CHALMERS

Androidapplikation för fjärrövervakning av affärskritiska driftsystem

Android application for remote monitoring of business-critical operating systems

Examensarbete inom högskoleingenjörsprogrammet

ANNA GUSTAFSSON
JONAS ÅSTRÖM

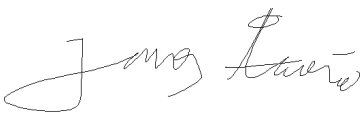
Instutitionen för Data- och Informationsteknik
Examinator Lars Svensson
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige, 2014

Förord

Vi, författarna av detta examensarbete, studerar båda två på Högskoleingenjörsprogrammet på Chalmers Universitet i Göteborg, om än med olika inriktningar (data och mekatronik), och under de 3 år som utbildningen omfattar ska många ämnen behandlas på relativt kort tid för att erhålla titeln ingenjör. I och med detta är det därför befriande att genom examensarbetet kunna fokusera helhjärtat på ett arbete som innefattar det man fått upp intresse för under sin utbildning, och utforska de möjligheter som man tidigare bara skymtat. Detta projekt omfattar 15 högskolepoäng, motsvarande 10 veckor, men det var så roligt och givande att det kändes som att det gick mycket snabbare än så, till stor del tack vare alla inblandade, både från Ilait och Chalmers.

Vi vill därför tacka Ilaits alla medarbetare för en mycket trevlig tid på företaget. Extra stort tack till Kim Lundgren på Ilait, som förutom att ha agerat beställare för projektet även har varit till stor hjälp under arbetets gång, både vid stora som små frågor.
Stort tack även till vår handledare på Chalmers, Sakib Sistek, som med humor och värme varit vårat ”ankare” på Chalmers, redan innan projektet hade börjat.

…….
Anna Gustafsson



…….
Jonas Åström

Sammanfattning

Ilait, ett företag som tillhandahåller webbhotell och molntjänster exklusivt till återförsäljare, har sedan start använt sig av ett sms-system för att meddela jourhavande driftpersonal då någon av deras tjänster råkar ut för ett fel. Då detta tillvägagångssätt varken är flexibelt eller effektivt (speciellt vid de tillfällen då ett fel ger upphov till många följdfel) ville företaget att en applikation till Android skulle tas fram, som kan ersätta det gamla sms-systemet och även ger möjlighet till ytterligare övervakning av driftssystemen. Syftet med detta projekt blev därmed att undersöka möjligheterna till server till Android-kommunikation genom Google Cloud Messaging, och med hjälp av detta ta fram den önskade applikationen åt företaget. I andra änden utav denna larmtjänst behövs även en server med vissa funktioner, men detta arbete är begränsat till Android-applikationen och kommer därmed endast ställa upp en lista med krav på den server som i framtiden ska användas, utan att vidare behandla hur servern ska byggas upp för att tillgodose dessa krav. Resultatet av projektet blev en applikation som matchade de förutsättningar och krav som ställts på den, med undantag för enstaka funktioner som övergavs efter överenskommelse med beställaren under arbetets gång.

Summary

Ilait, a company that provides web-hosting and cloud services exclusively for resellers, has since it started been using a sms-system for sending messages informing the operating staff on call when any of their services stops working as expected. Since this way of sendning alarms to the personell is not flexible nor effective (especially when one error induces several consequential failures), the company was asking for an Android-application that could replace the old system and also make it possible to monitor the systems further. The purpose of this project was therefore to examine the possibilities of server to Android-communication through Google Cloud Messaging, and with this create the desired application for the company. On the other end of this alarm-service a server with some special functions is also needed, but this project is limited to creating the Android-application and will therefore only set up a list of requirements on the server that is going to be used in the future, without furter instructions on how it should be set up in order to live up to these demands. The project resulted in an application which matched the prerequisites and the requirements that had been made, with the exception of some functions that was abandoned after an agreement with the client during the time of the project.

INNEHÅLLSFÖRTECKNING

BETECKNINGAR

GCM	Google Cloud Messaging (Googles molntjänst)
GUI	Graphical User Interface (grafiskt användargränssnitt)
IPC	Interprocess Communication (interprocesskommunikation) Kommunikation mellan olika processer i program/applikationer.
XML	Extensible Markup Language (programspråk)
HTTP	Hypertext Transfer Protocol (kommunikationsprotokoll)
XMPP	Extensible Messaging and Presence Protocol (protokoll för snabbmeddelande)
SDK	Software Development Kit (Mjukvaru-utvecklarpaket)
JSON	JavaScript Object Notation (Java objekt notation) Ett format för beskrivning utav objekt i samband med utbyte utav data.
API	Application Programming Interface (programmeringsgränssnitt för applikationer)
Hashtabell	Datastruktur som, tack vare sin uppbyggnad, går mycket snabbt att söka igenom efter enskilda värden.

Objekt och monitorer
I denna dokumentation kommer begreppen ”objekt” och ”monitor” användas flitigt. I detta fall avser objekt ett utav Ilaits system, exempelvis en server, och med monitor menas en del utav ett objekt som kan ge upphov till fel. Detta innebär alltså att ett objekt kan ha massvis utav monitorer, men en monitor tillhör aldrig mer än ett objekt.

1. INLEDNING

1.1 Bakgrund

Ilait är ett företag lokaliserat i Kungälv som tillhandahåller molntjänster och webbhotell exklusivt för återförsäljare. I dagsläget erhåller Ilaits jourhavande driftpersonal ett eller flera textmeddelande via telefonen då ett fel uppkommer i någon utav deras tjänster, system eller servrar. Detta innebär att ett fel kan ge många larm på samma gång, larmet låter lika högt oavsett om det är kritiskt eller ej och det finns lite eller inget utrymme för egna inställningar.

1.2 Syfte

Uppdraget innebär att en Android-applikation ska skapas, som kan ersätta det nuvarande sms-systemet. Användaren ska kunna välja om denne vill vara inloggad på jouren eller inte, samt om larmen ska ge olika signaler beroende på om de är kritiska eller ej. Inträffar flera fel samtidigt ska fortfarande bara en signal erhållas medan felen samlas i en kö som kan expanderas till en längre lista. Användaren ska dessutom kunna se en översikt över de objekt som övervakas, samt historik från tidigare inträffade fel. Resultatet bör vara en applikation som, förutom att fjärrövervaka Ilaits system, servrar och tjänster, kan ge driftspersonalen en snabb översikt över de olika objektens nuvarande status.

1.3 Mål

Det slutgiltiga målet med projektet är att tillhandahålla ett fungerande användargränssnitt i Android-miljö, som enbart efter ändring av några konstanter i programmet direkt kan börja användas utav Ilaits personal för att erhålla larm vid driftstörningar.

1.4 Avgränsningar

Projektet innefattar ej serversidans uppgift, utan endast applikationen som kommer hämta uppgifter och göra förfrågningar till företagets server. Applikationen ska endast skapas för Android-plattformen, då det är denna som används utav Ilaits personal.

1.5 Precisering av frågeställningen

Under arbetets gång skall ett antal frågeställningar besvaras:

- Kan en mer effektiv larmfunktion utvecklas, om den baseras på Android och ”molnet” istället för sms?
- Kommer en smartphone kunna utgöra ett mer mångsidigt verktyg för fjärrövervakning av system?
- Vilka krav ställs på servern för att applikationen ska fungera som tänkt?
- Vad kan det nya systemet ge för andra fördelar?
- Vad kan det nya systemet ha för nackdelar?
- Finns det fler funktioner som kan byggas in i systemet i framtiden

2. TEKNISK BAKGRUND

2.1 Android-applikationers uppbyggnad

Programutveckling i Android-miljö bygger på det objektorienterade programmeringsspråket Java. Android har en hel del egna funktioner för att underlätta utvecklingen utav applikationer till mobila enheter. En plug-in kan laddas ner till Eclipse för att underlätta vid kompilering utav applikationer, vilket dessutom innebär att Android-projekt inte behöver skapas från grunden. Vid utveckling av applikationer i Android så används ett antal olika så kallade komponenter, vilka i grund och botten är egna java-klasser från Android-biblioteket.

