten i det nuvarande systemet går det inte att använda *Lazy Load* för att minska mängden data som kommuniceras mellan klient och server.

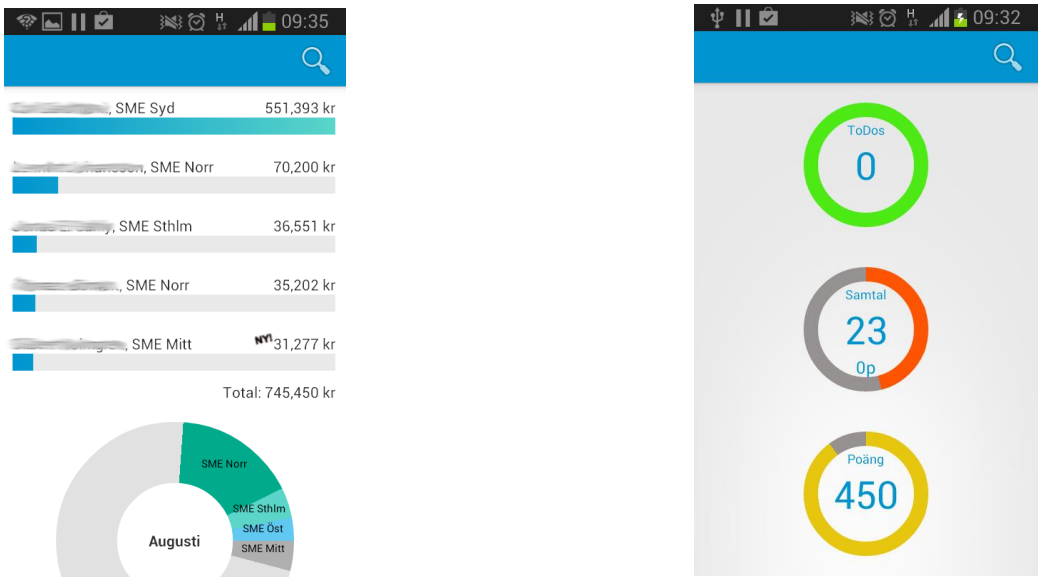
För att undvika alltför långa laddningstider har istället en lösning där hämtad data sparas på en *cache*-fil implementerats. Lösningen gör så att användaren inte märker av tiden det tar att hämta data från servern då detta alltid görs i bakgrunden. En nackdel är dock att informationen som finns lokalt i appen kanske inte alltid är i synk med servern. De fel som skulle kunna uppkomma på grund av detta har dock bedömts som försumbara i den här specifika applikationen.

6.3 Körning av appen på tablet och telefon

För att skapa ett användargränssnitt som fungerar likvärdigt på både tablet och telefon har *fragment* använts vid utvecklingsarbetet. Appen har testats på en Samsung Galaxy S2 och en Samsung Galaxy Tab 3.0 8” och uppvisar likvärdig funktion på de båda enheterna. I och med användningen av fragment så har det kunnat undvikas att skapa Layout-filer för specifika skärmstorlekar. Skillnaden på hur layouten ser ut på en tablet, Samsung Galaxy Tab 3.0 8” och en telefon, Samsung galaxy S2, kan ses i Figur 26 och Figur 27.



Figur 26
Skärmdump från en Samsung Galaxy Tab 3.0 8”



Figur 27
Skärmdumpar från en Samsung Galaxy S2 som visar samma sidor som Figur 26.

6.4 Vad har inte blivit implementerat

Följande av det innehåll och den funktionalitet som i det inledande arbetet bestämdes för appen har inte blivit implementerat.

- Det finns önskemål om att kunna jämföra poäng tagna av säljare i samma säljteam. Då systemet för tillfället endast tillåter en säljare att se sina egna poäng var detta inte realiserbart. För att implementering skall kunna ske krävs ingrepp på serversidan vilket inte har varit aktuellt för examensarbetet.

- En sida med en ”händelsevägg” eller anslagstavla har inte blivit implementerad. Viss experimentering av möjlighet att utnyttja Google Cloud Messaging har påbörjats. Funktionaliteten har behövts prioriteras bort på grund av tidsbrist och att även den kräver ingrepp på serversidan.

- Det var tänkt att visarna på sidan DashBoard även skulle finnas synliga i ActionBar i appen. På grund av tidsbrist och att de visarna inte har någon betydelse för appens funktionalitet har detta prioriterats bort.

