CHALMERS

PersonAlarm

Överfallsskydd för iPhone

Kandidatarbete inom Data- och Informationsteknik

WILLIAM GABRIELSSON
GUSTAV MÖRTBERG
SIMON OLSSON
MAX WITT

Chalmers tekniska högskola
Göteborgs universitet
Institutionen för Data- och Informationsteknik
Göteborg, Sverige, Juni 2013

PersonAlarm
Mobile assault alarm for iPhone.

WILLIAM GABRIELSSON
GUSTAV MÖRTBERG
SIMON OLSSON
MAX WITT

Examiner: ARNE LINDE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Förord

Detta kandidatarbete har bedrivits under våren 2013 och är ett arbete vid instutitionen för data- och informationsteknik, Chalmers Tekniska Högskola. Kandidatgruppen önskar tacka alla inblandade parter för fint samarbete och vägledning under vårens gång. Speciellt vill gruppen utmärka och tacka vår handledare Christer Carlsson.

Abstract

The presented thesis seeks to clarify the development of PersonAlarm. PersonAlarm is an application built for the iOS platform. In today's society there is an unfortunate fear of walking home alone or visiting particular areas. Therefore there is a growing need of systems, tools or aids to reduce this fear. This is where PersonAlarm enters the picture and attempts to diminish this fear.

The application gives the user a way of inviting friends to 'follow' his or her way home through their iPhones. Thus reducing the fear of being alone in certain situations.

The development and need of the application was based off the feedback received from a field survey. User tests have also proven the app necessary and useful.

This thesis describes the development of this application from pre-studies to its conclusion and discussion. The bachelors group came across several obstacles and these are presented within the thesis.

Sammanfattning

Denna rapport beskriver utvecklingsarbetet av iOS-applikationen PersonAlarm.

Många människor känner en rädsla och oro för att utsättas för våld och övergrepp när de till exempel vistas ensamma utomhus på kvällar och nätter. Eftersom ca. 86 % av den vuxna befolkningen i Sverige äger en smartphone, kan en mobilapplikation utvecklas som ökar tryggheten för dessa människor och minskar risken för att de faktiskt utsätts för övergrepp.

Målet med PersonAlarm var att utveckla en sådan applikation för operativsystemet iOS. Stor vikt har lags på att applikationen skall vara intuitiv och användarvänlig.

PersonAlarm ger användare en möjlighet att bjuda in vänner att följa dennes position på en karta som visas i vännernas iPhones. På så sätt minskas oron och rädslan av att vara ensam i vissa situationer. Vidare finns möjlighet att snabbt påkalla hjälp i en överfallssituation.

Behovet och utvecklingen av PersonAlarm är baserad på den respons som erhölls från en enkätundersökning, respektive den feedback som mottogs via användartester. Detta underlag har lagt grunden för den första fungerande versionen av PersonAlarm.

Ordlista

API	*Application Programming Interface,* gränssnittet mellan ett externt system/ramverk och en applikation som använder systemet. Inkluderar ofta dokumentation och beskrivning för hur det bör användas.
Applikation, App	Spel eller program som körs på en (i vårt fall) smartphone.
App Store	Apples egna marknadsplats för distribuering av mobila applikationer.
IDE	*Integrated Development Environment,* en generell beteckning för ett verktygsprogram som underlättar programmeringen.
iOS	Operativsystemet som används i iPhone.
Objective-C	Programspråket som används vid utveckling av applikationer till iPhone.
Ramverk	Ett paket färdigskrivna klasser utvecklare kan använda sig av.
Release	En fungerande version av applikationen.
Repository	Projektets arbetsyta inom ramen för ett versionshanteringssystem. I teknisk mening den översta mapp som versionshanteringssystemet betraktar som en enskild enhet.
Session	Den tid mellan att en användare startar larmet tills att han eller hon avslutar det.
Smartphone	En mobiltelefon med internetåtkomst och möjligheten att exekvera applikationer.
Sprint	Iterationscykel som inkluderar planering, utveckling och testning av en specifik del av systemet. Sträcker sig i vanliga fall över en vecka till en månad, men kan även vara längre än så.
Vänlista	En lista innehållande en användares vänner.

Innehåll

1 Inledning

På brottsrummets hemsida kan man läsa om hur brottsantalet har ökat kraftigt de senaste tio åren [Bro12]. Däribland finns brott mot personers kroppsliga integritet, det vill säga överfall i form av misshandel, våldtäkter med mera. Från försäkringsbolaget If's hemsida ser vi citatet:

> "I Sverige anmäls ca 15 våldtäkter om dagen och antalet våldsbrott ökar stadigt" [If12]

En undersökning gjord av Statistiska centralbyrån visar att många människor ofta känner sig otrygga vid utevistelse sent på kvällen [Cen12]. Kombinerat med att ca. 86 % av vuxna svenskar, 15-64 år, har en smartphone [Flu12] vore det eftertraktat med en mobil lösning som bidrar till en ökad trygghetskänsla. Vårt lösningsförslag är en teknisk implementation i form av en mobilapplikation. Applikationen ökar chansen för användare att snabbt få hjälp vid ett överfall eller annan incident. Detta genom att dela med sig relevant information som exempelvis kartposition, ljudinspelningar och vem som har utsatts för överfallet till ett par förvalda personer eller om möjligt, en larmcentral av något slag.

Målgruppen för applikationen är de överlappande mängderna mellan personer där majoriteten av oron för överfall finns, vilket är kvinnor mellan 16 - 80 år [Cen12], samt majoriteten av smartphoneanvändare, vilket är personer i åldersgruppen 15 - 45 år [Nie13].

På marknaden finns redan ett antal mobilapplikationer för överfallsalarm och skydd att välja mellan, men de kostar oftast pengar [Pre13a]. Vi tror att prislappen gör att många avstår från att införskaffa ett överfallsskydd, därför är vår applikation kostnadsfri.

1.1 Syfte

Syftet med detta projekt är att skapa en intuitiv, effektiv och funktionell applikation till iOS-baserade enheter. Målet med applikationen är att tillhandahålla möjligheten att utlösa larm i överfallssituationer.

Fokus lades på att applikationen skall vara användarvänlig och responsiv, detta för att skapa så stor motivation för användning som möjligt. Målet är att användaren ska kunna förmedla så mycket information som möjligt om ett, eventuellt, pågående överfall.

En målsättning med projektet är att utveckla applikationen PersonAlarm till ett stadie som är tillräckligt för att applikationen ska kunna distribueras till marknaden. Ytterligare en målsättning med projektet är att det skall generera mycket kunskap och erfarenhet för projektgruppens medlemmar.

1.2 Problembeskrivning

Två huvudsakliga delproblem gruppen stod inför var hur larmet skulle utlösas och vad som ska hända efter att ett larm har utlösts. Gruppen övervägde en mängd olika metoder och lösningar på nämnda problem och fann både för- och nackdelar med dem. Efter research, enkätundersökning och användartestning bestämdes de metoder som fungerar bäst för vår applikation.

Apple, vilket är det företag som skapat iPhonen, har speciella krav för att få distribuera applikationer på App Store, ett exempel är att det grafiska användargränssnittet måste följa vissa riktlinjer [DL12d]. Apple har även satt upp regler för vad applikationer får och inte få göra, under utvecklingen har vi därav strävat efter att följa dessa.

1.3 Rapportens upplägg

I kapitel 2 presenteras den tekniska bakgrund som är nödvändig för att ge läsaren en förståelse för utveckling av iOS-applikationer. En kort beskrivning av Objective-C och strukturen i iOS ges. Vidare förklaras de essentiella verktyg som används vid iOS-utveckling. Därefter beskrivs de vanligaste och viktigaste delarna som agerar byggstenar i iOS-applikationer. Sist ges en redogörelse över webbutvecklingen och vilka ramverk som använts där.

Kapitel 3 presenterar de metoder gruppen använt under projektets gång och ansvarsfördelningen förklaras. Vidare förklaras hur insamlingen av information skötts. Sist ges en beskrivning av det versionshanteringssystem gruppen använt.

Gruppen har fastställt krav som ska gälla för applikationen och dessa presenteras i kapitel 4. Här finns förklaringar av hur kraven indelats och hur de strukturerades. Även hur gruppen valt de krav som definieras förklaras i detta kapitel.

Den grafiska designen presenteras i kapitel 5. Här ges motivering till de designval gruppen gjort. Även psykologiska aspekter av interaktionsdesign tas upp, samt varför viss design lämpar sig väl för applikationens syfte.

I kapitel 6 beskrivs de funktioner som är implementerade i mobilapplikationen. Här presenteras och förklaras applikationens funktioner i detalj tillsammans med korta tekniska beskrivningar.

I de resterande kapitlen ges projektets slutgiltiga resultat och diskussioner kring gruppens arbete. Vidare ges förslag och diskussion kring möjlig fortsatt utveckling av applikationen samt en redogörelse över gruppens slutsatser.

2 Teknisk beskrivning och bakgrund

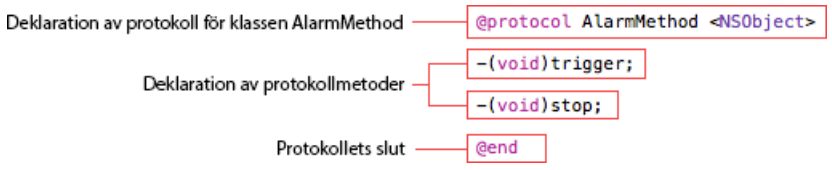
Utveckling av iOS-applikationer kräver en viss teknisk bakgrund vilket läsaren av rapporten ej kan förutsättas besitta. För att delge den information och bakgrund som krävs för att hantera denna rapport ges följande avsnitt som teknisk grund.

2.1 Objective-C

Objective-C är det programspråk Apple uppmanar utvecklare att skriva iOS-applikationer i [Lib12]. Objective-C är en utökning av C och stödjer objektorienterad programmering. Det är ett språk som till viss del liknar Java och har blivit väldigt populärt bland utvecklare [TIO13]. Popularitet har bidragit till att det finns mycket information att ta del av på forum och diskussionsgrupper online.

• Implementation (.m) och header (.h)
Objective-C är en utökning av programspråket C, detta innebär att även Objective-C använder implementations- och header-filer. Varje klass består av två filer med samma namn och ändelserna .h respektive .m. Det vill säga att om en användare väljer att skapa en klass med namnet MyClass, genereras två filer: MyClass.m samt MyClass.h. Filen som slutar med .m kallas för implementationsfilen, och det är här all implementation av klassen definieras. Implemenationsfilen är inte synlig för andra klasser, det är här privata instansmetoder och variabler deklareras. Man får även en fil som slutar på .h, detta är interfacet där man kan deklarera protokoll, metoder och fält som ska vara synliga för andra klasser.

• Protokoll och *delegation*
Inom objektorienterade språk är interface ett vanligt fenomen, motsvarigheten inom Objective-C är protokoll och delegering. Protokoll och delegering är viktiga principer som används mycket inom Objective-C. Som exempel finns en klass i vårt projekt vars uppgift är att på något sätt avfyra larmet. Det finns flera larmmetoder, exempelvis att ringa ett samtal, skicka ett nätverksmeddelande eller spela ett ljud. Avfyrarklassen vet inte förrän programmet körs vilken metod som skall användas, därför kräver den endast att en *delegate* tillsätts, vilket är ett objekt som uppfyller protokollet AlarmMethod, se figur 2.1.



Figur 2.1: Utdrag från applikationskoden som illustrerar användandet av protokoll.

Klassen kan då anropa metoderna trigger och stop och med säkerhet veta att något

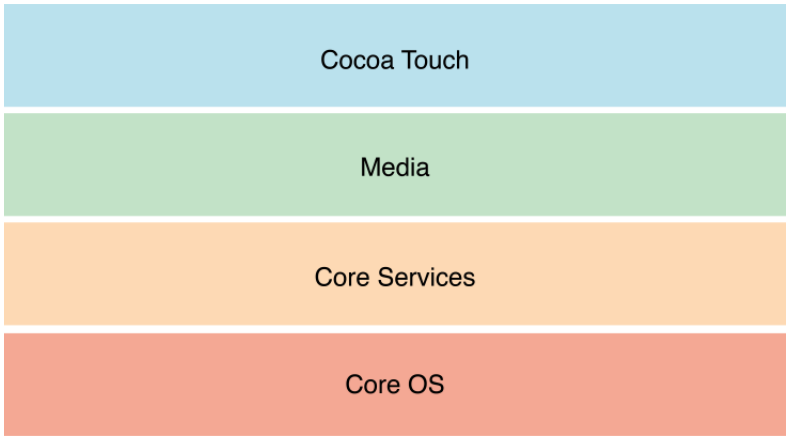
händer. De olika larmmetodklasserna följer protokollet och implementerar de specificerade metoderna. Vid körning ser klassen som instansierar avfyrarklassen till att en av larmklasserna sätts som *delegate*, ofta efter användarens önskemål.