2.1.1 Activities – Aktiviteter

En aktivitet är komponent som visar en skärm för användaren där denne kan interagera med aktiviteten, t.ex. skicka text-meddelande, ta ett kort eller ringa ett telefonsamtal. Varje aktivitet får ett fönster för att kunna visa sitt gränssnitt, GUI. Oftast så tar gränssnittet upp hela telefon-skärmen, men de kan vara mindre än skärmen och ligga ovanför ett annat gränssnitt.

En Android-applikation är uppbyggd av en eller flera aktiviteter, oftast består applikationerna av flera aktiviteter. En aktivitet i applikationen är satt som "launcher", alltså den första skärmen som ska visas när applikationen startar. Aktiviteter kan sedan startas via andra aktiviteter, genom att användaren väljer det genom att t.ex. klicka på en knapp till en ny aktivitet eller att applikationen själv startar en. Telefonen startar bara aktiviteter själv om någonting händer, exempelvis om användaren får ett telefonsamtal eller att ett larm ringer, så kommer det upp en skärm med information om vad som händer.
Varje gång en ny aktivitet startas så stoppas den gamla aktiviteten, men den gamla aktiviteten sparas i en "back stack" (1). Varje gång en ny aktivitet startas så sparas den på stacken. Stacken använder sig av principen "last in, first out", vilket betyder att när användaren är klar med en aktivitet och går bakåt, med hjälp av en bakåtknapp i applikationen eller telefonens inbyggda, så raderas den nuvarande aktiviteten från stacken och hämtar den senaste tillagda aktivitet och kör den.

När en aktivitet har stoppats så blir aktiviteten meddelad att den har bytt tillstånd genom någon av sina callback-metoder i aktivitetens livscykel (2). Det finns ett flertal callback-metoder aktiviteten kan bli anropad via, på grund av ändring i sitt tillstånd. Det finns metoder för när en aktivitet skapas, stoppas, återupptas eller förstörs. Varje sådan metod går att modifiera så att aktiviteten utför visst arbete när den kommer i något av de olika tillstånden. När en aktivitet stoppas är det bra om allt arbete även stoppas som den aktiviteten utför för tillfället så att inte aktiviteten lever vidare och körs i bakgrunden utan användarens kännedom. Om metoderna inte läggs till eller modifieras så kommer applikationen fungera helt vanligt ändå, de metoderna kan läggas till för att hantera om någonting extra behöver göras precis innan aktiviteten stoppas eller startas. Ett exempel kan vara om aktiviteten behöver ladda hem de senast sparade värdena innan den startas, då fungerar det bra att använda en callback-metod för att utföra detta arbete på ett enkelt sätt när aktiviteten har startats, stoppats, förstörts eller återupptagits.

2.1.2 Services

En service är en applikations-komponent som, till skillnad från aktiviteter, inte har något användargränssnitt. Services används istället för att utföra bakgrundsprocesser, utan att låsa interfacet mot användaren. Ett exempel på detta kan vara att spela musik i bakgrunden utav en applikation, användaren kan då lämna applikationen för att t ex läsa ett e-postmeddelande, medan musiken fortsätter spela i bakgrunden. Något som är viktigt att poängtera är dock att en service inte per automatik startar en egen tråd, utan utför arbetet på samma tråd som applikationens övriga processer (3).

En service existerar på två former:

Started (startad) - En komponent har använt metoden startService för att köra servicen i bakgrunden under en odefinierad tidsrymd. Servicen kan därmed fortsätta köra i bakgrunden även efter det att komponenten som startade den har förstörts. Vanligtvis utför servicen en operation utan att rapportera det till någon annan komponent och stannar sedan sig själv.

Bound (bunden) - När en komponent startat en service genom att använda metoden bindService kommer dessa två komponenter kunna kommunicera med varandra genom så kallad IPC (interprocess communication), vilket kan vara önskvärt om t ex en service ska ladda ner en fil och aktiviteten som startar servicen ska visa nedladdningsförloppet i en dialogruta. Servicen kommer då bara existera så länge som en annan komponent är bunden till den.

Services kan också vara både startade och bundna, vilket möjliggör IPC men innebär att de även kör obegränsat.

2.1.3 BroadcastReceivers

BroadcastReceiver (svenska: sändningsmottagare) är en basklass för kod som kommer ta emot “intenter” som skickas genom metoden sendBroadcast (4). BroadcastReceivern registreras på en viss händelse, antingen i manifestet eller dynamiskt, och kommer sedan startas när denna händelse inträffar. Ett exempel på en BroadcastReceiver är den som “lyssnar” efter att ett textmeddelande skickas till en telefon; när detta sker kommer en BroadcastReceiver som registrerats i telefonens sms-program att starta en process som meddelar användaren om att ett meddelande tagits emot.
BroadcastReceivers kan också registreras på händelser som inte är system-händelser, alltså specifika kommandon från tredjeparts-applikationer.

2.1.4 Manifest

För att en Android-applikation ska kunna installeras på en enhet krävs det att det i rot-katalogen ligger en XML-fil med namnet “Manifest.xml”. I denna listas grundförutsättningarna som krävs för att köra applikationen, och programmeraren kan även

göra en del övriga inställningar som antingen påverkar hela applikationen eller enskilda aktiviteter (5). En del utav det som står med i Manifest.xml är:

- Namnet på applikationens java-paket. Detta ska vara en unik identifierare för applikationen.
- En lista med de komponenter applikationen består utav (aktiviteter, services m.m.), samt beskrivningar utav dessa, exempelvis om de ska startas under vissa omständigheter. Om en komponent inte står med i manifestet kommer applikationen att krascha då den försöker använda komponenten.
- En lista med eventuella tillstånd som begärs vid installation, för att applikationen ska kunna interagera med vissa delar och funktioner hos enheten.
- En lista med eventuella tillstånd som kommer krävas av andra applikationer för att få använda den berörda appens komponenter.
- Miniminivån på Android API'n hos enheten som krävs för att köra applikationen.
- Klass-biblioteken som applikationen måste länkas mot.

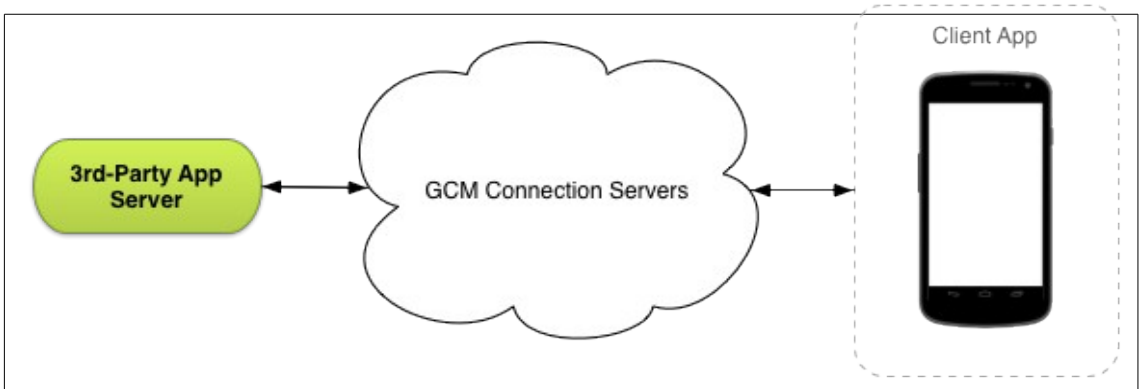
Manifestet är alltså avgörande för att applikationen ska fungera som förväntat och interagera på ett önskvärt sätt gentemot enheten.

I Eclipse skapas ett manifest automatiskt då användaren väljer att skapa ett nytt Android-projekt på "arbetsytan", men det fungerar mer som en ram för hur det ska vara utformat och användaren behöver själv fylla i det som krävs för att kunna installera och köra applikationen framöver.