7 Slutsatser

Målet med det här examensarbetet var att utveckla en app och i samband med det undersöka möjligheter att enkelt utveckla mot enheter med olika skärmstorlekar. Vidare syftade examenssarbetet till att hitta metoder för att minska negativ påverkan av användarupplevelsen på grund av låga överföringshastigheter.

7.1 Utveckling mot olika skärmstorlekar

Genom att använda sig av fragment går det att enkelt skapa ett modulärt användargränssnitt som kan anpassas vid körning av applikationen. Detta gör så att man slipper skapa flera komplexa *Layout-XML* då det räcker att definiera enkla grund *Layout-filer* vilka bestämmer placeringen av *fragmenten* baserat på storleken hos skärmen.

Man bör också tänka på att använda sig av element som inte påverkas nämnvärt av skillnad i upplösning och skärmdensitet, exempelvis *ListViews* och *ScrollViews*.

7.2 Metod att minska försämrad användarupplevelse på grund av låga uppkopplingshastigheter

Det går att ”gömma” laddningstider från användaren genom att låta all kommunikation mellan klient och server ske med en *cache*-fil som buffert. All data som hämtas från servern skrivs till filen, och all data som laddas in i klienten laddas från filen. Genom att använda en *cache*-fil så upplever användaren endast laddningstiderna som det tar att ladda från filen. Användning av *cache*-fil gör även att appen alltid har tillgång till den data som sist hämtades från servern.

I examensarbetet har *Lazy Load* studerats som möjlig metod för att minska mängden data som kommuniceras mellan klient och server. På grund av hur det befintliga systemet ser ut har dock detta inte kunnat implementeras och testas. Detta för att det skulle krävas en omfattande omstrukturering av

systemet samt ingrepp i hur kommunikationen mellan klient och server ser ut vilket inte har kunnat rymmas inom examensarbetets omfång.

Eftersom klasserna *SingleLeadsHandler* och *SingleStatisticsHandler* skapar objekt med informationen som skall visas på sidorna just när de skall visas, kan man rent principiellt säga att kommunikationen mellan datastrukturen och användargränssnittet använder sig av *Lazy Load*, om än i annan skala. Detta gör att det i teorin borde vara enkelt att flytta ut funktionaliteten från dessa båda klasser och låta servern ta hand om det arbetet. Det och att istället för att som i nuvarande *Leads API* skicka *JSON-objekt* med all information som finns i databasen istället skicka objekt med endast det som skall presenteras på sidan som visas för tillfället. Detta skulle givetvis behöva testas ordentligt då det kommer att innebära att flera uppslag mot databasen behöver göras, samt att servern behöver göra fler beräkningar. Med en användarbas på omkring 50 användare borde detta inte vara något problem.

8 Framtida utvecklingsmöjligheter

Då uppdelningen mellan Androidtelefoner och Iphones hos säljarna på Telavox är ungefär 50/50 så skulle det innebära att för att alla säljare skall ha tillgång till samma tjänst när de befinner sig i fält behöver appen *porteras* till *IOS*. Alternativt behöver en likadan app utvecklas även till *IOS*-plattformen vilket skulle vara resurskrävande vid utveckling, underhåll och uppdatering av applikationen.

Ett alternativ är istället att med grunden i samma innehåll och sidor som i appen som tagits fram i det här examensarbetet utveckla en app där användargränsnittet är webbaserat för att på det sättet göra appen mer plattformsoberoende.

För närvarande sker alla beräkningar av den information som skall visas på *dashboardet* på klientsidan och dessa beräkningarna behöver tillgång till all data som ligger i ***Leads***. Detta innebär att det för tillfället inte ger någon vinst i att använda *Lazy Load* för att hämta data från ***Leads***. Om istället alla beräkningar och sammanställning av data sker på servern skulle man kunna få ner mängden data som behöver finnas lokalt i appen avsevärt. Det finns många vinster med det, dels mindre att kommunicera över det mobila nätverket, dels bättre hantering av mobilenheters begränsade minne.

Ett förslag till vidareutveckling är alltså att flytta alla beräkningar av information som skall visas i listor och diagram flyttas från klienten till servern. Detta för att på det sättet minimera mängden data som behöver finnas tillgängligt lokalt i appen och skickas mellan *server* och *klient*.