2.2 iOS

Utveckling av iPhone-applikationer förutsätter att de ramverk och kodbibliotek som finns i iOS används. I följande sektioner ges en grundläggande beskrivning av iOS, de verktyg som finns att tillgå vid utveckling av iOS samt de kodbibliotek som använts i projektet.

2.2.1 Lager i iOS

iOS släpptes 2007 av Apple Inc [Inf07] och används i flera produkter, till exempel iPhone, iPod och iPad. iOS är uppbyggt av fyra lager, vilka visualiseras nedan i figur 2.2.



Figur 2.2: Bilden illustrerar iOS fyra lager [DL12a].

• **Core OS**
Core OS är lagret närmast hårdvaran och innehåller ramverk som de flesta övriga teknologier inom iOS bygger på. Ett exempel på ramverk i Core OS är System, som tillhandahåller funktioner för bland annat trådning, nätverk, standard I/O och minnesallokering [DL12b].

• **Core Services**
I Core Services finns viktiga funktioner som alla applikationer använder. Några exempel på detta är automatisk referenshantering, se avsnitt 2.2.6, samt betalningar som sker inuti applikationer [DL12c].

• **Media**
Media-lagrets uppgift är att styra allt som har med grafik, ljud och video att göra [DL12e]. Exempel på ramverk i detta lager är Core Audio, Core Video och Core Graphics.

• **Cocoa Touch**
I Cocoa Touch-lagret finns funktioner och teknologier som ligger till grund för att utveckla applikationer till iOS [DL12f]. Detta är funktioner som *multitasking* och touch-input. Även verktyg för att bygga det grafiska gränssnittet finns i Cocoa Touch. Det är detta lager utvecklare använder sig mest av och som tillhandahåller de funktioner som är vanligast att träffa på i applikationskod.

2.2.2 iOS Developer Library

iOS Developer Library beskriver i detalj allt från hur utvecklingsmiljön och hur projekt startas upp till detaljerade instruktioner av hur kod ska skrivas enligt Apples riktlinjer. Det följer även med beskrivningar om hur utvecklare får åtkomst till de bibliotek som inkluderas i iOS SDK. Vidare finns många guider och exempelprojekt där utvecklare kan få insyn i hur man kan gå tillväga för att lösa diverse problem.

2.2.3 Plattform

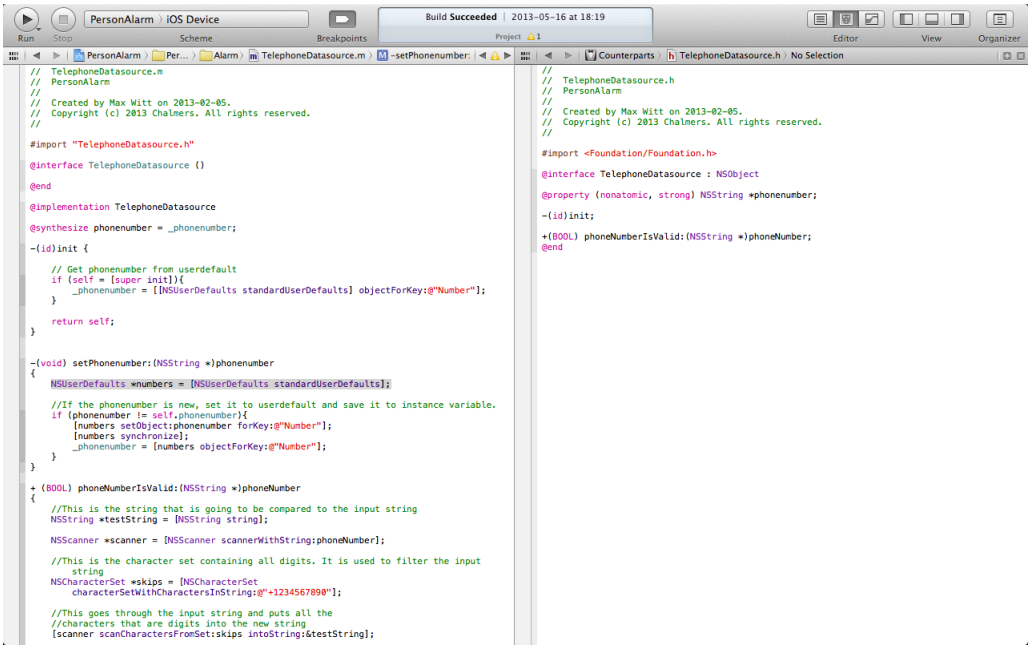
En plattform definieras av kombinationen hårdvaruenhet och operativsystem. I vårt fall är det iPhone och iOS som bygger upp plattformen. Valet av plattform förenklar arbetet för utvecklaren då appens resurser och funktioner kan avgränsas och applikationens källkod blir mindre komplex. PersonAlarm kommer att distribueras och köras på en iPhone som kör iOS 6. Motivet till beslutet är att iOS 6 är den senaste versionen av iOS. Som utvecklare behöver vi inte oroa oss över att funktioner inte har stöd på vissa telefoner, eftersom Apple endast låter kompatibla enheter använda iOS av en specifik version.

2.2.4 Verktyg

Vid utveckling av applikationer till iOS finns ett flertal verktyg att tillgå för utvecklare. De två mest relevanta att nämna är XCode (den integrerade utvecklingsmiljön) och iOS SDK. Dessa verktyg har gjort det möjligt att bygga applikationen samt underlättat utvecklingsarbetet.

• **XCode 4**
XCode är den utvecklingsmiljö som krävs för utveckling av applikationer till iOS och Mac OSX, vi använder senaste versionen av XCode - version 4. I XCode finns grundläggande funktioner såsom kompilator, debugger, code completion, samt en inbyggd

editor för att skapa grafiska gränssnitt. Till XCode finns det även ett inbyggt plugin, Instruments, där utvecklare kan analysera prestanda för att upptäcka exempelvis minnesläckor i sina applikationer. XCode har en mängd avancerade funktioner men trots detta är både design och gränssnitt enkla och intuitiva. Programmering i Objective-C innebär att man ofta arbetar samtidigt med både .m och .h-filen som tillhör samma klass. XCode har en inbyggd funktion för att visa två olika filer i separata textfönster och en annan funktion som automatiskt öppnar motsvarigheten till filen som utvecklaren valt. Detta illustreras nedan i figur 2.3 där .m-filen visas i vänster kolumn och .h-filen i den högra.



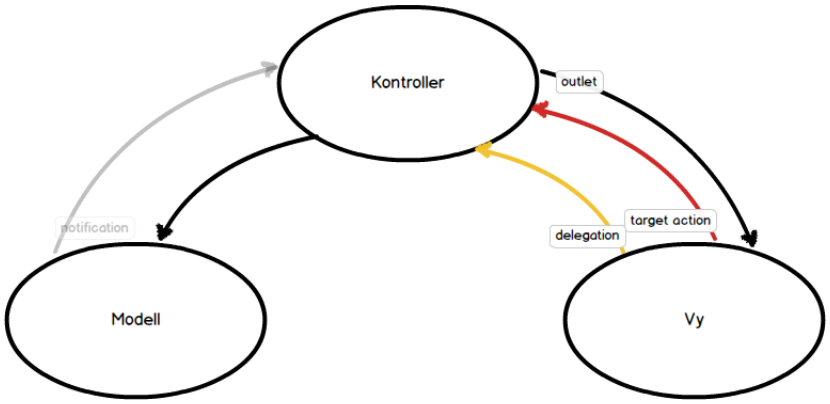
Figur 2.3: Tvådelad vy i XCode.

• **iOS SDK**
iOS SDK (Software Development Kit) är ett paket med utvecklingsverktyg som ger tredjepartsutvecklare möjlighet att utveckla applikationer för iOS-plattformar. Detta paket innehåller de ramverk som presenterades i avsnitt 2.2.1. Ramverken låter utvecklaren nyttja de hårdvaru- och mjukvaruresurser som plattformen erbjuder, exempelvis hårdvaruenheter som GPS och kamera, eller mjukvaruresurser som databaser och generella nätverksanslutningar som i sin tur använder trådlösa nätverk eller internetanslutning över telefonnätet.

2.2.5 Model-View-Controller

MVC (*Model-View-Controller*)-mönstret är den övergripande designprincip som används och förespråkas vid utveckling av iOS-applikationer. Användandet av MVC leder till

kodindelning som är enkel att förstå. En illustration av hur MVC ser ut i iOS illustreras nedan i figur 2.4.



Figur 2.4: MVC i iOS

MVC är en mjukvaruarkitektur som delar in kod i tre lager, modell, vy och kontroller och tanken är att detta skall separera användares interaktion från den bakomliggande modellen. En kort beskrivning av dessa lager ges nedan.

- **Modell**

Modellen är datan som applikationen visar eller använder. Modellen känner inte till kontrollern eller vyn. När dess tillstånd ändrats finns det möjlighet för modellen att skicka en *notification* till kontrollern.

- **Kontroller**

Kontrollern styr programmet i den mening att den har alla rättigheter till att manipulera modellen och vyer. Kopplingen mellan vy och kontroller benämns som outlet. Genom kopplingen kan kontrollern avläsa och uppdatera grafiska komponenter såsom textfält och knappar med data.

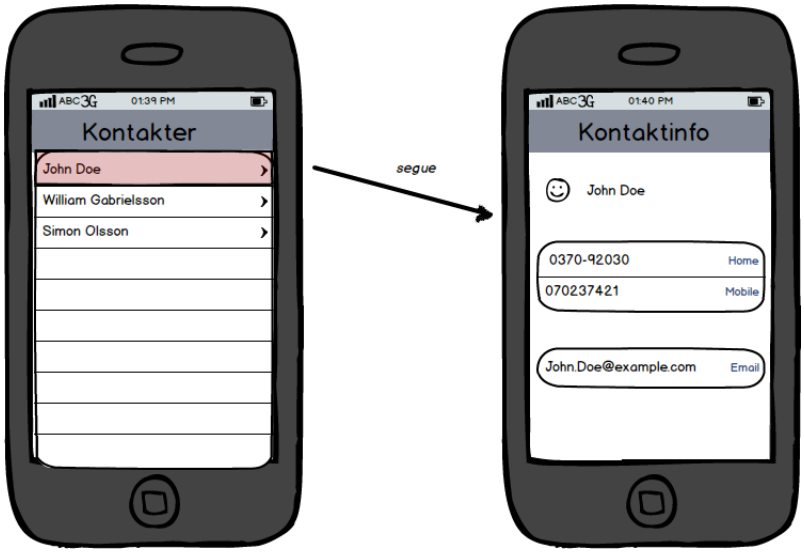
- **Vy**

En vy visar data för användaren, den känner inte till kontrollern eller modellen och ska inte innehålla egen data. För att kunna meddela kontrollern när en händelse sker kan två olika tekniker användas: *target action* eller *delegation*. I *target action* sätter kontrollern sig själv som det mål till vilken vyn skickar händelser när exempelvis en knapptryckning sker. Detta är enkelt att åstadkomma i XCode 4 med hjälp av inbyggda funktioner. I *delegation* har vyn specifika metoder där ansvaret för implementeringen lämnas över till kontrollern. Kontrollern benämns då som en *delegate*. För mer information om *delegation*, se avsnitt 2.1.

2.2.6 Byggstenar i en iOS-Applikation

Koden som utgör iOS-applikationen karaktäriseras dels av syntaxen i Objective-C, dels av de designmönster och konventioner som iOS SDK förordar.

• ***View Controller***
*View Controller*s är en viktig del av iOS eftersom de innehåller logiken för hur data skall presenteras. En *View Controller* är en behållare för andra komponenter, såsom knappar, textfält och listor. En app byggs upp av ett antal *View Controller*s som arbetar och kommunicerar med varandra. Ett exempel på detta illustreras i figur 2.5.



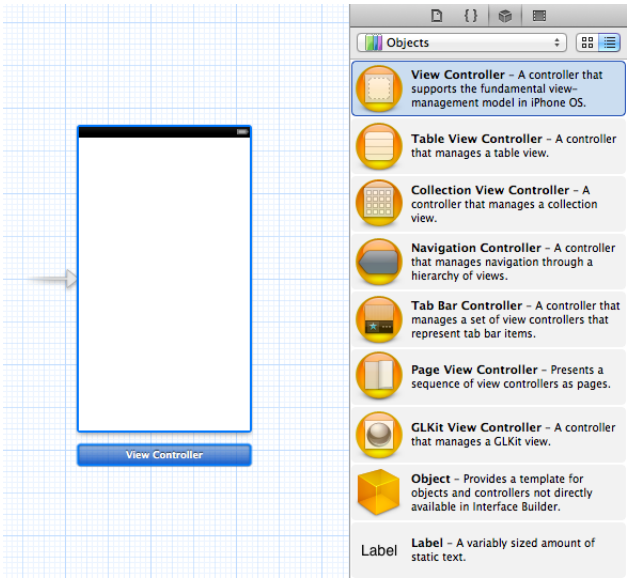
Figur 2.5: Illustration av *View Controller*s och en segue.