2.2 Google Cloud Messaging

Google har skapat en gratistjänst som heter Google Cloud Messaging. Denna tjänst gör det möjligt att skicka data från en server, till en Android-enhet som tar emot och behandlar datan. GCM-tjänsten hanterar alla aspekter gällande meddelande-kö och leverans till den specifika Android-applikationen som körs på en eller flera utvalda enheter (8).

Då GCM ska implementeras på en server går det att välja mellan två olika anslutningstyper; HTTP och XMPP (CCS). XMPP möjliggör för den mobila enheten att kommunicera tillbaks till servern via GCM, men i detta projekt används en enklare webbserver för att skicka GCM-meddelanden, så denna dokumentation kommer endast behandla HTTP.



Figur 2.1: Illustration utav GCM-kommunikation mellan server och applikation (8)

Det som krävs för att server, applikation och mobil enhet ska "hitta varandra" i molnet är tre olika datasträngar: GCM-projektets unika sifferkod, en API-nyckel, samt den mobila enhetens applikations-ID. När ett meddelande ska skickas från servern kommer sedan en rad händelser att inträffa;
1. Applikation-servern (3:e partsservern) skickar meddelandet till GCM servrarna.
2. Google köar meddelandet och sparar det, om enheten skulle vara offline, annars skickas det till den/de berörda enheterna.
3. På enheten kommer systemet endast att sända en så kallad broadcast till den applikation och enhet som specificerats av servern som mottagare. Detta kommer att "väcka" applikationen på enheten, vilket i sin tur innebär att applikationen inte behöver vara igång för att ta emot meddelandet.
4. Android-applikationen hanterar meddelandet.

Mottagandet på den mobila enheten kan i sin tur också delas upp i steg:
1. Systemet tar emot meddelandet och extraherar eventuell data/nycklar som skickats med.
2. Systemet skickar data/nycklar till Android-applikationen i form av så kallade "extras".
3. Applikationen extraherar den data som skickas med och hanterar den enligt programmerarens specifikationer.

De meddelanden som kan skickas via GCM kan, lite förkortat, delas upp i två kategorier; "send-to-sync" och meddelande med "payload".
Ett "send-to-sync" meddelande är bara ett kort ping-meddelande som säger till applikationen att det nu finns någon data att hämta ner från servern. Ett exempel är en e-postmeddelande applikation. När användaren får ett nytt meddelande på servern så pingar servern den mobila applikationen med ett "Nytt e-postmeddelande" meddelande så applikationen vet om att det nu finns ett nytt meddelande att hämta. Det kan vara så att att servern skickar samma meddelande flera gånger, om användaren har fått flera e-postmeddelande under en period utan att applikationen haft chansen att synkronisera med servern och ladda hem de nya meddelandena. Då behöver inte applikationen spara de gamla "Nytt e-postmeddelande"-meddelandena, det räcker med att ett sådant meddelande kommer fram till applikationen.

E-postmeddelandet-exemplet är ett fall där det skulle behövas en GCM-collapse key. En sådan nyckel är en godtycklig sträng av text som används för att sortera en grupp lika meddelanden som applikationen har fått skickat till sig när enheten är avstängd eller utan täckning, så att endast det senaste meddelandet får skickas till klienten. Exempel på sådana nycklar kan vara "nya meddelanden", "uppdateringar tillgängliga" eller "senaste resultaten".

GCM tillåter ett maxantal på fyra stycken olika strängar, collapse keys, att användas av GCM-servern. Med andra ord så kan GCM-servern parallellt spara fyra stycken "send-to-sync"-meddelande per Android-enhet, de fyra strängarna är då inte likadana. Till exempel kan Android-enhet A ha fyra olika collapse key, A1, A2, A3 och A4, medan Android-enhet B kan ha fyra andra collapse key, B1, B2, B3 och B4.
Det finns möjlighet att överstiga maxantalet collapse key, men då har GCM inga garantier om vilka de fyra är som kommer fram, och vilka nycklar det är som inte kommer fram. Användaren kan då inte ens se att några nycklar inte har kommit fram.

Att ett meddelande har så kallad "payload" innebär att annan data (förutom collapse key) skickas med meddelandet. Ett GCM-meddelande kan rymma upp till 4kb data.

3. METOD

Projektet kommer att genomföras med hjälp utav programmet Eclipse med Android-plugin. Ett alternativ som övervägdes var NetBeans, men då Eclipse erbjuder många plugin-program och funktioner för både felsökning och testning utav just Android-applikationer ansåg författarna att det ger mer utrymme för att fokusera på detaljerna i själva applikationen. Dessutom kommer Git att användas för att dela filer med alla projektets deltagare, då det är den versionshanterare som redan används på företaget genom webbapplikationen redmine.
Till att börja med kommer ett simpelt användargränssnitt att upprättas i en Android-applikation, riktad mot plattformen Android 4.0 (Ice Cream Sandwich). Ett antal olika aktiviteter ska skapas; huvudmenyn (det första användaren ser när applikationen startas), översikt över övervakade objekt (en lista över de system, servrar och tjänster som kan larma), detaljvy över övervakat objekt, samt inställningar. Utifrån dessa aktiviteter kan funktionerna sedan börja utvecklas.
Då beställaren önskar att så lite data som möjligt ska sparas i applikationen, både av säkerhetsskäl och för att undvika att enhetens batteri och lagringsutrymme ska dräneras för snabbt, kommer Googles tjänst GCM (Google Cloud Messaging) att användas för kommunikation mellan server och applikation. På så vis behöver inte enheten hämta information från servern i korta tidsintervall hela tiden, utan kan vara passiv tills servern påkallar uppmärksamhet.
Applikationen kommer att testas regelbundet under utvecklingen, både offline och mot en server, för att snabbt hitta eventuella buggar.

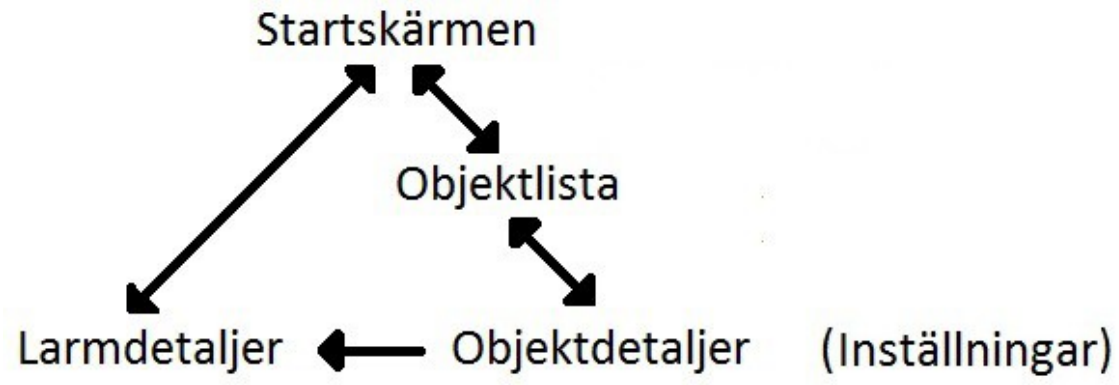
Under hela arbetsgången kommer loggbok att föras, och minst ett möte per vecka kommer att hållas med handledaren/beställaren på företaget. Versionshistorik, Gantt-schema och en lista över problem/buggar kommer att finnas på redmine, som redan används vid projekt på företaget.

4. RESULTAT

Applikationen, döpt till “IlaitApp”, fungerar i huvudsak som ett interface för att tolka data från en server, så att de enheter som kör applikationen ska behöva utföra så lite arbete som möjligt. Förutom de relativt självklara fördelar som detta medför för enhetens batteritid så innebär det också att ingen känslig data behöver lagras i applikationen, något som är extra viktigt ur säkerhetssynpunkt.