9 Definitioner och terminologi

ADT	Android Development Tools, en samling verktyg för utveckling mot Androidenheter.
2G	Andra generationens mobiltelefoni teknik.
AbsoluteLayout	Sätt att beskriva en layout i en Layout fil till Android. Stöds ej längre av Android API.
ActionBar	Komponent i Android API, området längst upp på skärmen i en Androidenhet.
Activity	se aktivitet
ADT-plugin	Plugin till utvecklingsverktyget Eclipse. Möjliggör utveckling mot Androidenheter med Eclipse.
Aktivitet	En del i en android applikation som hanterar visning av användargränssnitt och i många fall är huvudprogrammet i en Anbdroid-app..
Android API	API för utveckling mot Androidenheter
Apache Licence 2.0	Open source license som medger gratis användning av kod både för icke kommersiellt och kommersiellt bruk.
API	Application programming interface.
Array	En samling värden eller objekt av samma typ i en indexerbar lista
Bitmap	En digital bild uppbyggd av en matris av punkter. Vid 100% skalning motsvarar varje punkt i matrisen en pixel. Se pixel.
Buffrad ström-läsare	bufferedStreamReader, en läsare av dataströmmar som innan behandling kan dela upp strömmen i mindre stycken för att minska mängden data som behöver hållas i minnet.
Cache	Mellanlagring av data.
Calendar	Java klass som används för att beskriva datum
Callback-metoder	Metoder som ett objekt kan kalla på hos objektets ägande objekt.

Collection	Samlingsnamn för en mängd klasser för hantering och lagring av data med vissa specifika gemensamma egenskaper.
Dashboard	Instrumentbräda, sida för visning och hantering av information.
Datamodell	modell över hur datastrukturen är uppbyggd i ett system.
Datastruktur	Struktur som bestämmer relationen av objekt och informationsflöden dem emellan.
Dataström	Ström av data och information med obestämd storlek och innehåll.
Dynamisk	I sammanhanget, att bestämma något vid körning av programmet.
Eclipse	IDE för utveckling av Java-applikationer. Går att utöka till andra programspråk med tillägsmoduler.
FrameLayout	En Layout i Android API som fungerar som en ram att stoppa andra Layout-element i.
Gameifiering	Sätt att försöka införa tävlingselement i ett arbete för att på så sätt öka motivation och produktivitet.
Git	Versionshanteringssystem
GitHub	Onlinetjänst för Git
GWT	Google Web Toolkit, verktyg för att skapa komplexa JavaScript-applikationer i programmeringsspråket Java.
GZIP	Format för komprimering av data.
Http	Hyper Text Transfer Protocol, transportprotokoll för kommunikation över internet.
Https	Krypterat HTTP
Hårdkodat	Inskrivet från början, går ej att ändra vid körning av programvaran.
IDE	Integrated Development Environment, utvecklingsverktyg.
Implementera	I sammanhanget, att en klass implementerar ett Interface betyder att klassen lovar att ha de metoder som finns beskrivna i Interfacet.

Instans	Ett objekt är en instans av en klass.
Instantiering	Handlingen att skapa ett objekt av en klass.
Intent	En klass i Android API som används för att skicka meddelanden mellan Aktiviteter, Mellan system och Applikationer samt mellan Applikationer.
Interface	En mall eller ritning som beskriver klasser av en speciell typ. En klass kan implementera ett Interface. Se *Implementera*.
IOS	Operativsystem som används i Iphones
Java	Objektorienterat Programmeringsspråk, används vid programmering av Androidenheter.
Klass	En beskrivning av objekt i objektorienterade programspråk.
Klassbibliotek	En samling klasser som på något sätt hänger ihop och bildar ett paket
Konstruktor	Speciell metod i en klass som anropas då ett nytt objekt skapas.
Lead	Från engelska; ledtråd, spår. En potentiell kund för en säljare på Telavox.
Leads	Webbgränssnitt som används av säljarna på Telavox.
Leads API	En samling fördefinierade metoder för kommunikation mellan Leads klienter, Web/Android och servern.
ListAdaptor	Komponent i Android API som används för att hantera data och information som skall visas i Listor i ett Android användargränssnitt.
ListFragment	Fragment som endast innehåller in lista.
Map	Datastruktur som sparar data i form av en nyckel och ett värde.
Metodanrop	Anrop till ett object för att utföra en förutbestämd operation. Sättet som objekt kommunicerar med varandra.
Metod	se metodanrop
Microsoft Excel	Microsofts kalkylprogram