Till vänster visas en komponent innehållande en lista vilken styrs av en *View Controller*. *View Controller*n har i uppgift att presentera alla kontakter. En annan *View Controller* styr den högra helskärmsvyn, innehållande knappar och textfält och har i uppgift att presentera detaljerad kontaktinformation. En övergång mellan olika *View Controller*s benämns som segue.

• ***Storyboard***
I XCode innan version 4 anropade *View Controller*s varandra i kod när det var dags för en annan *View Controller* att presentera data. För att inte *View Controller*s skulle skapa cirkelreferenser och därmed orsaka minnesläckor, användes en stack-struktur där den aktiva *View Controller*n låg överst. För att gå vidare i navigeringen lades en *View Controller* överst, som då blev aktiv. För att backa i navigeringen togs den aktiva *View Controller*n bort från stacken och den underliggande *View Controller*n blev då aktiv.

En av de nya funktionerna i XCode 4 var att utvecklaren istället kunde designa navigationen mellan helskärmsvyer i samma verktyg som används för design av grafiska

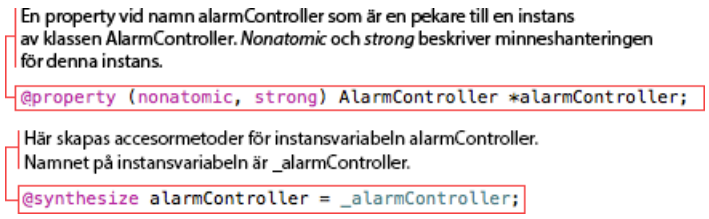
gränssnitt, genom att bygga s.k. *storyboard*s. Stack-strukturen behölls som ett underliggande designmönster, varje *View Controller* representeras av en helskärmsruta på arbetsytan och navigeringshierarkin beskrivs med grafiska pilar som kallas segues.



Figur 2.6: Bilden visar en generisk *storyboard* med en tillhörande sidovy.

Den *storyboard* som visas i figur 2.6 innehåller en generisk *View Controller*. Till höger i figur 2.6 finns en sidovy, i denna visas de färdigdefinierade komponenterna som är möjliga att lägga till i en *storyboard*. Trots att verktyget presenterar en helhetsbild av applikationen, är det här som detaljerna i det grafiska gränssnittet definieras. För att finjustera komponenters placering och attribut kan utvecklaren zooma i arbetsytan, samt ändra objektens egenskaper med reglage i verktygsfältet till höger.

• ***Properties* och *synthesize***
Alla objekt i Objective-C ligger i en heap och åtkomst ges genom pekare. En *property* definierar hur minneshanteringen av instansvariablen ska se ut. Nyckelordet *synthesize* skapar sedan accessormetoder för en *property*. Dessa accesormetoder varierar beroende på hur en *property* definierats.



Figur 2.7: Utdrag från applikationskoden som visar *properties* och *synthesize*.

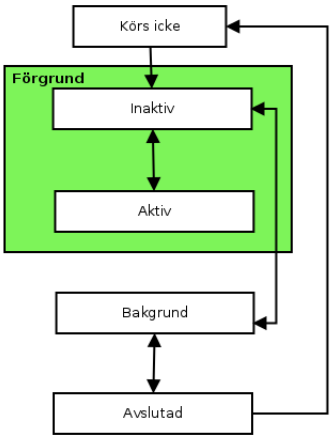
I figur 2.7 ses en *property* vid namn alarmController samt dess *synthesize*. I Java hade detta motsvarat en instansvariabel med set- och get-metoder.

• **Automatisk referenshantering (ARC)**
Eftersom alla objekt i Objective-C ligger i heapen, måste någon form av städning av heapen ske för att undvika minnesläckor och diverse störningar. I Java och många andra programmeringsspråk används garbage collection, det vill säga att vid vissa tidpunkter genomsöks heapen efter objekt som kan tas bort. I Objective-C finns ingen garbage collection, istället används automatisk referenshantering (Automatic Reference Counting, ARC). Varje objekt förknippas med en räknare som håller reda på antalet referenser till objektet. När inga referenser finns kvar tas objektet bort.

• ***Multitasking***
Multitasking infördes i samband med introduktionen av iOS 4. Detta innebär att applikationer kan fortsätta att köras trots att användaren inte längre har applikationen öppen. Appens resurser och uppförande skiljer sig beroende på vilket tillstånd den befinner sig i [DL13]. De tillstånd en app kan befinna sig i visas i tabell 2.1 och hur de är relaterade till varandra ses i figur 2.8.

Tabell 2.1: En iPhone-applikations olika tillstånd

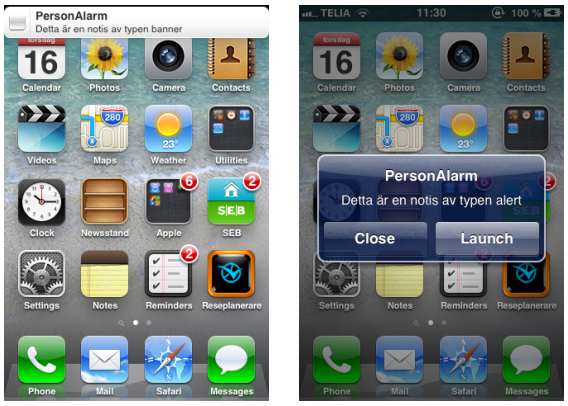
Tillsånd	Beskrivning
Körs icke	Appen har inte startats eller har blivit avslutad av systemet
Inaktiv	Appen körs, men den kan inte ta emot händelser. Applikationen är alltså låst för användaren i detta läge. Detta tillstånd är kortvarigt och sker oftast när appen ändras från ett tillstånd till ett annat.
Aktiv	Appen körs och kan ta emot samt svara på händelser.
Bakgrund	Appen körs i bakgrunden, det vill säga att appen fortfarande exekverar kod trots att applikationen inte är i förgrunden. Exempelvis kan användaren ha öppnat en annan app.
Avslutad	Appen är inte i förgrunden och exekverar ingen kod. Appen finns fortfarande i minnet och kan rensas bort av systemet då minne behövs av en annan app. Den övergår då till tillståndet Körs icke.



Figur 2.8: Tillståndsövergångarna en app genomgår.

Ett av Apples krav för få att distribuera en app på App Store är att appen ska kunna anpassa sig själv efter dessa tillstånd. Motivet är att det annars kan ske förlust av data, samt att användarupplevelsen påverkas negativt [DL13].

• ***Push Notification***
Push Notification är en tjänst som Apple erbjuder, som gör det möjligt att skicka *notifications* från en webbserver till enheter som kör iOS eller Mac OSX. Med hjälp av denna tjänst kan en aktiv app ta emot och hantera händelser. Om appen inte är aktiv tar enheten emot händelsen och presenterar den för användaren i form av en notis. En notis kan vara av två olika typer, banner och alert, dessa typer illustreras i figur 2.9. Användaren ställer själv in vilken typ av notis som ska användas. Det är även möjligt att att stänga av notiserna.



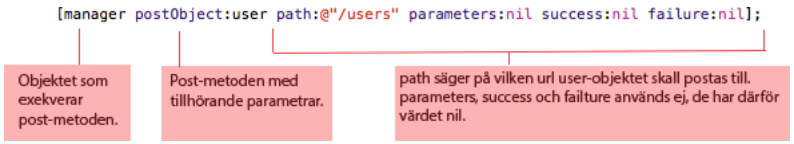
(a) (b)

Figur 2.9: Illustration av olika typer av notiser.

2.3 Ramverk

Förutom de kodbibliotek som följer med iOS SDK, har vi även undersökt och använt oss av ramverk och bibliotek från tredje part. Här följer en beskrivning av dessa ramverk.

• **RestKit**
RestKit är ett ramverk för iOS som ger möjlighet för kommunikation med webbtjänster som följer REST-mönstret [Res13]. RestKit ger utvecklaren, på ett smidigt sätt, möjligheten att koppla upp sig mot webbtjänster. RestKits främsta mål är dock att få utvecklare att tänka mer i termer av applikationens datamodell än hur meddelanden skickas, hur svar ska analyseras och hur objekt skall byggas om. RestKit är till för att underlätta arbetet för utvecklaren och ett exempel på dess användarvänlighet ges i figur 2.10, där illustreras HTTP post.



Figur 2.10: Post av ett User-objekt till webbtjänsten.

REST (REpresentational State Transfer) är en modell för att strukturera och överföra information som normalt existerar i en objektorienterad modell över HTTP-protokollet och dess fyra metoder: POST, PUT, GET och DELETE. REST används främst för att överföra datamodellen från en webbtjänst till en webapplikation eller en klientapplikation [Elk08]. REST refererar till modellobjekt som resurser, exempelvis användare och vänförfrågningar. Resurser ordnas i en hierarkisk struktur som påminner om ett filsystem och dess sökvägar, eftersom applikationen får tillgång till resurser genom web-adresser. Skillnaden är att olika sökvägar kan resultera i samma resurs, beroende på hur resurser relaterar till varandra. Låt säga att klienten vill finna alla sessioner som tillhör användaren med ID 3, och därefter lista alla sessioner, då hade resurserna hittats genom följande adresser:

http://example.com/users/3/sessions
http://example.com/sessions

Båda adresserna pekar på mängder av sessioner, delvis med samma innehåll. Strukturen för hur modellen är organiserad är skräddarsydd för varje webbtjänst och därmed kan strukturen för en okänd webbtjänst inte förutses. Dock finns ett par riktlinjer om hur resurser bör namnges och hur relationer mellan objekt beskrivs i sökvägar.

Resurserna manipuleras med hjälp av HTTP-metoderna POST, PUT, GET och DELETE. Vad varje metod utför är specificerat i REST, eftersom samma fyra metoder används för alla typer av resurser är de väldigt generella. GET hämtar alltid en resurs eller en mängd resurser utan att påverka dem, POST skapar en ny resurs, PUT uppdaterar en vald resurs med ny information eller ersätter en mängd resurser med en ny mängd, DELETE raderar resursen eller mängden.

• **Parse SDK**

Parse SDK är ett ramverk för iOS som tillhandahåller molnbaserad datalagring. Ramverket använder sig av en färdigskriven webbtjänst som ansvarar för databasen. Objekten i databasen samt dess relationer kan bestämmas genom Objective-C kod, därmed kan utvecklare konfigurera webbtjänsten i applikationskod direkt i iOS-applikationen, utan att själv behöva bygga en separat webbtjänst med motsvarnde datamodell. Figur 2.11 visar hur ett objekt kan definieras och sparas i databasen.

Ramverket ser till att alla modellobjekt som ska sparas centralt struktureras om till ett serialiserat format och skickas upp till servern utan att utvecklaren behöver sätta sig in i något annat än Parses egna API.

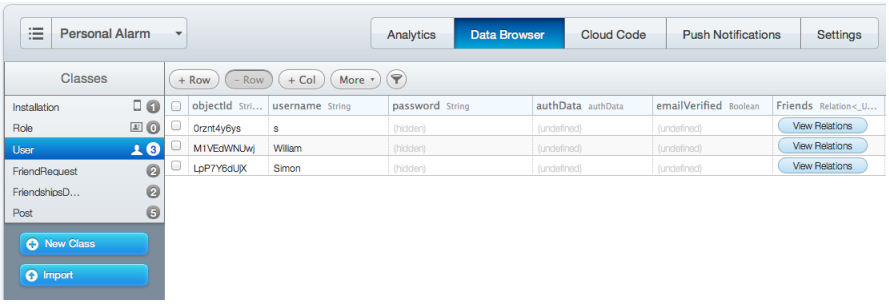
```
//Skapa ett Parse objekt av typen Post.
PFObject* post = [PFObject objectWithClassName:@"Post"];

//Sätt "Hello World" som titel.
[post setObject:@"Hello world" forKey:@"title"];

//Spara objektet på den molnbaserade databasen
[post saveInBackground];
```

Figur 2.11: Ett objekt av klassen Post skapas och sparas i den molnbaserade databasen.

Parse SDK låter utvecklaren inspektera och modifiera databasen genom ett gränssnitt på Parses egna hemsida[Par13]. I figur 2.12 presenteras ett objekt som har lagts till i databasen. Objektets attribut visas i kolumnerna. En rad motsvarar ett objekt med tillhörande attributvärden. Alla objekt i databasen har attributen “objectId”, “createdAt” samt “updatedAt”, vilka bestäms av Parse.



Figur 2.12: Inspektion av en databas skapad genom Parse SDK.

Genom Parses webapplikation kan utvecklaren även styra funktioner som *Push Notification*, vilket är väldigt användbart för att uppdatera användare om att det finns ny information hos webbtjänsten, som exempelvis i vår applikation där en GPS-position kan skickas mellan användare via webbtjänsten.

3 Metod

Detta kapitel beskriver hur gruppen har genomfört projektet, samt hur gruppen har strukturerat och fördelat arbetet tas upp. Även insamling av information och data samt olika tekniska lösningar och utvecklingsmetoder som har underlättat genomförandet av projektet presenteras.