Alla aktiviteter, utom registrerings-aktiviteten och larm-aktiviteten, har en meny-bar längst upp på skärmen. Den meny-baren har fyra stycken knappar. Den första knappen som är längst till vänster (se figur 4.3) är Ilaits logo, denna kallas populärt för ”hem”-knapp. Det är vanligt att Androids applikationer har en sådan knapp. Det populära namnet på denna kan tyckas något missvisande, då den fungerar precis som en bakåtknapp, användaren kommer till den senast besökta aktiviteten. Ett undantag är startskärmen, då är den aktiviteten redan längst ner i listan över öppna aktiviteter och ingenting händer när den klickas på.
Den andra knappen från vänster som ser ut som en lista, det är knappen för att komma till aktiviteten objektlista, som visar en lista över objekt. Den knappen fungerar i alla aktiviteterna, förutom i aktiviteten objektlista själv, för användaren är redan i den aktiviteten.
Den tredje knappen från vänster som är två pilar i en cirkel är uppdateringsknappen. Den knappen fungerar bara från startskärmen och från aktiviteten objektlista. Det den gör är att hämta ner informationen från servern som behövs för respektive aktivitet. Om användare uppdaterar från startskärmen så hämtar applikationen ner alla nya larm från servern, om de finns några. Är användaren är i aktiviteten objektlista och vill uppdatera så hämtar applikationen ner en lista med alla objekt.
Den sista knappen, den som är längst till höger och ser ut som tre reglage är knappen för att ta sig till aktiviteten inställningar. Den knappen fungerar oberoende i vilken aktivitet användaren befinner sig i för tillfället. Den fungerar dock inte i sin egen aktivitet, då användaren redan befinner sig där.

Aktiviteternas sammanhållning är uppbyggd som ett träd. Med trädet så menas att första skärmen är roten på trädet, sen varje aktivitet som kan öppnas från startskärmen går neråt i trädet, och när användaren går bakåt så flyttas vyn tillbaks närmare roten i trädet. Registreringsaktiviteten och larmaktiviteten är inte med i trädet eftersom de kan användaren inte välja att gå till. Registreringsaktiviteten kommer en användare till om denna inte har registrerat sig. Larmaktiviteten aktiveras bara när ett larmmeddelande från servern skickas till applikationen, ingenting som användaren kan styra.



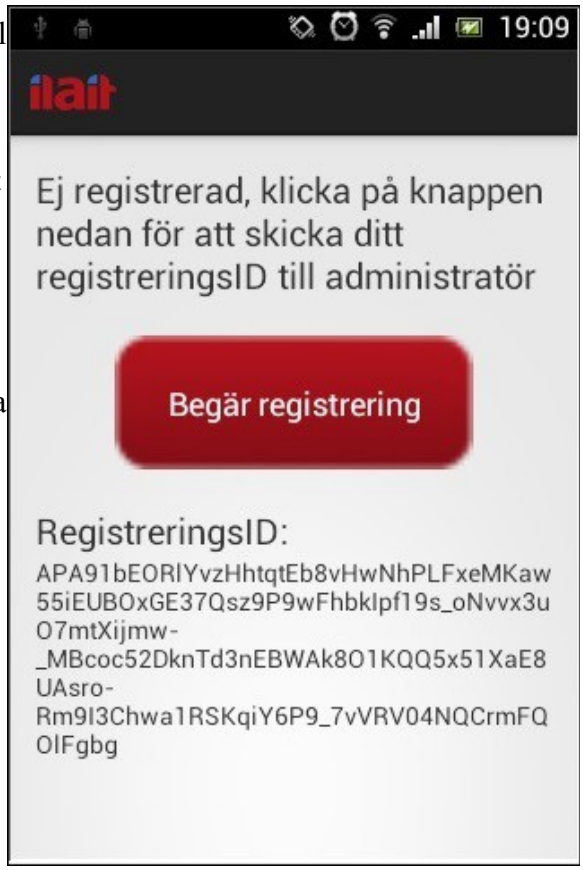
Figur 4.1: Applikationens aktivitetsträd

Enligt figur 4.1, vilken beskriver det tidigare nämnda ”trädet” så kan en användare ta sig till aktiviteten larmdetaljer via objektdetaljer eller startskärmen. Pilen mellan larmdetaljer och objektdetaljer är bara riktad åt vänster, vilket betyder att en användare som kommer via startskärmen till larmdetaljer kan inte ta sig till objektdetaljer. Men om användaren kommer från objektdetaljer till larmdetaljer så kan han backa sig tillbaks till objektdetaljer.

Anledningen att aktiviteten Inställningar inte har några pilar dragna till eller ifrån sig är att den är inte med i aktiviteternas träd-hierarki, den aktiviteten kan en användare nå från alla de andra aktiviteterna i trädet. Inställningar är inte heller med i “back stack” som de andra aktiviteterna är med i. Exempelvis om användaren går från aktivitet A vidare till Inställningar, och därifrån vidare till aktivitet B och sedan i aktivitet B klickar på någon bakåtknapp, så kommer användaren vanligtvis tillbaks till den senaste aktiviteten, som i detta fallet är Inställningar. Eftersom Inställningar nu inte är med i “back stack” så kommer användaren hela vägen tillbaks till aktivitet A. Detta åstadkoms genom att aktiviteten Inställningar termineras när användaren lämnar den.

4.1 Registrering och startskärm/huvudmeny

För att undvika att obehöriga får tillgång till Ilaits information krävs det att alla som önskar använda sig av applikationen först registrerar sin enhets registrerings-ID hos ansvariga på företaget. Innan detta har skett kommer användaren alltså endast att se skärmen som visas till höger.



Figur 4.2 Registreringsskärmen

Det som sker när användaren öppnar applikationen för första gången är egentligen att startskärmen som listar aktiva larm startas, då den är listad i manifestet som “launch activity”, men innan den fylls på med eventuella larm kontrollerar applikationen sina sparade registreringsvärden. Om användarens sparade värde är någonting annat än “REGISTERED” kommer aktiviteten som syns till höger att startas istället. Denna kontroll är det första som görs i klassen huvudmeny, och går mycket fort, så användaren kommer inte märka av aktivitetsbytet.

Skärmen som nu visas innehåller två stycken intressanta objekt; en knapp märkt “Begär registrering” och en textvy som visar användarens registrerings-ID. Registrerings-ID:n är ett unikt ID som genereras utav Google med hjälp av applikationens projekt-nummer i kombination med varje unik enhet som applikationen installerats på.

Detta ID är inte konstant, och byts ut på en ej regelbunden basis, vilket innebär att det behöver hanteras på ett visst sätt, detaljerna kring detta kommer behandlas mer utförligt i kapitel 4.8 Funktionskrav för Server.
När användaren trycker på knappen “Begär registrering” kommer en så kallad POST-begäran skickas till webbservern via HTTP. I denna POST kommer registreringsID för enheten ligga under nyckeln “regid”, och webbservern kommer avgöra om det är en registrerad användare eller ej som ligger bakom förfrågan. Så länge ingenting gick fel vid själva begäran kommer användarens registreringsvärde sättas till “AWAITING_REGISTRATION”, och texten på knappen bytas ut till “Inväntar godkännande, klicka för att uppdatera”. Nu har alltså en begäran kommit in till servern, och här kan administratören välja hur denna ska behandlas på bästa sätt, men förutsatt att begäran accepteras så kommer registreringsID för enhet att sparas på webbservern. När då användaren trycker på knappen nästa gång kommer svaret från servern vara HTTP-kod 200 (OK), och registreringsvärdet kommer sättas till “REGISTERED” för att sedan starta upp huvudmenyn.

I huvudmenyn i applikationen finns den mesta informationen för användaren. Längst upp till vänster är det en textrad där det visas med ett Ja/Nej om användaren är på jouren eller inte. Det är bra att visa om användaren är på jouren eller inte, för att det kan hända att någon användare blir inloggad på jouren utav en administratör eller utav servern, utan att ha begärt det själv, då underlättar det för användaren att lätt kunna se om denna är på jouren eller inte för tillfället.
Användaren kan annars klicka på knappen “Logga in” längst upp i högra hörnet för att skicka en förfrågan till servern om att bli inloggad på jouren. När användaren har klickat på knappen så ändras texten på knappen till “Väntar..:”. Servern svarar sedan genom att antingen acceptera inloggning eller att neka den. Ifall inloggningen av någon speciell anledning skulle nekas så ändras texten på knappen tillbaks, om den accepteras så byts texten till “Logga ut” så att användaren vid ett senare tillfälle kan logga ut från jouren.