Microsoft Sharepoint	Program från Microsoft för kollaborativt arbete och delning av information.
MS Outlook.	Microsofts Emailprogram
Navigationsmönster	Beprövade "allmänna" sätt att navigera mellan olika sidor på till exempel en webbsida eller i en mobilapp.
Null	Ingenting, ett objekt som är null innebär att objektet inte finns.
Number	I sammanhanget kan det vara ett nummer av typerna heltal (int), long (stort heltal), flyttal (float), decimaltal (double)
Objekt	En instans av en klassEn atomär enhet som är en instantiering av en klass.
Obuffrad dataström	En dataström som inte är uppdelad i mindre enheter.
onCreateView()	Metod hos ett Fragment i Android API, anropas alltid när ett fragment skapas. Här beskrivs till exempel vilken Layout-fil som skall användas när fragmentet visas.
Parser	Klass för tolkning av dokument av ett specifikt format exempelvis XML-parser och JSON-parser.
Partner	Telavox affärssystem.
PDF	Portable Document Format. Filformat för textdokument utvecklat av Adobe Systems.
Pixel	Kort för ”Picture Element”. En liten punkt som kan anta olika färger och som utgör den minsta beståndsdelen i hur en bild skapas på en skärm. Skärmars upplösning brukar anges i antalet pixlar i bredd och höjd, till exempel: 1024x768.
Portera	Översätta kod specifik för ett system eller språk till ett annat. Exempelvis från Android till IOS.
Projektmodell	En samling processer som används vid styrning av projekt.
Relationsdiagram	Diagram som beskriver delar i ett system och relationerna dem emellan.
Repository	I sammanhanget en förvaringsplats för kod som versionshanteras.

Runtime	När något inträffar under tiden en programvara körs sägs det hända vid runtime.
ScrollViews	Layoutkomponent i Android API som fungerar som en hållare för andra layouts. Gör innehållet som visas på skärmen scrollbart.
Session-cookie	Liten textfil som innehåller information om koppling mellan en klient och server vid kommunikation över internet.
SSL/TLS	Sätt att kryptera information vid kommunikation över internet.
Statiskt definierat	Bestämt vid kompilering
String	se *textsträng*
Subklass	Klass som ärver en annan klass inomobjektorienterad programmering.
Swipa	Sätt att interagera med en enhet med touchscreen. Dra med fingret över skärmen.
Textsträng	Dataformat som består av en lföljd tecken.
ToDo	En uppgift som skall utföras av en säljare, kan till exempel vara ett samtal att ringa.
Transparent	i den här kontexten, utvecklaren behöver inte ta hänsyn till.
Tråd	Genom att köra olika delar av en applikation i olika trådar kan man åstadkomma en illusion av parallell hantering av data.
UI	User Interface, användargränssnitt.
UI-tråd	Tråden som Användargränssnittet körs i. Se *tråd*
UML	Unified Modelling Language, ett standardiserat sätt att med enkla figurer beskriva strukturer och skeenden.
UML-sekvensdiagram	Diagram som beskriver en följd av händelser i UML-notation.
XML-Layout-fil	Fil i xml-format som används för att definiera utseende på det som visas på skärmen i en enhet.

10 Referenser

[1] ”Om Telavox,” Telavox, [Online]. Available: www.telavox.se/sv/om/telavox/. [Använd 12 08 2014].

[2] ”https://www.eclipse.org/,” Eclipse Foundation, 2014. [Online]. Available: https://www.eclipse.org/. [Använd 14 08 2014].

[3] ”ADT Plugin,” [Online]. Available: http://developer.android.com/tools/sdk/eclipse-adt.html. [Använd 14 08 2014].

[4] ”git --local-branching-on-the-cheap,” [Online]. Available: git-scm.com. [Använd 11 08 2014].

[5] ”GitHub,” [Online]. Available: https://github.com/. [Använd 09 08 2014].

[6] E. T. Bray, ”RFC7159,” Internet Engineering Task Force (IETF), Mars 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7159.txt. [Använd 10 juli 2014].

[7] R. F. e. al, ”RFC2616,” juni 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2616.txt. [Använd 10 juli 2014].

[8] T. P. S. Turner, ”RFC6176,” mars 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6176.txt. [Använd 10 juli 2014].

[9] E. G. e. al, ”Singleton,” i *Design Patterns, Elements of Reuseable Object-Oriented Software*, Indianapolis, Addison-Wesley, 1995, pp. 127-133.

[10] S. Lauesen, ”Focus groups,” i *Software Requirements, Styles and Tehniques*, Harlow, Pearson Education Limited, 2002, pp. 352-354.

[11] S. Lauesen, ”Interviewing,” i *Software Requirements, Styles and Techniques*, Harlow, Pearson Educational Limited, 2002, p. 339.