3.1 Ansvarsuppdelning

Ett flertal roller har definierats med syfte att strukturera upp arbetsprocessen och underlätta ansvarsuppdelningen. Att varje gruppmedlem har ett specifikt ansvarsområde innebär att gruppmedlemmen skall ta ansvar för att arbetet kommer att utföras av någon i gruppen.

• **Gruppledare**
Gruppledaren har ansvar för att kalla till gemensamma möten, sköta den löpande kontakten med handledare samt ha uppsikt över att samtliga gruppmedlemmar sköter sin del av arbetet.

• **Utvecklarsupport/Agil administratör**
Utvecklarsupport innebär att se till att alla kringsystem (GitHub, Apples utvecklarlicenser) underhålls. Samt att dessa system eller motsvarande tjänster finns tillgängliga för gruppen. Den agila administratören ser till att sprintplanering utförs och att GitHub används. Dessa två ansvarsroller överlappar varandra, vilket gör det naturligt att låta samma person ta båda rollerna.

• **Produktansvariga**
Produktansvarig har sista ordet för kod- och designdetaljer, ansvarar för funktionsprioritering vid sprintplannering samt vidareutveckling av produkten. I detta projektet har vi två produkter och därmed två produktansvariga. iPhone-appen är den största och viktigaste, den tillhörande webbtjänsten är en mindre produkt.

• **Dokumentationsansvarig**
Dokumentationsansvarig har ansvar för att arbetet med slutrapport och andra inlämningar sköts kontinuerligt under arbetets gång samt att deadlines till dessa följs.

3.2 Utvecklingsmetodik

Utvecklingsarbetet har präglats av ett iterativt tankesätt, där gruppen utvecklat i korta sprinter. Detta eftersom gruppen i ett tidigt skede insåg att specifikationerna för projektet skulle komma att förändras allt eftersom respons erhålles från användartester

och gruppen lär sig mer om utvecklingsplattformen. Eftersom projektets totala tidsspann är relativt kort valdes en sprintcykel på en vecka. Korta sprintar gör det lättare att kontrollera arbetet, men mindre mängd arbete utförs mellan *releases*.

3.2.1 Agil utveckling med SCRUM

Gruppen har baserat utvecklingsmetodiken på en teori som kallas SCRUM, vilket har sina rötter i ett managementverktyg från mitten av 1980-talet som under 90-talet anpassades för att användas vid mjukvaruutveckling [Sut11]. Metoden bygger på att en utvecklingsgrupp skall ha en tydlig bild av vad som har gjorts och vad som behöver göras i varje del av utvecklingsfasen, detta för att effektivt öka produktiviteten. Två av de vitala byggstenarna är väl definierade projektroller samt införandet av kortare utvecklingssprinter.

Projektrollerna består av tre stycken huvudroller samt ett par stödroller. Huvudrollerna är produktägare, utvecklingsteam och ScrumMaster. Produktägaren har ansvar för att företräda kunden och se till att kundens intressen representeras väl i utvecklingsarbetet. Produktägaren arbetar bland annat med att skapa användarhistorier och att prioritera dessa. Utvecklingsteamet har som ansvar att sköta själva utvecklingen av produkten, vilket innebär att det skall finnas en produkt att leverera i slutet av varje sprintcykel. ScrumMasterns uppgift är att överse att utvecklingsgruppen nyttjar systemet på ett korrekt sätt.

3.3 Insamling av data

Applikationen förlitar sig till stor del på kvalitetssäkring genom användarrespons. Eftersom produkten inte är affärskritisk eller förlorar värde av att en konkurrent kopierar applikationen, finns ingen risk i att låta allmänheten ta del av produkten innan den är färdigutvecklad. Det finns däremot flera fördelar i att låta användare komma med synpunkter på våra idéer i olika skeden i projektet. Den främsta fördelen är att gruppmedlemmarna inte är representativa för applikationens målgrupp, vilket gör att beslut som baseras på användares erfarenheter och åsikter snarare än utvecklarnas, är mer relevanta för slutproduktens kvalitet.

Under projektets tidiga fas inhämtades åsikter med hjälp av en enkät. I mitten och slutet av projektet, med en halvfärdig respektive färdig applikation att testa, användes själva appen som utgångspunkt för att samla in information från användare.

• **Enkät**
Som en del av av insamlingsprocessen skapade gruppen en användarenkät, vars syfte var att ge gruppen en bättre insikt över vad för funktioner som efterfrågades av den tänkta målgruppen samt vilka tankar och förväntningar som fanns på en applikation av den här typen.

Enkäten utformades som ett 12 frågor långt webbformulär där respondenten värderar hur relevant produkten är, vilken funktionalitet som är intressant samt i vilken mån respondenten skulle använda den här typen av applikation för egen del. Frågorna valdes efter noga övervägande, där gruppen har tagit sådant som relevans, förväntningar och förutfattade meningar hos respondenterna i beaktande.

De svar som gruppen erhöll från enkäten hade stor inverkan på framtagandet av krav för applikationen, då svaren gav tydliga indikationer på vilka påtänkta funktioner som efterfrågades och vilka som möjliga användare inte såg som speciellt nödvändiga. Vi valde även att sammanställa responsen vi erhöll av de kvinnor som svarade på enkäten. Detta för att jämföra och se en möjlig skillnad av svar mellan hela enkätgruppen och endast kvinnor. I bilaga B respektive C finns sammanställningar av resultaten från enkäten.

3.4 Versionhanteringssystem

Projektet versionshanteras i Git. Versionshanteringssystemet Git valdes eftersom erfarenhet och kunskap om systemet fanns inom gruppen redan vid projektets start.

Gruppen eftersträvade att få en tydlig struktur i versionshanteringen som följer utvecklingsarbetet och projektplaneringen. Med en speciell modell för *branching*, Git-Flow [Dri10], kan arbetet parallelliseras och kategoriseras efter vilken funktionalitet arbetet tillhör. Git-Flow förespråkar användande av många *branches*, dels för ny funktionalitet, dels för att sammanställa *releases*. Separata *branches* gör det möjligt att fortsätta utveckla och skriva ny kod utan att påverka arbetet med att sammanställa den färdiga koden som ska finnas med i nästa *release*. Den här modellen och aktivt arbete med *branches* var okänt för alla gruppmedlemar vid projektets start och valdes till viss del för att öka förståelsen och upptäcka styrkan med att arbeta med *branches* inom versionshantering.

Versionshanteringssystemet nyttjas även för distribution av kodbasen inom utvecklarteamet. Git används i en central modell där ett repository på tjänsten GitHub agerar central server. GitHub valdes för att tjänsten är gratis för projekt med öppen källkod och har smidiga funktioner för att underlätta samarbetet mellan utvecklare.

4 Kravanalys

I projektets början definierades de krav och funktioner som skulle kunna tänkas finnas i applikationen. Gruppen insåg tidigt att kravställning är en iterativ process. Därför har kraven ändrats och finslipats allt eftersom projektet fortlöpt. Den feedback som erhölls från enkäten har använts i stor utsträckning vid framtagandet av applikationskraven. Även feedback från användartester har givit god insikt i vilka krav som är viktiga och bör tas med.

Viktigt att påpeka är att den uppsättning funktioner gruppen bestämde sig för att implementera tidigt i projektets gång inte är den samma som finns i den slutgiltiga applikationen. Funktionerna har ändrats av olika orsaker. Det kan vara att en funktion ansågs vara redundant efter användartester eller att en funktion inte var önskvärd enligt enkätsvaren. En del funktioner kunde inte heller implementeras på grund av tekniska begränsningar. Ett exempel på detta är SMS (Short Message Service)-funktionen som tänktes finnas i applikationen. Denna fick snabbt bytas ut mot en ny funktion och nya krav på grund av att iOS inte stödjer automatiska utskick av SMS.

I sektionerna nedan specificeras och förklaras de funktionella, icke-funktionella krav och designkrav gruppen valt att arbeta mot. Termer som införs är: sändare, mottagare och observatör. Dessa förklaras i detalj i följande sektioner.

4.1 Funktionella krav

Ett antal funktionella krav har definierats av gruppen efter egna omdömen och erfarenheter. Även enkäten och användartester har format kraven. Alla kraven sammanställdes sedan till en lista med tillhörande beskrivning och prioritet. Krav delades sedan in i tre kategorier: Användare och vänner, Larm och Följning. Detta i syfte att, på ett överskådligt sätt, få en bild av alla krav och vilka kategorier de tillhör. En beskriving av indelningarna följer nedan. En detaljerad lista av samtliga krav finns i bilaga D.1.

• Användare och vänner
En användare skall enkelt kunna lägga till vänner i sin vänlista samt på ett enkelt sätt få en överblick av alla vänner användaren har för tillfället. Vänlistan ska presenteras i en egen vy med tillhörande alternativ för att kunna lägga till en vän. Funktionen för att lägga till vänner ska vara lättförståelig. När en användare, sändaren, väljer att lägga till en vän skickas en vänförfrågan till den valda vännen, mottagaren, som sedan har möjligheten att acceptera eller avvisa förfrågan. I det fall mottagaren accepterade förfrågan kommer denne att läggas till i sändarens vänlista. Även sändaren kommer att läggas till i mottagarens vänlista. Om mottagaren avvisar förfrågan sker ingen uppdatering av respektive vänlista.

• **Följning**
Vid aktivering av en session skickas en förfrågan ut till alla vänner i sändarens lista. Dessa mottagare har då möjlighet att acceptera eller avvisa denna förfrågan. Om mottagaren avvisar förfrågan skickas ett meddelande tillbaka till sändaren att denne inte kan följa sessionen. Om däremot mottagaren accepterar förfrågan kommer denne att börja se sändarens position på sin karta inuti applikationen. Mottagaren har därmed blivit en observatör av sändaren. Sändaren kommer att få en bekräftelse på att mottagaren observerar den aktuella sessionen.

• **Larm**
En användare skall kunna larma vid behov, detta sker genom att användaren aktiverar larmet i huvudvyn. Larmet kommer att illustreras av en stor röd knapp för att underlätta för användaren. Vid larmning kommer ett meddelande att skickas ut till alla användarens vänner. Meddelandet består av ett generiskt textmeddelande samt en GPS-positition som visar var sändaren befinner sig på en kartvy hos observatörerna.

4.2 Icke-funktionella krav

Efter egna erfarenheter visste gruppen att användarvänlighet är oerhört viktigt när det kommer till små skärmar och input i form av touch. Detta ledde till en rad icke-funktionella krav som kontrollerar användarvänlighet och ger en riktlinje om hur duglig applikationen är ur en användares synvinkel. Listan av dessa krav fastställdes tidigt men har genomgått ständig förbättring under hela projektets gång. Enkäten och användartester har till stor del bidragit till dessa krav. De icke-funktionella kraven bestämmer den bakomliggande användarvänligheten samt diverse andra punkter som underlättar för framtida utveckling och underhåll. En lista över applikationens icke-funktionella krav finns i bilaga D.2.

4.3 Designkrav

För att få en attraktiv produkt krävs det att applikationen är användarvänlig och intuitiv. För att förenkla detta arbete har Apples guidelines för applikationsutveckling till iPhone använts [DL12d]. Apple beskriver hela flödet från idé till realisering och dessa punkter har inspirerat PersonAlarms design.

För att applikationen skall upplevas som användarvänlig och intuitiv upprättades ett antal designkrav som enligt gruppen bidrar till en god användarupplevelse. Fullständig lista över designkrav finns i bilaga D.3.

5 Grafisk design

För att PersonAlarm ska uppfylla de funktionella och icke funktionella krav som ställts, är det nödvändigt att designbesluten är väl analyserade och motiverade. Efter tester med olika designprototyper framkom att ett passande alternativ är flikar. Med flikar får användaren en överskådlig blick över applikationens funktioner, samt blir navigationen mellan applikationens olika funktioner intuitiv.

5.1 Prototyp

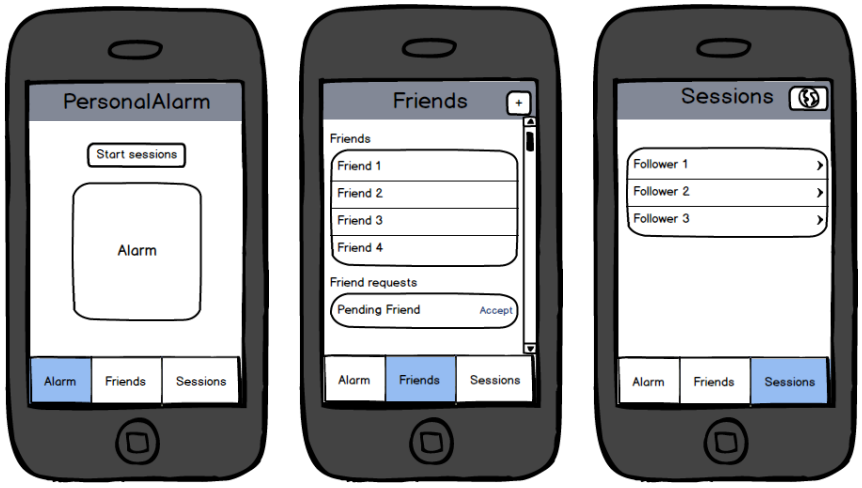
Den designprototyp som skapades visas i figur 5.1. Den innehöll tre flikar:

Alarm: Innehåller larmfunktionalitet.