Figur 4.3 Huvudmenyn/Startskärmen

Det är värt att poängtera att en användare som enligt applikationen inte har loggat in på jouren ändå kan vara det enligt servern, och få jour-larm som vanligt. Detta eftersom det alltid är webbservern som avgör vilka enheter som ska erhålla larm, och vad för larm dessa ska räknas som. Detta kommenteras ytterligare i slutsatsen.
Knappen längst ner till vänster på skärmen som det står “Inloggade” på, används för att se vilka andra användare på företaget som är inloggade på jouren vid tillfället. När en användare klickar på knappen så kommer det fram en dialogruta med all personal som är inloggade för tillfället.

Längst ner till höger visas det hur länge sedan användaren hade en godkänd kontakt med servern, den skrivs som hh:mm:ss, alltså i detta fallet 2 min och 16 sekunder sedan. Med "godkänd kontakt med servern" menas i detta fall att applikationen har mottagit en statuskod 200 (OK) från servern.

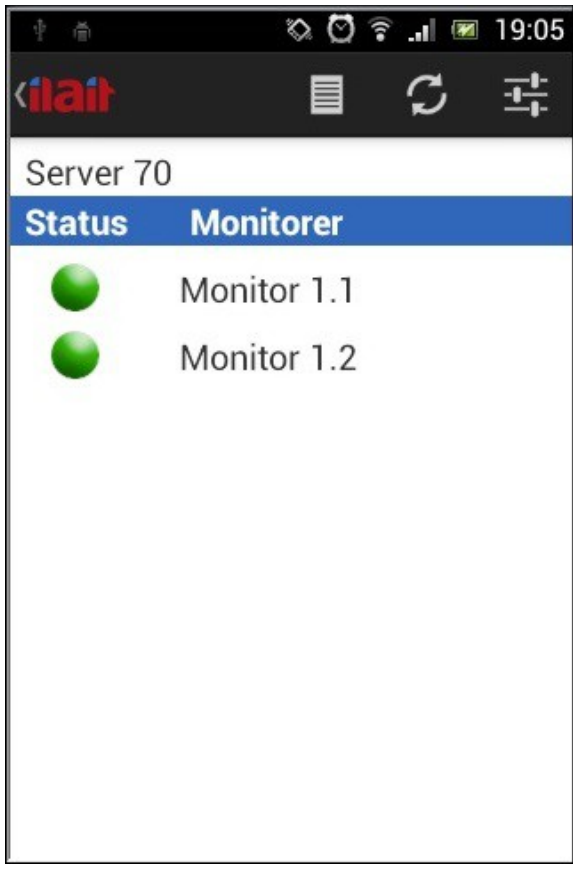
I mitten av huvudmenyn visas alla nya larm, alltså de larmen som inte ännu har blivit åtgärdade. Det finns två kolumner som visar lite information angående larmet, det är "Tid" och "Larm". Tiden skrivs ut som först datum och sedan ett klockslag. Det översta larmet larmade alltså den 7e Oktober, klockan 19:01.
Alla larmen i listan sorteras på datum och klockslag, så det senaste hamnar alltid längst ner och det nyaste larmet ska ligga längst upp så den syns tydligast för användaren. Under kolumnen "Larm" så står det listat vad larmet heter. Alla rader med larm är klickbara för användarna. En användare kan klicka på en rad med ett larm för att komma till aktiviteten Larmdetaljer, som beskrivs nedan, för att se mer specifik information angående det markerade larmet.

4.2 Lista på övervakade objekt

Aktiviteten objektlista, som syns i figur 4.4, listar de olika objekten som Ilait har. För att få fram några objekt i listan så måste användaren skriva in någonting i sökrutan som finns längst upp i aktiviteten, och sedan är det bara att klicka på sök. När användaren har klickat på sök så söker applikationen igenom alla objekt på servern och ser om den hittar några objekt som matchar med sökordet. Har användaren tryckt på uppdatera-knappen innan en sökning gjorts finns redan hela listan på objekt sparad i en så kallad hashtabell (se beteckningar för förklaring), och denna kan sökas igenom mycket snabbt. Sökfunktionen är gjord så att den skriver ut alla resultat den hittar, om de är färre än 30. Om det inte blev några träffar alls så kommer det fram en text på skärmen där det står att den inte hittade några matchande sökord. Är det fler träffar än 30 så kommer det upp en ruta som förklara för användaren att inte alla resultat som matchade sökordet skrevs ut, då skriver den bara ut de 30 första objekten. Den funktionen är till för att om det är väldigt många objekt som passar in på det användaren söker på så ska listan inte bli väldigt lång och jobbig för telefonen att skriva ut. Listan skrivs ut med namnet på objektet under "Övervakade objekt" och dess status under "Status". Statusen är en grön prick ifall ingenting är fel med respektive objekt, annars är det en röd prick.

Figur 4.4 Lista på objekt

En användare kan klicka på något av objekten för att öppna upp en ny aktivitet med mer information om det specifika objekten. Den aktiviteten användaren kommer till då är Objektdetaljer, som syns till höger. Den aktiviteten är till utseendet väldigt lik objektlista. Istället för sökraden längst upp på skärmen så har den en textrad som visar namnet på det objektet användaren klickade på. Aktiviteten Objektdetaljer har, precis som objektlista, två stycken kolumner, en för status och en för monitorer. Statusen visar en grön prick om det inte finns några larm för tillfället för det valda objektet, annars visar det en röd prick. Kolumnen för monitorer visar de monitorer ett objekt har. Ett objekt kan ha en eller flera monitorer.
Ifall det hade varit någon monitor som hade haft någonting fel, och så vis ha en röd prick som status, så blir den raden klickbar. Det betyder att det går att klicka på den raden och få upp aktiviteten Larmdetaljer, som visar lite mer detaljerad information om det larmet.



Figur 4.5 Detaljerad vy för objekt

4.3 Hantering av inkommande larm

Servern kan skicka ut två stycken olika larm. Det ena larmet är till alla som är inne på jouren, och tanken är att de som är inne på jouren ska få alla larm under tiden de är inne på jouren, och det andra larmet är till de som får larm men som inte är inne på jour. Det kan exempelvis vara om någonting går fel som har med en specifik person arbete att göra, så kan den personen få det specifika larmet, även personen i fråga inte är inne på jouren. Det är helt upp till servern att skicka ut larm till de användarna som ska få larmen, applikationen tar bara emot larmen som servern skickar. Applikationen ser inte om användaren är inne på jouren eller inte, det applikationen kan göra är att spela rätt volym vid rätt typ av larm.

När en användare får ett inkommande larm så visas skärmen som är upp till höger på nästa sida. Aktiviteten visar med stor text vad klockan är när larmet inträffat, vilket underlättar om exempelvis en användare får ett larm mitt i natten, då det kan vara bra för användaren att få se vad klockan är. Aktiviteten visar också en liten textrad som kort förklarar lite information om meddelandet, det kan exempelvis vara hur många larm det är, eller vad det är för något larm. Aktiviteten har bara en stor röd knapp med texten “Okej”. Klickar användaren på den knappen så stänger larmet av sig och användaren får upp startskärmen för applikationen.

Det kommer också upp en liten ikon, i form av Ilaits logo, i telefonens så kallade “actionbar”, vilket är den rad längst upp på skärmen (där exempelvis tiden och nya meddelanden vanligen visas). Denna ikon fungerar som en indikation om att ett larm inträffat, för de tillfällen då användaren inte märkt av larmet. Trycker användaren på ikonen startas applikationen och visar de senaste larmen.