[12] A. Lantz, ”Att välja intervjuform,” i *Intervjumetodik, andra upplagan*, Annika Lantz och Studentlitteratur, 2007, pp. 29-35.

[13] Microsoft Corporation, ”Sharepoint,” Microsoft Corporation, 2014. [Online]. Available: http://office.microsoft.com/en-001/sharepoint/. [Använd 10 juli 2014].

[14] The Grizzly Labs, ”Genious Scan, A scanner in your pocket,” The Grizzly Labs, 2014. [Online]. Available: http://thegrizzlylabs.com/. [Använd 10 juli 2014].

[15] ”Navigation Drawer,” [Online]. Available: https://developer.android.com/design/patterns/navigation-drawer.html. [Använd 14 08 2014].

[16] Open Signal, ”fragmentation-2013,” [Online]. Available: http://opensignal.com/reports/fragmentation-2013/fragmentation-2013.pdf. [Använd 03 07 2014].

[17] ”opensignal.com,” Open Signal, [Online]. Available: opensignal.com. [Använd 12 08 2014].

[18] ”developer.android.com,” [Online]. Available: : http://developer.android.com/guide/practices/screens_support.html. [Använd 03 07 2014].

[19] ”developer.android.com,” [Online]. Available:

http://developer.android.com/guide/components/fragments.html. [Använd 03 07 2014].

[20] ”https://developer.android.com/design/patterns/navigation-drawer.html,” [Online]. Available: https://developer.android.com/design/patterns/navigation-drawer.html. [Använd 09 08 2014].

[21] M. F. e. al, Patterns of Enterprise Application Architecture, Boston, MA: Pearson Education, 2002.

[22] ”http://cometd.org/documentation,” dojo foundation, [Online]. Available: http://cometd.org/. [Använd 09 08 2014].

[23] A. R. e. al., ”http://docs.cometd.org/reference/bayeux.html,” dojo foundation, 2007. [Online]. Available: http://docs.cometd.org/reference/bayeux.html. [Använd 09 08 2014].

[24] S. L. e. al, ”Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP,” IETF, April 2011. [Online]. Available: tools.ietf.org/html/rfc6202. [Använd 11 08 2014].

[25] Ecma international, ”http://www.ecma-international.org/,” Oktober 2013. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. [Använd 08 08 2014].

[26] ”www.json.org,” [Online]. Available: www.json.org/java/index.html. [Använd 08 08 2014].

[27] W3Schools, ”XML DOM Tutorial,” [Online]. Available: http://www.w3schools.com/dom/default.asp. [Använd 08 08 2014].

[28] ”developer.android.com,” [Online]. Available: developer.android.com/reference/android/util/JsonReader.html. [Använd 08 08 2014].

[29] Oracle, ”Class HttpsURLConnection,” Oracle, [Online]. Available: docs.oracle.com/javase/7/docs/api/javax/net/ssl/HttpsURLConnection.html. [Använd 12 08 2014].

[30] M. A. Jean-Loup Gailly, ”the gzip homepage,” 27 6 2003. [Online]. Available: www.gzip.org. [Använd 12 08 2014].

[31] Apache Software Foundation, ”HttpClient Overview,” The Apache Software Foundation, 09 08 2014. [Online]. Available: hc.apache.org/httpcomponents-client-ga/. [Använd 12 08 2014].

[32] J. Wilson, ”Android's HTTP Clients,” 9 2011. [Online]. Available: android-developers-blogspot.se/2011/09/androids-http-clients.html. [Använd 12 08 2014].

[33] ”Google Cloud Messaging for Android,” [Online]. Available: developer.android.com/google/gcm/index.html. [Använd 12 08 2014].

[34] Google, ”Google Analytics,” Google, 15 juli 2014. [Online]. Available: http://developers.google.com/analytics. [Använd 11 08 2014].

[35] ”http://androidplot.com/,” [Online]. Available: http://androidplot.com/. [Använd 09 08 2014].

[36] ”developers.android.com,” [Online]. Available:

http://developer.android.com/reference/android/os/AsyncTask.html. [Använd 08 08 2014].

[37] ”http://developer.android.com/reference/android/widget/FrameLayout.html,” [Online]. Available: http://developer.android.com/reference/android/widget/FrameLayout.html. [Använd 09 08 2014].

[38] ”http://developer.android.com/reference/android/widget/ListAdapter.html,” [Online]. Available: http://developer.android.com/reference/android/widget/ListAdapter.html. [Använd 09 08 2014].