Friends: Innehåller vänlista och alternativ för att lägga till vänner.

Sessions: Presenterar de aktuella sessionerna och en kartvy som visar positioner över alla användare observatören följer.

Designen lämpade sig väl på grund av det begränsade antalet vyer samt att den ger användare en god överblick över applikationens funktionalitet.



Figur 5.1: Den första designprototypen.

• **Sessioner**
En intressant frågeställning vi stod inför var hur sessionerna skulle illustreras och presenteras för användare. Enkätrespons och användarfeedback visade att det var önskvärt att observatörerna skall kunna följa sändarens position på en karta. Sändaren ska i sin tur kunna se vilka observatörer som följer sessionen. Önskvärt var även att ha denna relation i omvänd ordning. Med detta menas att en sändare skall kunna följa en

annan sändare. Detta illustreras i figur 5.2(a) där användaren och “Friend 1” följer varandra.

(a) (b)

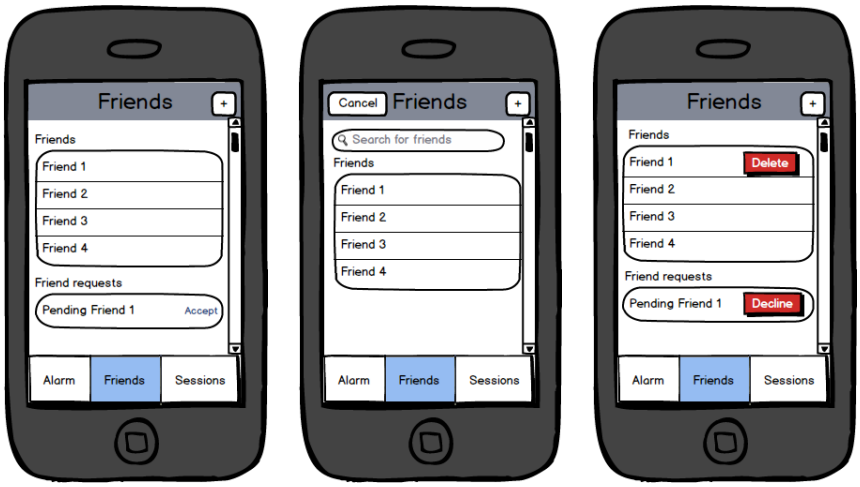
Figur 5.2: Sessions-hantering i applikationen

I listan “Following you” visas samtliga användarens observatörer. I den nedre listan “You are following” visas samtliga vänner som användaren observerar. Uppe i högra hörnet i gränssnittet ses en ikon i form av en jordglob. Om denna ikon klickas byts vyn till den högra där en karta visas. I denna karta visas alla de sändare som en observatör följer för tillfället. Om en användare vill tillbaka till den föregående vyn klickas knappen uppe till höger i kartvyn.

Detta sätt att byta mellan två relaterade vyer ansågs lämpa sig då informationen koncentreras på samma ställe. Att ha beskrivande ikoner visade sig även vara hjälpsamt för användare då ikoner och bilder ger en större tydlighet i vad knappen gör.

• **Vänner**
I figur 5.3(a) visas vänner-fliken. För att lägga till en ny vän klickas knappen med ett plustecken på uppe i högra hörnet. Då visas en sökruta ovanför vänlistan och här kan man söka efter en användare att lägga till som vän. Detta illustreras i figur 5.3(b). För att ta bort en vän drar man fingret över vännen och trycker på “Delete”.

En till lista under vänlistan har även lagts till. I denna ligger vänförfrågningar. Dessa kan accepteras genom att klicka på dem eller avvisas genom att dra fingret över dem och klicka på “Decline”. Borttagning av vän och avvisning av vänförfrågan visas i figur 5.3(c).



(a) (b) (c)

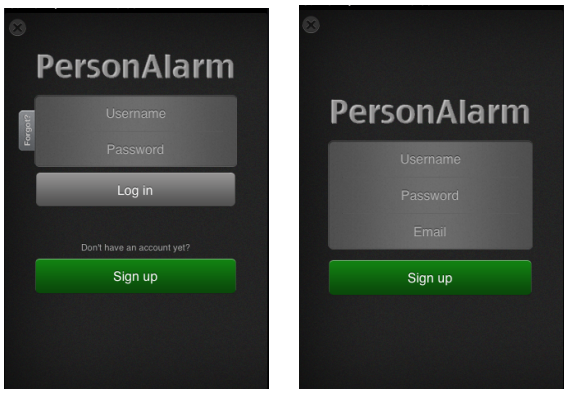
Figur 5.3: Design av vänner-fliken

Idéen om att dra fingret över knappen för att sedan få ytterliggare alternativ är inget nytänkande utan snarare en standard i iOS. Detta medför att iPhone-användare kommer att känna sig trygga med detta förfarande. Eftersom att alternativet att ta bort en vän inte måste visas hela tiden är detta en lämplig lösning. Däremot är funktionen att klicka på en vän och därmed acceptera förfrågan inte trivial för användare att förstå. Därför valde vi att skriva “Accept” och på så sätt göra det enklare för användaren att förstå vad en knapptryckning skulle innebära. Detta illustreras i figur 5.3(a).

5.2 Slutgiltig design

Mycket av den design som skapades i prototypen bevarades till den slutgiltiga versionen. Den användarfeedback gruppen erhöll pekade på att designen av gränssnittet gjorde att applikationen var enkel att använda. De ändringar som gjorts är till största del utformning av de grafiska komponenterna. I stället för knappar och simpla gränssnitt har dessa fyllts ut med mer grafiskt tilltalande och förklarande objekt. Den största ändringen är tillägget av en inloggnings-vy och en registrerings-vy. Till en början fanns ingen funktionalitet för att skapa användare och länka dessa som vänner. När detta sedan implementerades skapades inloggnings- och registreringsvyerna.

• **Inloggning och registrering**
När en användare vill logga in på sitt konto fyller denne i användaruppgifterna och trycker på "Log in". Om ett konto saknas kan användaren välja att registrera sig, vilket görs genom att trycka på "Sign up", där får användaren fylla i sin användarinformation. Användaren kan avbryta registeringen genom att trycka på krysset uppe i vänstra hörnet, vilket illustreras i figur 5.4(b).



(a) (b)

Figur 5.4: Design av inloggnings och registrerings-vyerna

• **Alarm**
När en användare vill utlösa larmet klickar denne på knappen i mitten av skärmen, se figur 5.5(a). Om användaren trycker på knappen visas ett meddelande som verifierar att larmet är utlöst. Detta för att larmfunktionen är en kritisk del och användaren bör bli underrättad om att ett larmmeddelande har skickats ut.

(a) (b)

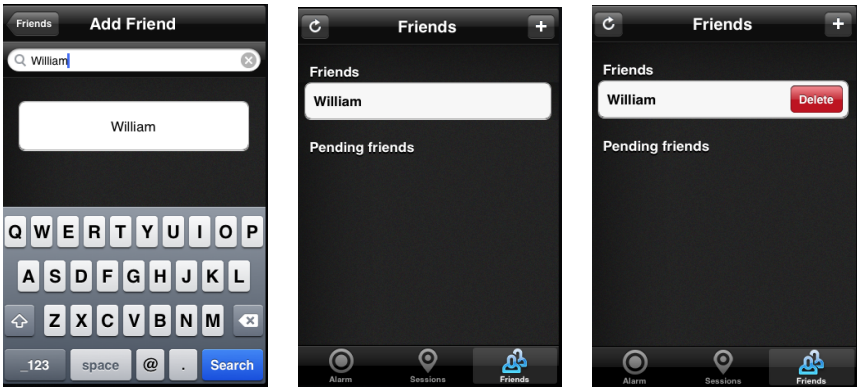
Figur 5.5: Design av alarm-vyn

Knappen som utlöser larmet liknar en nöd-stopp-knapp, vilket ansågs lämpa sig till

funktionen då det efterliknar något användare känner igen och kan relatera till. Det behövs därför ingen förklaring till vad knappen innebär eller hur den skall användas.

• **Vänner**
Den slutgiltiga designen av vyn vänner visas i figur 5.6. För att lägga till en vän klickar användaren på plusikonen och en ny vy visas. Här kan användaren sedan söka efter en vän att lägga till. När användaren sökt efter en vän visas alla träffar i en lista nedanför sökrutan och användaren kan då välja att klicka på en av dessa för att skicka iväg en vänförfrågan. Detta visas i figur 5.6(a). Denna design lämpade sig väl och valet att gå över till en ny vy för att lägga till vänner var nödvändigt då det kan finnas många sökträffar. Om alla dessa ska visas i vyn där man även ser vänner vore det plottrigt och svårt att få en god överblick.

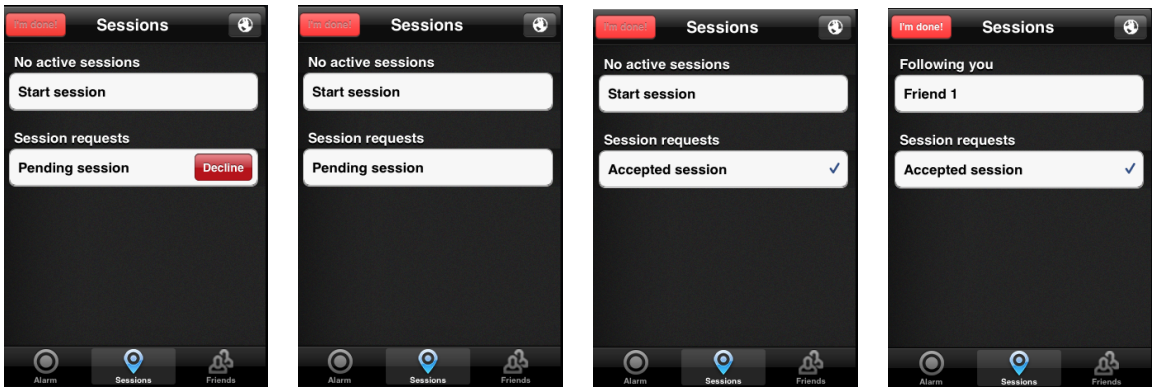
För att ta bort en vän drar man fingret över vännen och klickar sedan på "Delete". Detta visas i figur 5.6(c). Ett nytillskott är den uppdatera-knapp som finns i vyn upp till vänster. Denna knapp uppdaterar vyn och är en standardkomponent i iOS-applikationer som används ofta. På grund av detta krävs ingen förklaring för användare utan de kan antas veta vad knappen innebär.



(a) (b) (c)

Figur 5.6: Design av vänner-vyn

• **Sessioner**
En vy som tillkom efter att prorotypen hade färdigställts var sessions-vyn, vilken visas i figur 5.7. I denna vy kan användare skapa en session genom att klicka på knappen "Start session". Denna knapp visas endast när det inte finns någon aktiv session då användare endast kan ha en egen session åt gången. För att avsluta en aktiv session klickas knappen uppe till vänster där det står "I'm done!".

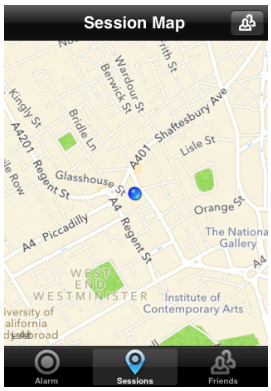


(a) (b) (c) (d)

Figur 5.7: Design av sessions-vyn

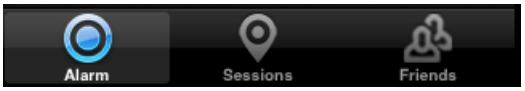
De sessioner man övervakar visas under “Session requests”. I figur 5.7(c) har det tillkommit en bock-markering från figur 5.7(b). Skillnaden här är att den utan markering inte har blivit accepterad ännu och den med markering är accepterad. Om en användare vill avvisa en förfrågan åstadkomms det genom att dra fingret över förfrågan och klicka på knappen som det står “Decline“ på vilket visas i figur 5.7(a). De användare som skickade sessionsförfrågningarna och som accepterats kan sedan ses i kartvyn.

Observatörer kan se positioner för de vännerna de följer i vyn Session map", vilket visas figur 5.8. I denna vy finns en karta i vilken användare kan navigera och zooma in och ut. För att lämna kartvyn och komma tillbaka till sessions-vyn klickas knappen uppe till höger i form av två människor.



Figur 5.8: Design av session map-fliken.