Figur 4.6 Meddelande vid larm

Om en användare får ett larm när enhetens skärm är avstängd så startas och syns aktiviteten på en gång. Automatisk så tänds enhetens skärm, skärmlåset stängs av, alltså det passeras utan att användaren behöver fylla i lösenordet och aktiviteten håller skärmen tänd så länge som larmet låter. Det är en stor fördel för att larmet ska synas ordentligt när det är igång, även om telefonskärmen är släckt. Om användaren vill stänga av larmet fort utan att det ska låta så länge, så behövs det inte skrivas i något lösenord, vilket gör att det går mycket fortare för användaren.
Det medför inga risker att avaktivera en användares telefon-lösenord. Direkt efter användaren har stängt av larmet, och han vill använda telefonen, så måste lösenordet fyllas i, alltså kan ingen annan som inte kan lösenordet till telefonen komma in på den, även om denna person får ett larm som avaktiverar lösenordet, eftersom det bara ignorerar skärmlåset under denna aktiviteten och aktiverar det igen direkt efteråt.

Ett krav Ilait hade på applikation var att larmets ljudvolym skulle öka successivt under tiden det låter. Det är en stor fördel om en användare får ett larm när denna sover, då kan de vara skönare att vakna upp med lite tyst volym, istället för att den ska gå igång på maxvolym på en gång. Var femte sekund så ökar larmets ljudvolym med en enhet tills att volymen har nått upp till användarens personliga ljudvolym för den typ av larm som spelas. Det finns volymenheter mellan 0 och 7, vilket betyder att det tar 35 sekunder innan ljudvolymen kommer till maxvolym, ifall användaren har ställt in maxvolym under inställningar.

4.4 Användarinställningar

Figur 4.7 Volym-inställningar

Från alla aktiviteter finns det möjlighet att gå till aktiviteten inställningar, till höger, genom att trycka på ikonen längst upp i högra hörnet på skärmen (som ser ut som tre reglage). Här finns de personliga inställningarna som en användare kan välja mellan. Det finns två stycken olika ljudvolymer som är justerbara. Den ena ljudvolymen är för de larm användaren får när han/hon är inloggad på jouren, den andra volymen är för larm som erhålls vid övriga tillfällen. Det kan vara så att någon användare ska få en viss typ av larm, även om denna inte är inne på jouren, då ska det finnas möjligheten att sänka, eller höja, den volymen för att det är inte lika viktigt att han ser larmet på en gång.
Om exempelvis en administratör vill få ett meddelande för alla larm som inkommer, kan det vara behagligt att stänga av ljudet så han slipper bli väckt på natten. Användarna får också möjligheten att stänga av eller sätta på vibrationen för inkommande larm.

Alla privata inställningar sparas sedan i telefonen och laddas varje gång som användaren startar applikationen. De laddas också in innan larmet går igång, så att larmet vet vilken volym som ska spelas. Grundinställningarna, inställningarna som applikationen har när den installeras, är volymenhet 5 av 7 vid jourlarm respektive volymenhet 0 av 7 vid larm när användaren inte är inne på jouren och vibratorn är avstängd. Med andra ord ganska hög ljudvolym när användaren är inne på jouren, men annars ljudlöst.

När en användare vill ändra sina privata inställningar är det bara att dra i något utav volymreglagen, precis som det är på telefonens vanliga ljudvolym. Vibratorn slås av eller på genom att klicka på knappen AV/PÅ.
När inställningarna är klara så finns det antingen knappen “OK” eller “Avbryt” att klicka på. Båda fungerar som en bakåtknapp, alltså de går tillbaks till aktiviteten användaren kom ifrån. Enda skillnaden är att “OK”-knappen sparar dina inställningar, ifall användaren har ändrat några medans “Avbryt”-knappen inte sparar inställningarna utan går tillbaka till inställningarna som var innan användaren gick in till aktiviteten.

4.5 Larmdetaljer

Figur 4.8 Detaljvy för larm

Aktiviteten larmdetaljer, som syns i figur 4.8, är en aktivitet som användaren kan gå in på för att kolla på detaljerad information om specifika larm. Den aktiviteten skriver ut på skärmen vad det är för monitor som har larmat, vad det är för objekt monitorn tillhör och information angående larmet. Det är upp till servern om vad det ska vara för text som står under "Information om larmet". Den här aktiviteten kan en användare komma till genom att klicka på ett larm som visas i listan på huvudmenyn. Det går också att nå aktiviteten genom att en användare går in till aktiviteten Objektlista, den aktiviteten beskrivs nedan, och hittar vilket objekt det är som larmat.

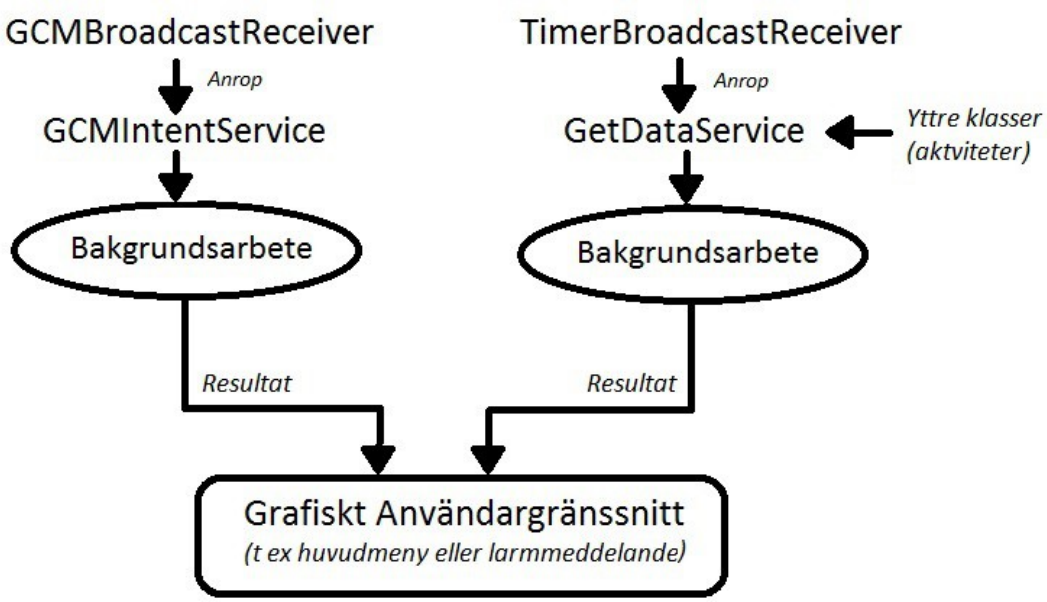
4.6 Bakgrundsklasser

Med en bakgrundsklass menas en klass som inte har ett GUI, alltså ingen aktivitet. Den klassen gör sitt jobb i bakgrunden. I applikationen används det fyra stycken bakgrundsklasser. Två stycken är BroadcastReceiver och två stycken är Services.

En av klasserna heter GCMBroadcastReceiver, och som det låter på namnet är det en BroadcastReceiver. Det är den klassen som lyssnar efter nya GCM-meddelande och tar emot dem. Det är en väldigt liten klass, det enda den gör är att ta emot nya GCM-meddelande och kollar om användaren har statusen "REGISTERED". När klassen får ett nytt meddelande från GCM så kollar klassen om användaren är registrerad eller inte, genom att hämta ett sparat värde som indikerar huruvida registreringen har utförts. Om användaren inte är registrerad så gör klassen ingenting, detta för att undvika att en användare utan registrering tar emot några larm, men om användare är registrerad så skickar den vidare allt jobb till GCMIntentService, och sedan stoppar sig själv, för att lyssna efter nästa GCM-meddelande.

När arbetet kommer till klassen GCMIntentService, som är en Service, så tar den hand om det inkomna GCM-meddelandet. GCMIntentService kollar vad det är för meddelande och utför lämpliga operationer efter det. Är det ett meddelande som kräver att larmet ska gå igång, så anropas larmet, krävs det att det ska komma en notifikation till telefonen så anropar den funktionen som gör det. På så sätt kan alltså resultatet som GCMIntentService tagit fram i bakgrunden rapporteras till användaren genom en lämplig aktivitet, så som visas i figur 4.9.