• **Flik-ikoner**
De ikoner vi valde att ha i applikationens flikar visas nedan i figur 5.9. Dessa utformades i slutet av projektet då de inte tillför någon utökad funktionalitet utan snarare är till för att förbättra applikationens grafiska design. Dessa ikoner har dock visat sig underlätta den initiala navigationen för användare. När en användare öppnar applikationen för första gången får han eller hon mer information av vad vår applikation kan göra och var tjänster finns.



Figur 5.9: Ikoner i flikarna.

6 Funktionalitet i applikationen

Liksom kraven är funktionerna uppdelade i tre olika kategorier: “Användare och vänner”, “Larm” och “Följning”. För implementeringen använder PersonAlarm en molnbaserad databas, som ramverket Parse SDK erbjuder, se avsnitt 2.3. Den molnbaserade databasen benämns i följande kapitel som Parse.

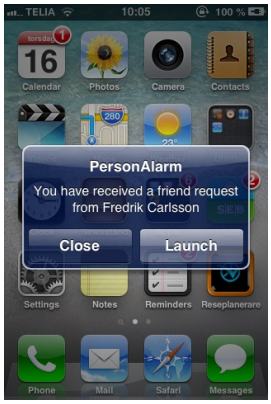
6.1 Användare och vänner

Nedan följer beskrivningar av alla funktioner som ingår i kategorin “Användare och vänner”.

• **Registrering av användare**
Vid registrering av användare utnyttjas en klass i Parse SDK, PFUser, som lämpar sig för registering av användare. Ett objekt av klassen PFUser måste ha ett unikt användarnamn, vilket kontrolleras av Parse. När registreringen av en användare är fullbordad sparas användarinformationen i den lokala databasen. Detta för att användaren inte ska behöva utföra en manuell inloggning varje gång appen startas.

• **Inloggning av användare**
Parse SDK erbjuder även funktionalitet för inloggning av användare. En inloggad användare ges åtkomst till sitt specifika användarobjekt och har de rättigheter som krävs för att manipulera detta objekt. När inloggningsinformationen har erhållits från användaren skickas denna till inloggningsklassen i Parse SDK. Om inloggningen lyckas sparas användarinformationen i den lokala databasen. Om inloggningen misslyckas visas ett felmeddelande och användaren får en ny chans att skriva in sina uppgifter. Efter en lyckad inloggning behöver ingen inloggningsskärm visas, detta realiseras genom att användarinformationen i den lokala databasen utnyttjas för inloggningen.

• **Se vänner och vänförfrågningar**
När användaren öppnar fliken vänner visas en tabell som innehåller de vänner och vänförfrågningar som är sparade i den interna databasen. Samtidigt hämtas de vänner och vänförfrågningar, som ännu inte blivit tillagda i den interna databasen. När hämtningen är klar uppdateras tabellen. Detta gör att onödiga och tidskrävande hämtningar av data undviks. Om PersonAlarm inte är aktiverad kan användare se vänförfrågningar i form av notiser, se figur 6.1, detta tack vare att sändaren av vänförfrågan skickar en Push Notification.



Figur 6.1: En vänförfrågan visas som en notis.

• **Sök efter användare**
För att lägga till en vän måste användaren först söka efter den tänkta vännen. Användaren börjar med att klicka på plustecknet, som illustreras i figur 5.6(b), då visas en ny vy med en sökruta. Användaren fyller i den eftersökta vännens användarnamn och väljer Sök". Då skickas ett meddelande innehållande det eftersökta användarnamnet till Parse, som i sin tur returnerar det eftersökta användar-objektet. Finns det ingen användare med det angivna användarnamnet i databasen visas detta i form av ett meddelande som säger att vännen inte kan hittas.

• **Skicka vänförfrågan**
Om en användare har sökt efter en annan användare och önskar att lägga till denne i sin vänlista, skickar användaren en vänförfrågan. En vänförfrågan består av ett objekt av typen Friend Request, Friend Request har två variabler, en mottagare och en sändare. Sändaren är den användare som skickar vänförfrågan, mottagaren är den användare sändaren önskar en vän-relation med. Friend Request objektet läggs till i Parse, därmed kan mottagaren sedan hämta vänförfrågan.

• **Svara på vänförfrågan**
Varje gång vänner-fliken öppnas hämtas användarens obesvarade vänförfrågningar, i form av Friend Request, från Parse. Om en användare har mottagit en vänförfrågan kan denne välja mellan att acceptera eller avvisa. När mottagaren accepterar en vänförfrågan skapas en vän-relation mellan mottagaren och sändaren. Om mottagaren väljer

att avvisa förfrågan tas Friend Request objetet bort från Parse.

• **Ta bort vän**
Om en användare vill ta bort en vän går denne in i vänner-fliken och letar upp den vän som önskas. När användaren väljer att ta bort denne, tas vän-relationen bort från Parse. Relationen tas även bort i den lokala databasen. Den borttagna vännen kan nu inte längre se användaren i sin vänlista, eftersom relationen är dubbelsidig och inte längre existerar. För illustration av borttagning av vän, se figur 5.6(c).

6.2 Larm

I följande avsnitt ges en beskrivning av de funktioner som ingår i kategorin "Larm".

• **Skicka larmmeddelande**
När en användare väljer att utlösa larmet skickas ett meddelande till alla vänner. Detta meddelande innehåller användarens position samt ett textmeddelande som beskriver att användaren är i behov av hjälp.

• **Ta emot larmmeddelande**
När en av ens vänner skickar ett larmmeddelande visas detta för alla observatörer. Observatörerna får då ett textmeddelande samt sändarens position som de sedan kan se på sin karta.

6.3 Följning

I sektionerna nedan följer beskrivningar av de funktioner som ingår i kategorin "Följning".

• **Starta en session**
När en användare väljer att starta en session genereras ett unikt Sessions-objekt för varje ansluten vän, i Parse. Användaren och vännen ges åtkomst till läsning och skrivning till detta objekt. Först när vännen accepterat sessionen skickas användarens position till observatören.

• **Svara på en session**
När en användare svarar på en session skickas en *Push Notification* tillbaka till användaren som startade sessionen. Om användaren accepterar sätts denne som observatör, och kan mottaga positioner från den användare som startade sessionen. Om användaren avböjer sessionsförfrågan tas det gemensamma Sessions-objektet bort från Parse.

• **Se användare på kartan**
När en användare öppnar sessions-fiken visas en kartvy över de vänner användaren följer. För att visa kartvyn utnyttjar PersonAlarm en färdig klass från ramverket MapKit,

UIMapView. Vännens position hämtas från webbtjänsten och läggs till som en punkt i denna klass. Punkten specifieras av koordinater i form av longitud och latitud.

• **Skicka position till följare**
Om användaren har startat en session med ett antal observatörer läggs användarens position kontinuerligt till i Parse. För att få tillgång till användarens GPS-koordinater används en klass, CLLocationManager, i ramverket CoreLocation. Genom att sätta sig själv som *delegate* till denna klass fås kontinuerlig uppdatering av användarens GPS-positioner. När en positionsförändring skett skickas den nya positionen till Parse. För en mer detaljerad beskrivning av *delegate* se avsnitt 2.1.

7 Resultat

Projektet hade som mål att skapa en mobilapplikation för iPhone som ökar tryggheten för användare vid utevistelse, minskar risken att faktiskt utsättas för överfall och som möjliggör att snabbt påkalla hjälp vid en nödsituation. Hur applikationen skulle uppfylla detta krav har förändrats allt eftersom vi erhållit enkätrespons och användarfeedback. I slutändan ledde det till en applikation där en användare bjuder in vänner som dels kan följa användarens position på en karta, dels erhåller ett meddelande om användaren utlöser ett larm.

En användare skapar en profil vilken sedan sparas i molnet. Användare kan sedan söka efter vänner och lägga dem i sin vänlista. När användaren aktiverar applikationen startas en session och alla användarens vänner får en förfrågan om att följa sessionen. Vännerna kan välja att acceptera eller avböja att följa sessionen. Om de accepterar kan de se positionen för användaren de följer på en karta i sin telefon.

Användartester har visat att den nya funktionaliteten är tillfredställande. Därmed kan vi konstatera att PersonAlarm uppfyller de krav vi satte upp när projektet startade.

8 Diskussion

Under projektets gång har vi stött på ett flertal problem. En av de målsättningar vi satte upp från början var att vi skulle erhålla nya kunskaper och erfarenheter. Att inte allting fungerat från början utan krävt fundering och logiskt tänkande, har givit oss motiv till en ökning av kunskap inom bl.a. projektorganisation, applikationsutveckling inom iOS och arbetsdokumetation. Det faktum att vi tvingats byta teknologier för webbtjänsten har bidragit till att öka våra kunskaper inom detta teknikområde. Att vi tvingats tänka om och testa nya idéer har drivit projektet framåt och i riktningar som inte var planerade men som i slutändan berikat projektarbetet.

8.1 Problem och hinder

Det största problem vi stötte på under utvecklingen var att Apple inte tillåter automatiska SMS-utskick. Detta var en essentiell del i vår originalidé som vi tvingades revidera. För att reda ut detta problem fick vi tidigt under projektet brainstorma kring alternativa lösningar. Detta ledde till att en webbtjänst utvecklades så att meddelanden och data kan skickas mellan användare.

Nästa stora problem vi stötte på var språkvalet för utvecklingen av webbtjänsten. Vi valde till en början att utveckla i Ruby och då använda Rails-plattformen. Till en början föreföll detta vara ett lämpligt val. Ruby är ett kraftfullt språk med stor funktionalitet, problemet är inlärningskurvan. Ingen i projektet hade någon större kunskap kring språket och detta ledde till mycket slösad tid. En månad lades ner på detta innan vi bestämde oss för att byta språk till Java och använda Spring Framework. Detta visade sig också vara ett olämpligt val. Det uppstod problem när vi försökte kommunicera mellan mobilapplikationen och webbtjänsten och även detta ramverk valdes bort till fördel för Parse, se avsnitt 2.3.

Ett problem som uppstod tidigt var att en av gruppmedlemmarna hoppade av projektet. Från början var vi 5 personer men under nästan hela projektets gång har vi varit 4 personer. Detta har lett till tidsbrist under implementeringen men vi känner att det resultat som uppnåtts ligger på en god nivå och alla gruppmedlemmar är både nöjda och stolta över det vi åstadkommit.

8.2 Att arbeta med Objective-C och iOS

För att kunna utveckla en applikation till iPhone krävs det att program skrivs i Objective-C och för iOS-plattformen. Detta språk och plattform har varit trevliga att arbeta med. Det finns gott stöd från API:er och mycket information och guider finns att hämta på Apples egna sidor [DL12d].

Objective-C är ett lättförståeligt men även komplext språk. Eftersom att alla medlemmar i gruppen har god erfarenhet av både Java och C var det mycket enkelt att sätta sig in i. Objective-C är plattformsspecifikt och detta gör att språket är direkt kopplat till plattformen, i vårt fall iPhone. Detta leder till hög integration mellan språk och plattform.

Det skall tilläggas att språket hämmas en del av plattformen i den mening att det finns begränsningar iOS lägger på Objective-C. Ett exempel på detta är den SMS-funktion vi tänkte utveckla. Denna kunde inte implementeras på grund av att Apple har lagt en restriktion på automatiska SMS-utskick av applikationer vilket har satt käppar i hjulen för oss.

8.3 Förarbete

Inledningsvis utfördes ett grundligt förarbete av samtliga medlemmar. Förarbetet bestod dels i teknisk kunskapsinhämtning, dels en enkätstudie. Ingående studier gjordes av iOS, Objective-C och mobila överfallsskydd. Detta ledde till att starttiden för projektet kunde minskas och applikationens kvalité började på en god nivå. Frågorna i enkätstudien utgick från de förslag på krav som vi utarbetade i början av projektet. Responsen från enkäten innehöll värdefull kritik och lade grunden för den kravspecifikation vi sedan utvecklade.

För att förbättra och utöka responsen hade en andra enkät kunnat skickas ut. Resultatet hade varit ny respons vilken hade kunnat jämföras med den föregående och viktiga funktioner och krav hade kunnat urskiljas. Även att rikta enkäten mot en mer specifik målgrupp, i vårt fall kvinnor, hade givit ännu mer detaljerad respons. Detta hade lett till en reviderad och mer detaljerad kravspecifikateion.

8.4 Relaterade arbeten

Vid projektets start fanns det ett antal liknande applikationer på App Store. Detta har dock inte påverkat vårt arbete. Det har under hela projektet varit viktigt för oss att skapa en applikation med de funktioner vi känner att vi vill inkludera. Däremot har gruppen laddat ner ett antal av applikationerna som finns tillgängliga på App Store för att testa funktionalitet och för att få inspiration. För tillfället finns det två liknande iOS-applikationer som gruppen anser vara av riktigt god kvalité vad gäller gränssnitt och funktionalitet och dessa är Panik [Pre13b] och PFO Sheild [Pre13c]. Under projektets gång släppte även Missing People ett överfallsskydd i form av en iOS-applikation [Pav13]. Detta tycker vi understryker vikten av dessa applikationer och ytterligare motiverar gruppens arbete.