En annan bakgrundsklass är TimerBroadcastReceiver, som är en BroadcastReceiver. Det den klassen gör är att försöka ta kontakt med servern på regelbunden basis. Den här klassen finns med för att användaren ska kunna veta om telefonen har kontakt med servern eller inte. Det visas på startskärmen hur länge sedan senaste kontakt med servern var. Klassen har en timer på 20 min, när den tiden har gått ut så försöker den ta kontakt med servern ett antal gånger. Om något av de försöken lyckas så startar timern om sig, om alla försökt misslyckades så kommer klassen att anropa larmet så att användaren får reda på att kontakten med servern är dålig för tillfället. Klassen larmar användaren bara om denna är inne på jouren, då det är nödvändigt att veta ifall kontakten är dålig. Är användaren inte på jour behövs det inte larma att applikationen inte har kontakt med servern, för då det är inte lika viktigt för användaren att ha kontakt med servern, då denna inte behöver få larmen. Timern på 20 min startar om sig varje gång telefonen har fått kontakt med servern på annat vis, exempel om användaren själv hämtar någonting från servern.



Figur 4.9 Applikationens bakgrundsklasser och deras uppbyggnad

Den sista klassen är GetDataService, det är en Service som tar kontakt med servern och hämtar de filer som önskas av användaren. Det är den klassen som anropas när en användare klickar på uppdateringsknappen för att hämta de senaste uppdateringarna från servern. Den här klassen kollar först vad det är för fil den ska hämta, sedan försöker den hämta ner den filen. I figur 4.9 illustreras detta med att TimerBroadcastReceiver såväl som ”yttre klasser” kan anropa GetDataService, ett exempel på en sådan klass är aktiviteten som utgör huvudmenyn.

4.7 Testning

Likt all annan programvara krävs det utförliga tester för att kunna lita på att inga kvarvarande fel kommer ställa till problem i framtiden. Dessa är aldrig en garanti för att ett program ska fungera felfritt, men desto utförligare de är, desto mindre blir risken för dåliga användarupplevelser.

Tre olika testningsmetoder användes för IlaitApp, nedan beskrivs de var för sig.

4.7.1 Manuell testning (förväntad användning)

Till att börja med utfördes manuella tester utav applikationen på en regelbunden basis. Detta innebär att funktioner, metoder och klasser testas genom att helt enkelt installeras på en enhet eller emulator och sedan användas så som de förväntas hanteras utav användarna. Dessa tester innehar ingen speciell struktur och ger sällan några konkreta data kring applikationen, men däremot en bra känsla för om det finns omedelbara fel att åtgärda. Det verktyg som användes mest vid den manuella testningen i detta fall var Android SDK's LogCat, som presenterar de utskrifter som applikationen ger upphov till, och på så sätt i vissa fall kan ge vägledning om vad som orsakar kritiska fel. En mycket simpel webbserver, uppbyggd i Googles egna språk Golang med GCM implementerat, användes för att kontrollera att meddelanden kom fram och larmfunktionen fungerade som avsett.

4.7.2 UI/Application Exerciser Monkey

UI/Application Exerciser Monkey (refereras hädanefter till som "The Monkey") är ett program som körs på den enhet eller emulator testen ska utföras genom. The Monkey skickar ett valt antal pseudo-slumpmässiga kommandon till användargränssnittet, därav namnet (svenska: Apan); det påminner om att släppa lös en apa som trycker och drar på skärmen. De största skillnaderna mellan The Monkey och en verklig apa är dock att The Monkey kan utföra ett tusental kommandon på några sekunder och rapporterar sedan till testaren om något kritiskt fel inträffade. Detta är ett bra test utav applikationens gränssnitt, ett så kallat "stresstest", där programvaran måste klara av den svåraste formen av användande; sådan som programmeraren inte har förväntat sig.

The Monkey kördes på IlaitApp med 500, 1000 och 2000 kommandon utan några inträffade fel.

4.7.3 Eclipse Memory Analyzer Tool

Ett vanligt problem vid utveckling utav Android-applikationer är att ingen hänsyn tagits till det relativt lilla utrymme som en applikation har att röra sig med. Android-applikationer får som mest använda sig utav 16 MB RAM-minne när de körs (i vissa nyare enheter kan upp till 40 MB användas, men detta bör inte användas som en förutsättning vid utveckling). För att utvecklare inte ska behöva ägna större delen av sin tid åt minneshantering har Android en inbyggd funktion, "Garbage Collector", som vid behov tar bort objekt som inte används (6). Detta innebär dock inte att utvecklare kan släppa tanken på minneshantering helt, för det finns fortfarande fällor att falla i, som i slutänden kan innebära att applikationen slutar fungera då operativsystemet kommer stänga ner de applikationer som tar upp mer minne än tillåtet. Mycket frekventa "Garbage Collections" bör också undvikas, då det kommer göra applikationen mycket långsam, när den måste använda stora delar av sina resurser åt att slänga gamla objekt. Varje seriös utvecklare bör därför alltid göra en minnesanalys utav den egna applikationen, innan den släpps för användning.

I Eclipse, med Android SDK installerat, finns redan verktyg för att se applikationers minnesanvändning, men de är inte alltid så tydliga i de fall en så kallad "minnesläcka"

förekommer. Då kan det istället vara fördelaktigt att installera en plug-in till Eclipse; Eclipse Memory Analyzer Tool (7). Med detta verktyg får utvecklaren en mer översiktlig bild utav minnesanvändningen i applikationen, i form av diagram och lista på "misstänkta" objekt, med möjligheten att gå in närmre på detaljerna vid behov.

Detta verktyg användes för att göra en utförlig analys utav IlaitApp, och resultatet var klart godkänt. De objekt som tog mest plats var nödvändiga och utformade på bästa sätt, vilket innebär att användning utav applikationen inte utgör någon risk för "överbelastning" utav minnet.

4.8 Funktionskrav för server

För att applikationen ska leva upp till de krav som ställs på den krävs det att även servern som larmar om fel uppfyller vissa funktioner och svarar på ett visst sätt. Nedan listas dessa krav, som tagits fram efter hand som applikationen skapats.

Grundläggande

- HTTP-webbserver
- GCM ska vara implementerat
- Ska ha sparat API-nyckeln för projektet

Registrering

- Vid en POST-begäran till *serveradressen*/reg/ sparas det UTF-8 kodade värdet med nyckeln "regid", om det är okänt, och svarskod 401 (Unauthorized) presenteras till den "frågande" enheten
- Om POST-begäran till *serveradressen*/reg/ innehåller ett känt registreringsID presenteras svarskod 200 (OK)

Alarm

- Vid nytt larm ska en POST-begäran skickas, formaterad enligt JSON, till GCM-servern med API-nyckeln och registrerings-id för de användare som ska ta emot meddelandet inkluderat i headern, samt ”onCall” eller ”notOnCall” som "collapse_key" (beroende på vem som räknas som jourhavande för larmet/tillfället)

Jour

- Vid en POST-begäran från godkänt registreringID till *serveradressen*/jour/login/ eller *serveradressen*/jour/logout/ ska svarskod 200 (OK) skickas tillbaka och begäran om in- eller utloggning från jouren ska behandlas
- Godkänd in- eller utloggning, alternativt forcerad in- eller utloggning, ska ge upphov till en POST till GCM-servern, liknande den vid larm men med collapse_key:n "jour" och meddelandet "Logged in" respektive "Logged out", skickad till den berörda enheten
- Icke godkänd in- eller utloggning ska ge upphov till ett GCM-meddelande likt det ovan, men med meddelandet "Denied login" respektive "Denied logout"

Läsning

- Begäran om att få läsa från servern kommer alltid att ske i POST-form, och servern måste jämföra det medskickade registreringsID:t mot de som finns sparade bland godkända enheter. Ej godkänt ID ska ge upphov till statuskod 401 (Unauthorized)
- Servern ska svara på tre typer av förfrågningar om läsning i JSON-form; *serveradressen*/objectList/, *serveradressen*/errors/, samt *serveradressen*/jourUsers/
- JSON-objekten i objectList ska innehålla nycklarna: “name” (objektets namn), “id”, “status” (kan bara vara “ok” eller “alarm”) och “monitors”. Nyckeln “monitors” kan innehålla ett eller flera objekt med nycklarna “name” och “status” (samma villkor som ovan)