8.5 Förslag och vidare utveckling

Vi ser att det finns stora möjligheter till vidare utveckling för applikationskonceptet. Problematiken med hur man med IT på ett smidigt, säkert och enkelt sätt kan möjliggöra för människor att känna sig trygga är fortfarande ett olöst problem.

En mängd funktioner diskuterades under projektets gång men valdes bort med hänsyn till arbetets omfattning. Några av dessa var att applikationen skulle erbjuda direktkoppling till ett eller flera säkerhetsbolag alternativt larmcentraler, någon form av premiumtjänst som erbjuder professionell bevakning eller möjligheten att koppla till taxitjänst.

9 Slutsats

I projektets början hade vi en idé om att det fanns ett behov av ett mobilt överfallslarm för iOS-baserade enheter och vi har färdigställt en första fungerande version av PersonAlarm. Funktionaliteten har under projektets gång förbättrats till följd av respons och feedback från en enkätstudie respektive genomförda användartester.

De tjänster PersonAlarm tillhandahåller uppfyller de målsättningar och krav vi satte upp i början av projektet. PersonAlarm har mottagits väl av den tänkta målgruppen och bidrar med de tjänster som mest efterfrågas.

Referenser

[Bro12] Brottsrummet. *Brott och statistik*. 2012. URL: http://www.brottsrummet.se/sv/brott-och-statistik (hämtad 2013-04-17).

[Cen12] Statistiska Centralbyrån. *Personer som känner sig otrygga vid utevistelse sen kväll efter ålder 2011*. 2012. URL: http://www.scb.se/Pages/ThematicAreaTableAndChart____342927.aspx (hämtad 2013-02-27).

[DL12a] iOS Developer Library. *About the iOS Technologies*. 19 sept. 2012. URL: http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/Introduction/Introduction.html (hämtad 2013-03-25).

[DL12b] iOS Developer Library. *Core OS Layer*. 19 sept. 2012. URL: http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/CoreOSLayer/CoreOSLayer.html#//apple_ref/doc/uid/TP40007898-CH11-SW1 (hämtad 2013-03-25).

[DL12c] iOS Developer Library. *Core Services Layer*. 19 sept. 2012. URL: http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/CoreServicesLayer/CoreServicesLayer.html#//apple_ref/doc/uid/TP40007898-CH10-SW5 (hämtad 2013-03-25).

[DL12d] iOS Developer Library. *iOS Human Interface Guidelines*. 2012. URL: http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html (hämtad 2013-02-26).

[DL12e] iOS Developer Library. *Media Layer*. 19 sept. 2012. URL: http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/MediaLayer/MediaLayer.html#//apple_ref/doc/uid/TP40007898-CH9-SW4 (hämtad 2013-03-25).

[DL12f] iOS Developer Library. *View Controller Programming Guide for iOS*. 13 dec. 2012. URL: http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/Introduction/Introduction.html (hämtad 2013-03-26).

[DL13] iOS Developer Library. *App States and Multitasking*. 28 jan. 2013. URL: http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/ManagingYourApplicationsFlow/ManagingYourApplicationsFlow.html (hämtad 2013-05-19).

[Dri10] Vincent Driessen. *A successful Git branching model*. 2010. URL: http://nvie.com/posts/a-successful-git-branching-model/ (hämtad 2013-03-28).

[Elk08] M. Elkstein. *Learn REST: A Tutorial: 1. What is REST?* 2008. URL: http://rest.elkstein.org/2008/02/what-is-rest.html (hämtad 2013-05-16).

[Flu12] Flurry. *iOS and Android Adoption Explodes Internationally*. 2012. URL: http://blog.flurry.com/bid/88867/ios-and-android-adoption-explodes-internationally (hämtad 2013-05-19).

[If12] If. *Överfallsskydd*. 2012. URL: http://www.if-sakerhet.se/personligsakerhet/overfallsskydd (hämtad 2013-02-26).

[Inf07] Apple Press Info. *Apple Reinvents the Phone with iPhone*. 2007. URL: http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html (hämtad 2013-03-25).

[Lib12] Mac Developer Library. *Programming With Objective-C*. 13 dec. 2012. URL: http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html (hämtad 2013-03-27).

[Nie13] Nielsen. *The Mobile Consumer*. 2013. URL: http://nielsen.com/content/dam/corporate/us/en/reports-downloads/2013\%20Reports/Mobile-Consumer-Report-2013.pdf (hämtad 2013-02-27).

[Par13] Parse. *Docs Overview*. 2013. URL: https://www.parse.com/docs/ (hämtad 2013-05-19).

[Pav13] Adrianna Pavlica. "Missing Peoples egna app: Ett överfallslarm". I: *Expressen GT* (13 mars 2013).

[Pre13a] iTunes Preview. *App Store*. 2013. URL: https://itunes.apple.com/us/genre/ios/id36?mt=8 (hämtad 2013-02-27).

[Pre13b] iTunes Preview. *Panik - Personal Assault Alarm*. 2013. URL: https://itunes.apple.com/se/app/panik-personal-assault-alarm/id588969126?l=en&mt=8 (hämtad 2013-05-09).

[Pre13c] iTunes Preview. *PFO Shield*. 2013. URL: https://itunes.apple.com/se/app/pfo-shield/id537989365?l=en&mt=8 (hämtad 2013-05-09).

[Res13] RestKit. *RestKit*. 2013. URL: http://restkit.org (hämtad 2013-03-27).

[Sut11] Ken Schwaber; Jeff Sutherland. *The Scrum Guide*. 2011. URL: http://www.scrum.org/Portals/0/Documents/Scrum\%20Guides/Scrum_Guide.pdf (hämtad 2013-05-16).

[TIO13] TIOBE. *TIOBE programming community index for march 2013*. 1 mars 2013. URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (hämtad 2013-03-27).

Bilagor

A Enkätundersökning - Informationsundersökning om överfallsalarm

Vi är en kandidatgrupp som skall utveckla en mobilapplikation för överfallsalarm - och för att göra detta så bra som möjligt så skulle vi gärna vilja ha lite input om vad du tror möjligheterna och behovet för en sådan skulle vara.

- **Allmänna frågor**

1. Kön?
   (a) Man
   (b) Kvinna
2. Hur gammal är du?
   (a) 0-17
   (b) 18-24
   (c) 25-34
   (d) 35-49
   (e) 50-
3. Har du en smartphone som kan installera applikationer?
   (a) Ja, en iPhone
   (b) Ja, en Android telefon
   (c) Nej
4. Brukar du ofta känna dig orolig för överfall efter t.ex. en utekväll?
   (a) Skala från 1 (Nästan aldrig) till 10 (Väldigt ofta)

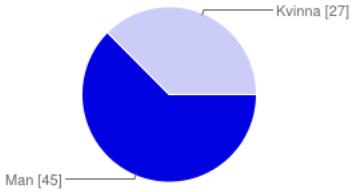
- **Applikationsspecifika frågor**

5. Tror du att det finns behov för en sådan här applikation?
   (a) Ja
   (b) Till viss del
   (c) Nej

6. Skulle du använda en sådan här applikation?
   (a) Ja, har ingen installerad i nuläget.
   (b) Ja, har en installerad i nuläget.
   (c) Nej.
7. Hur skulle du värdera följande funktioner i ett överfallsalarm?
   Alla frågor är en skala från 1 (Inte speciellt nödvändig) till 5 (Väldigt nödvändig).
   (a) SMS-utskick till vänner/familj
   (b) Automatisk uppringning av larmtjänst (t.ex. 112)
   (c) Högljutt larm
   (d) Inspelning av ljud/bild
   (e) Länka med vänner (t.ex. följa hemgång via GPS)
   (f) Larm vid avvikning från planerad rutt
   (g) Koppling till sociala medier (Facebook, Twitter, etc.)
   (h) Koppling till webbtjänst
   (i) Koppling till larmbolag (t.ex. Securitas)
   (j) Koppling till Taxi
8. Hur troligt vore det att du använder en sådan här applikation i följande situationer?
   Alla frågor är en skala från 1 (Inte speciellt troligt) till 5 (Väldigt troligt).
   (a) Våldtäktsförsök
   (b) Personrån
   (c) Fallskada
   (d) Cykelstöld
   (e) När man går hem (i förebyggande syfte)
9. Har du några övriga tankar kring behovet av en sådan här applikation?
   (a) (Fritextsvar)

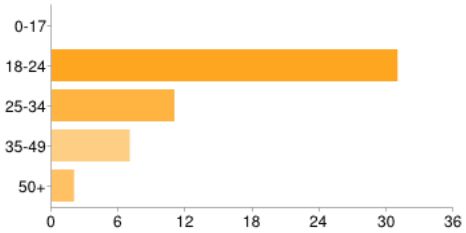
B Respons från enkätundersökning

Vad är du för kön?



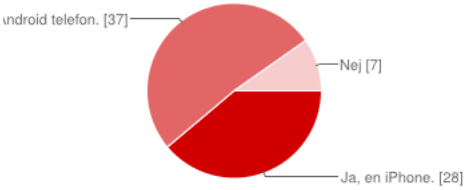
Man	**45**	63%
Kvinna	**27**	38%

Hur gammal är du?



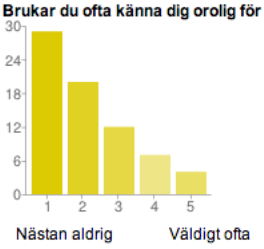
0-17	**0**	0%
18-24	**31**	43%
25-34	**11**	15%
35-49	**7**	10%
50+	**2**	3%

Har du en smartphone som kan installera applikationer?



Ja, en iPhone.	**28**	39%
Ja, en Android telefon.	**37**	51%
Nej	**7**	10%

Brukar du ofta känna dig orolig för överfall när du går hem efter t.ex en utekväll?

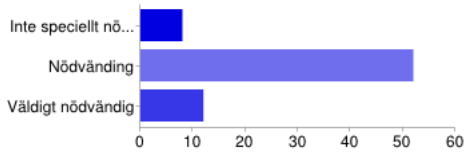


1 - Nästan aldrig	**29**	40%
2	**20**	28%
3	**12**	17%
4	**7**	10%
5 - Väldigt ofta	**4**	6%

Skulle du använda en sådan här applikation?

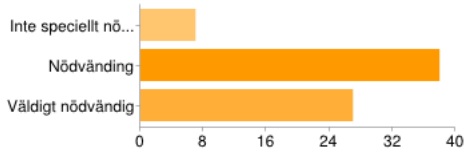
Ja, har ingen installerad i nuläget.	36	50%
Ja, har en installerad i nuläget.	0	0%
Nej.	36	50%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - SMS-utskick till vänner/familj



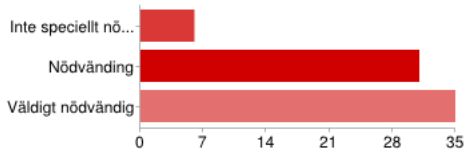
Inte speciellt nödvändig	8	11%
Nödvändig	52	72%
Väldigt nödvändig	12	17%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Automatisk uppringning av larmtjänst (t.ex 112)



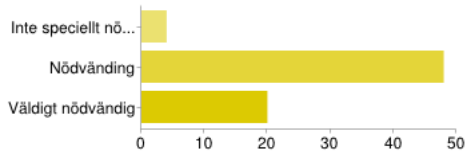
Inte speciellt nödvändig	7	10%
Nödvändig	38	53%
Väldigt nödvändig	27	38%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Högljutt larm



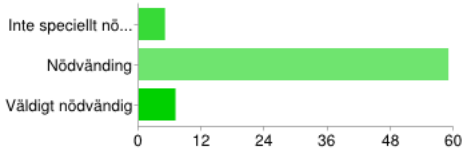
Inte speciellt nödvändig	6	8%
Nödvändig	31	43%
Väldigt nödvändig	35	49%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Inspelning av ljud/bild



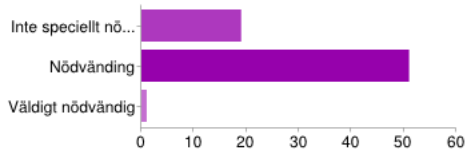
Inte speciellt nödvändig	**4**	6%
Nödvändig	**48**	67%
Väldigt nödvändig	**20**	28%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Länka med vänner (t.ex för att följa hemgång via GPS)



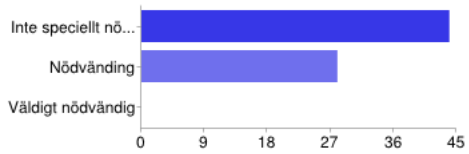
Inte speciellt nödvändig	**5**	7%
Nödvändig	**59**	82%
Väldigt nödvändig	**7**	10%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Larm vid avvikning från planerad rutt



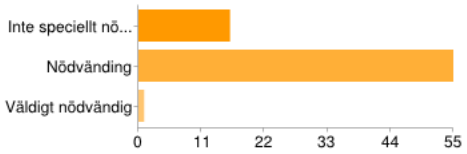
Inte speciellt nödvändig	**19**	26%
Nödvändig	**51**	71%
Väldigt nödvändig	**1**	1%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Koppling till sociala medier (Facebook, Twitter etc.)



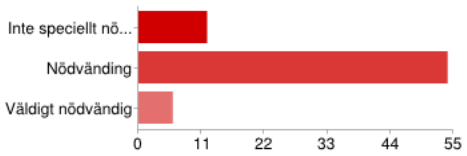
Inte speciellt nödvändig	**44**	61%
Nödvändig	**28**	39%
Väldigt nödvändig	**0**	0%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Koppling till webbtjänst



Inte speciellt nödvändig	**16**	22%
Nödvänding	**55**	76%
Väldigt nödvändig	**1**	1%

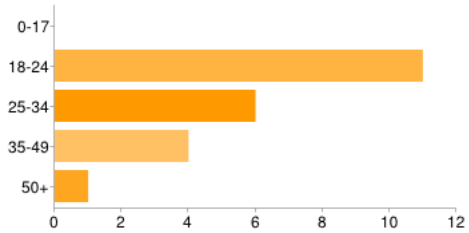
Hur skulle du värdera följande funktioner i ett överfallsalarm? - Koppling till larmbolag (t.ex Securitas)



Inte speciellt nödvändig	**12**	17%
Nödvänding	**54**	75%
Väldigt nödvändig	**6**	8%

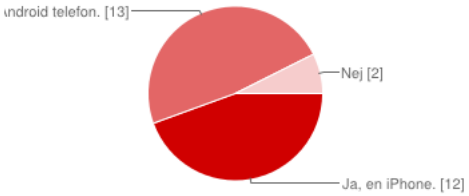
C Respons från enkätundersökning från kvinnor

Hur gammal är du?



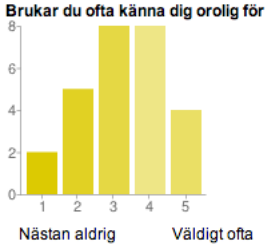
0-17	**0**	0%
18-24	**11**	41%
25-34	**6**	22%
35-49	**4**	15%
50+	**1**	4%

Har du en smartphone som kan installera applikationer?



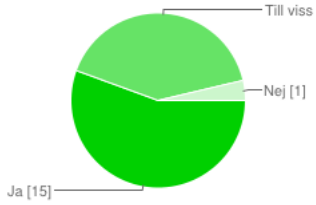
Ja, en iPhone.	**12**	44%
Ja, en Android telefon.	**13**	48%
Nej	**2**	7%

Brukar du ofta känna dig orolig för överfall när du går hem efter t.ex en utekväll?



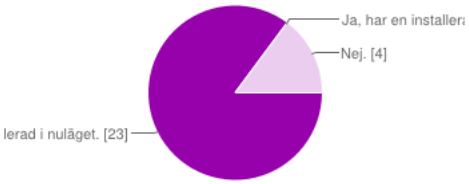
1 - Nästan aldrig	**2**	7%
2	**5**	19%
3	**8**	30%
4	**8**	30%
5 - Väldigt ofta	**4**	15%

Tror du att det finns ett behov för en sådan här applikation?



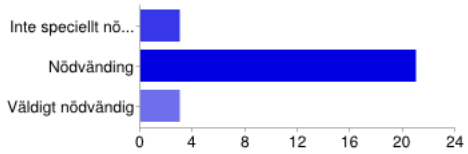
Ja	**15**	56%
Till viss del	**11**	41%
Nej	**1**	4%

Skulle du använda en sådan här applikation?



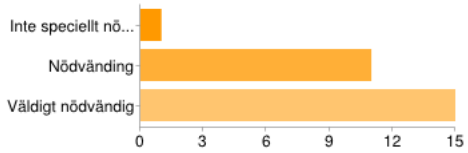
Ja, har ingen installerad i nuläget.	**23**	85%
Ja, har en installerad i nuläget.	**0**	0%
Nej.	**4**	15%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - SMS-utskick till vänner/familj



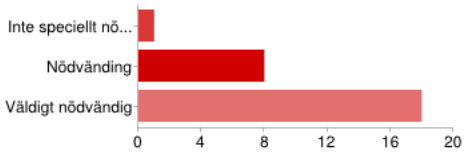
Inte speciellt nödvändig	**3**	11%
Nödvändig	**21**	78%
Väldigt nödvändig	**3**	11%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Automatisk uppringning av larmtjänst (t.ex 112)



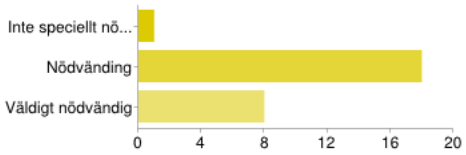
Inte speciellt nödvändig	**1**	4%
Nödvändig	**11**	41%
Väldigt nödvändig	**15**	56%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Högljutt larm



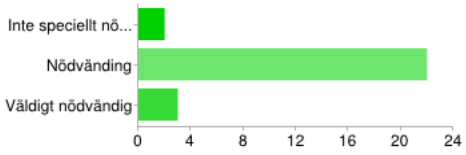
Inte speciellt nödvändig	1	4%
Nödvändig	8	30%
Väldigt nödvändig	18	67%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Inspelning av ljud/bild



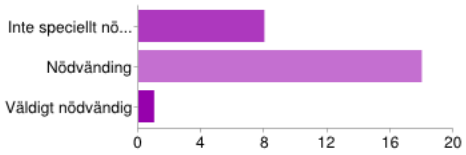
Inte speciellt nödvändig	1	4%
Nödvändig	18	67%
Väldigt nödvändig	8	30%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Länka med vänner (t.ex för att följa hemgång via GPS)



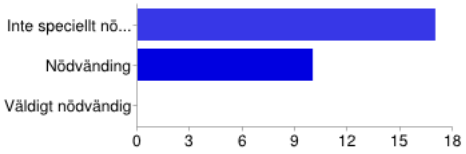
Inte speciellt nödvändig	2	7%
Nödvändig	22	81%
Väldigt nödvändig	3	11%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Larm vid avvikning från planerad rutt



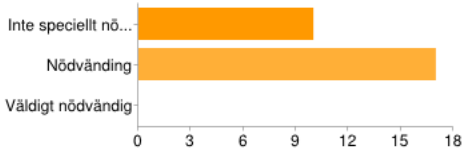
Inte speciellt nödvändig	8	30%
Nödvändig	18	67%
Väldigt nödvändig	1	4%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Koppling till sociala medier (Facebook, Twitter etc.)



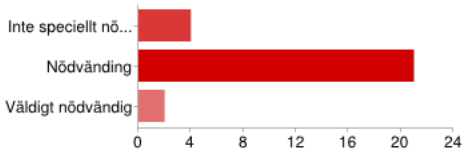
Inte speciellt nödvändig	**17**	63%
Nödvändig	**10**	37%
Väldigt nödvändig	**0**	0%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Koppling till webbtjänst



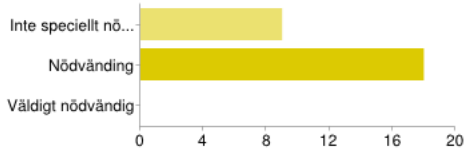
Inte speciellt nödvändig	**10**	37%
Nödvändig	**17**	63%
Väldigt nödvändig	**0**	0%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Koppling till larmbolag (t.ex Securitas)



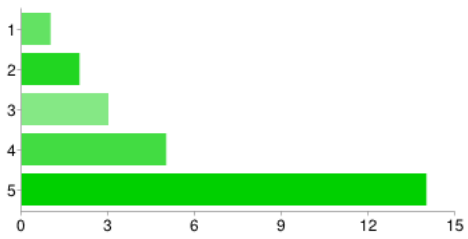
Inte speciellt nödvändig	**4**	15%
Nödvändig	**21**	78%
Väldigt nödvändig	**2**	7%

Hur skulle du värdera följande funktioner i ett överfallsalarm? - Koppling till Taxi



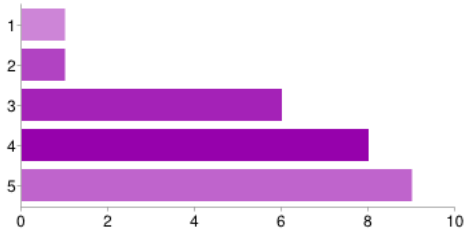
Inte speciellt nödvändig	9	33%
Nödvänding	18	67%
Väldigt nödvändig	0	0%

Hur troligt vore det att du använder en sådan här applikation i följande situationer? - Våldtäktsförsök



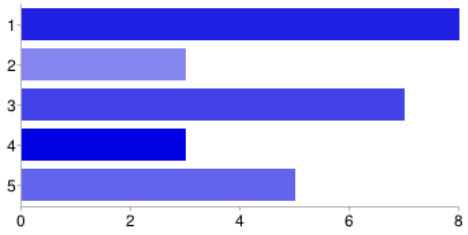
1	1	4%
2	2	7%
3	3	11%
4	5	19%
5	14	52%

Hur troligt vore det att du använder en sådan här applikation i följande situationer? - Personrån



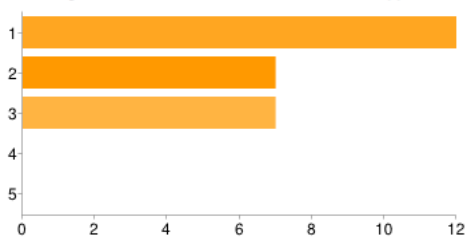
1	1	4%
2	1	4%
3	6	22%
4	8	30%
5	9	33%

Hur troligt vore det att du använder en sådan här applikation i följande situationer? - Fallskada



1	8	30%
2	3	11%
3	7	26%
4	3	11%
5	5	19%

Hur troligt vore det att du använder en sådan här applikation i följande situationer? - Cykelstöld



1	**12**	44%
2	**7**	26%
3	**7**	26%
4	**0**	0%
5	**0**	0%

D Kravspecifikation

Alltefter projektets gång har dessa kravlistor uppdaterats och itererats över.

Kravprioriteten sträcker sig från 1 till 3 där 1 har högsta prioritet.

D.1 Funktionella krav

Nedan följer applikationens funktionella krav vilka är indelade i tre kategorier: Användare och vänner, Larm och Följning.

D.1.1 Användare och vänner

Nr	Beskrivning	Prioritet
1	En användare skall kunna lägga till vänner i sin vänlista.	1
2	En användare skall kunna acceptera vänförfrågningar.	1
3	En användare skall kunna avvisa vänförfrågningar.	1
4	En användare skall kunna se alla vänner denne har i sin vänlista.	1
5	En användare skall kunna se detaljerad information om en vän.	2

D.1.2 Larm

Nr	Beskrivning	Prioritet
1	En användare skall kunna skicka ut ett meddelande till en eller flera personer vid larm.	1
2	En användare skall kunna manipulera den text som finns i meddelandet som skickas ut vid larm.	2
3	En användare skall kunna fördefiniera vilka personer som tar emot larmmeddelandet.	3

D.1.3 Följning

Nr	Beskrivning	Prioritet
1	En användare i rollen som observatör ska kunna, vid en session, följa en sändare på kartan i sin telefon.	1
2	En användare ska kunna skicka en förfrågan till sina vänner om att följa denne.	1
3	En användare ska kunna acceptera en följningsförfrågan.	1
4	En session ska startas när följningsförfrågan accepteras.	1
5	En användare ska kunna avvisa en följningsförfrågan.	1
6	En användare ska kunna avsluta en session.	1

D.2 Icke-funktionella krav

Nr	Beskrivning	Prioritet
1	Applikationen ska vara användarvänlig, intuitiv och lättförståelig.	1
2	Applikationens nätverkskommunikation måste vara säker ty känslig information kan förekomma.	1
3	Applikationen ska ha stöd för iOS 6.	1
4	Applikationen ska kunna köras på en iPhone 3GS eller valfri nyare iPhone-modell.	1
5	Applikationen får endast sluta vara responsiv när ett allvarligt fel uppstår	2
6	Applikationen ska vara enkel att underhålla.	2
7	Applikationen ska kunna hantera oväntade avbrott, exempelvis ett inkommande telefonsamtal.	2
8	Applikationen ska vara responsiv och användarens möjlighet att påverka gränssnittet ska ej påverkas av fördröjningar vid nätverkskommunikation.	2
9	Applikationen skall vara utökningsbar i den mening att det ska vara lätt lägga till samt utöka funktionalitet.	3
10	Applikationen ska kunna köras i bakgrunden.	3

D.3 Designkrav

Nr	Beskrivning	Prioritet
1	Designen ska vara konsekvent och sammanhängande.	1
2	Applikationen ska vara uppbyggd av vyer i flikar.	1
3	Funktioner eller inställningar ska ej vara gömda. I den mening att användare ska, utan bekymmer, kunna finna den funktion denne söker.	2