- JSON-objekten i errors ska innehålla nycklarna: “name” (den berörda monitorns namn), “id” (den berörda monitorns ID), “time” (tiden och datumet då felet inträffade, på formen ÅÅÅÅ-MM-DD*T*HH:MM:SS+01:00), “status”, “object” (objektet på vilken monitorn befinner sig) och “body” (evetuella felmeddelanden)
- Servern ska svara på förfrågningar till serveradressen/jourUsers/ med en lista på de som för tillfället är inloggade på jouren

Slutligen är det en mycket viktig faktor som servern måste ta hänsyn till; registreringsID:n är inte helt konstanta. När servern skickat ett meddelande till GCM’s servrar så ska den även få ett svar. I detta svar står det bland annat hur många meddelanden som kom fram av de som skickades, men också hur många “canonicals” det blev. Dessa är de ID:n som förnyats, och detta betyder att framtida meddelanden kanske inte kommer fram om servern fortsätter skicka till samma ID:n. Med i meddelandet från GCM finns dock vilka ID detta gäller och hur de nya ser ut, så det som krävs är helt enkelt att servern läser av i meddelandet från GCM om det finns canonicals och därefter byter ut dessa. Meddelandet som gav upphov till canonicals har alltid kommit fram, så det behöver inte skickas igen. Mer om detta kan läsas i Googles egna dokumentation kring GCM [9].

5. SLUTSATS

5.1 Diskussion

Projektet har resulterat i en Android-applikation som kan ersätta det gamla systemet för larmhantering, och dessutom erbjuda användarna ännu mer funktionalitet för övervakning utav Ilaits system. Applikationen behöver inte kompletteras innan användning, men möjligheten till utbyggnad av applikationens funktioner (t ex med detaljerad övervakning utav monitorer) finns tillgänglig för Ilaits personal.

Projektets beställare ville ha en applikation som är pålitlig, men inte riskerar att dränera batteriet på enheten som installerat den, därför lades förslaget fram om att använda GCM för kommunikation. Ett alternativ till GCM är att helt enkelt låta applikationen koppla upp enheten mot servern genom internetuppkopplingen, och via en webbapplikation på servern kontrollera om nya larm inkommit. Detta skulle dock innebära, förutom att enhetens batteri dräneras hastigt på grund utav ständiga, krävande nätverksoperationer, att det blir en viss fördröjning i larmen eftersom enheten inte kan vara uppkopplad hela tiden. Med detta i åtanke blev GCM ett självklart val för projektgruppen.

Larmfunktionen kan definitivt anses mer effektiv i det nya systemet, framför allt då fler än en person nu kan erhålla larm, vilket tidigare inte var möjligt. Dessutom bör driftpersonalen uppleva en ökning av andelen effektivt arbete, då den övervakande funktionen innebär att de kan skaffa sig en uppfattning om problemet snabbare och därmed börja bearbeta det tidigare.

5.2 Kritisk diskussion

En funktion som övergavs mot slutet av arbetet var knappen som skulle expandera en lista med larmhistoriken. Denna funktion ansågs aldrig som viktig för applikationen, och när den till slut hade prioriterats ned till slutet av listan med funktioner bestämdes det, tillsammans med beställaren, att den lika gärna kunde tas bort helt. Hade en annan arbetsmetodik använts, där applikationens olika delfunktioner specifierats mer utförligt innan arbetets start, så hade troligtvis denna funktion funnits med i applikationen i slutändan, men projektdeltagarna och beställaren valde att använda sig utav scrum istället för att vara mer säkra på att en färdig applikation skulle vara slutresultatet. I planeringsrapporten (appendix A) har larmhistoriken tagits med i Gantt-schemat som användes för planering utav projektet, men i slutändan lades alltså mer tid på de övriga funktionerna och slutgiltig testning istället.

I den slutgiltiga applikationen kan användare loggas in eller ut från jouren mot sin vilja, något som knappast ses som användarvänligt. Här prioriterades istället företagets krav framför individens, vilket är något ovanligt i dessa sammanhang. Det ställer i sin tur höga krav på administratörer på företaget i form av det ansvar som de då får över de anställdas välbefinnande. Samtidigt ställer applikationen också krav på att de anställda sköter sig

gentemot företaget; ingenting kan stoppa en anställd från att sänka volymen på jourlarm till noll (de ges endast en varning i meddelandefältet på enheten) eller stänga av enheten helt för att slippa erhålla larm. Det antas helt enkelt att både företag och anställda sköter sig gentemot varandra, för att en viss frihet ska finnas för individuella anpassningar av applikationen.

Slutligen finns ett specialfall som inte har hanterats av utvecklarna, och därför måste hållas i åtanke vid utvecklande av webbservern; trasiga meddelanden från servern kan inte alltid upptäckas. Detta innebär att om ett meddelande från servern är trasigt kommer det bara upptäckas då det försvunnit efterfrågade bitar, har däremot ett helt JSON-objekt tappats men övriga är hela så kan inte detta upptäckas.

5.3 Miljöaspekter

Vid första åsyn kan det tyckas svårt att se ett projekt som detta ur miljösynpunkt - det är ju trots allt "bara" programkod - men det finns trots allt några miljöfördelar som detta arbete resulterat i.
Först och främst har programmets upplägg en energisparande funktion, då mycket hänsyn har tagits till mobila enheters batteritid. Tack vare att GCM används som meddelande-funktion behöver inte enheten ständigt koppla upp sig mot internet för att kontrollera om någon händelse inträffat, signalen från GCM kommer istället att "väcka" enheten vid ett larm. Applikationen fungerar dessutom bättre än det nuvarande sms-systemet i detta avseende, då många larm inte behöver generera en uppsjö av textmeddelande som dränerar batteriet genom att låta och vibrera mer eller mindre konstant, utan kommer endast ge upphov till ett larm som låter under en viss tid och sedan är tyst tills användaren kollar vad som hänt (alternativt börjar om då en viss tid har gått).
Med denna applikation kommer jourhavande driftpersonal dessutom kunna få mer information om varje larm via sin telefon än vad som tidigare var möjligt. Detta innebär också en del energibesparing, då ett mindre eller självåtgärdande fel kan uppmärksammas utan att användaren behöver starta en dator.

REFERENSER

1. Activities (API Guides), Android Developers. Android Open Source Project, Google, 2014, http://developer.android.com/guide/components/activities.html (Acc 2014-01-12)

2. Activity (Reference), Android Developers. Android Open Source Project, Google, 2014, http://developer.android.com/reference/android/app/Activity.html (Acc 2014-01-12)

3. Services (API Guides), Android Developers. Android Open Source Project, Google, 2014, http://developer.android.com/guide/components/services.html (Acc 2014-01-15)

4. BroadcastReceiver (Reference) Android Developers. Android Open Source Project, Google, 2014, http://developer.android.com/reference/Android/content/BroadcastReceiver.html (Acc 2014-01-15)

5. App Manifest (API Guides), Android Developers. Android Open Source Project, Google, 2014, http://developer.android.com/guide/topics/manifest/manifest-intro.html (Acc 2014-01-16)

6. Managing Your App's Memory (Training), Android Developers. Android Open Source Project, Google, 2014, http://developer.Android.com/training/articles/memory.html (Acc 2014-01-16)

7. Investigating Your RAM Usage (Tools), Android Developers. Android Open Source Project, Google, 2014, http://developer.Android.com/tools/debugging/debugging-memory.html (Acc 2014-01-18)

8. Google Cloud Messaging Overview (Google Services), Android Developers. Android Open Source Project, Google, 2014, http://developer.android.com/google/gcm/gcm.html (Acc 2014-01-20)

9. GCM Advanced Topics (Google Services), Android Developers, Android Open Source Project, Google, 2014, http://developer.android.com/google/gcm/adv.html (Acc 2014-03-03)

A. PLANERINGSRAPPORT

Tidplanen för projektet presenteras nedan i form utav ett Gantt-schema.