Emotivation

En applikation för att öka Hälsa och Känslomässig Medvetenhet

Examensarbete inom Högskoleingenjörsprogrammet i datateknik

HENRIK MERLANDER
GUSTAV SIMONSSON

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2014

Emotivation
En applikation för att öka Hälsa och Känslomässig medvetenhet

Henrik Merlander, Gustav Simonsson

Institutionen för data- och informationsteknik
Chalmers tekniska högskola
412 96 Göteborg
Tel: 031-772 1000
Fax: 031-772 3663

[Omslagsbilden är den logotyp som använts för applikationen]

Institutionen för data- och informationsteknik
Göteborg, 2014

Innehållsförteckning

Sammanfattning

I denna rapport behandlas ett projekt som utökar en existerande Androidapplikation. Syftet med den befintliga applikationen är att erbjuda ett utbud av olika verktyg som hjälper människor att förbättra sin hälsa. De verktyg som fanns i den existerande applikationen använde sig av en lokal databas och har i detta projekt modifierats så att alla databasoperationer sker mot en central databas. För att låta administratören uppdatera innehållet i applikationen skapades en desktopapplikation. Desktopapplikationen kopplar upp sig mot databasen och tillåter redigering av de frågeformulär Androidapplikationen använder sig av när den ställer frågor som berör hur användaren upplevt sina känslor den senaste tiden. Det har skapats en känslokarta där användaren kan markera var i kroppen hon har känt förnimmelser i samband med en känsla. Sedan tidigare fanns en ljudspelare för att spela upp lokalt lagrade guidade meditationer. Den har vidareutvecklats och kan nu strömma ljudfiler från en server. Alla verktyg har givits möjlighet att användas offline, detta behandlas på olika sätt beroende på hur verktyget fungerar.

Summary

This report covers a project that involves further developing of an existing Android application. The purpose of that application was to offer a variety of tools in order to help people improve their health. The tools that existed in the application used a local database. In this project the tools have been modified to instead use a central database. A desktop application was created to let the administrator easily update the content of the application. It connects to the database and allows modification of the questionnaire that the Android application uses to ask the user about her recent emotional experiences. An emotion-map was created, where the user can highlight parts of the body where she has been feeling strong sensations in conjunction with an emotion. There existed a sound-player that was able to play locally stored guided meditation tracks. It has been further developed and is able to stream files from a server. All the tools have been given the ability to be used offline in some way, this has been achieved differently depending on the tool.

Ordlista

Android SDK - Android Software Development Kit. Ett utvecklingsverktyg för att bygga, testa och felsöka Androidapplikationer.

(REST)-API - (Representational State Transfer)-Application Programming Interface. Ett programmeringsinterface som beskriver hur man kontaktar en extern service.

ARGB - Alpha Red Green Blue. Ett sätt att representera färger i bitar.

ASCII - American Standard Code for Information Interchange. En sätt att representera tecken i datorer.

BSON - Binary JavaScript Object Notation. Serialiserad JSON-data.

CSS - Cascading Style Sheets. Ett sätt att beskriva presentationen för ett HTML-dokument.

DOM - Document Object Model. En konvention för att representera och interagera med objekt i olika typer av dokument.

EEG - Elektroencefalografi. En metod för att registrera hjärnans aktivitet med hjälp av elektroder.

ER - Entity-Relationship. En datamodell som beskriver sambandet mellan enheter i en databas.

GUI - Graphical User Interface. Ett grafiskt användargränssnitt.

Href - HyperText Reference. Ett attribut i en HTML-tagg som representerar URL:en dit man ska föras när man klickar på ett objekt.

HSSF - Horrible SpreadSheet Format. Apaches Java-implementation av Excel-formatet.

HTML - HyperText Markup Language. Språket som webbsidor skrivs i.

JDBC - Java Database Connection. Ett API för kommunikation med SQL-databaser från Java.

JobManager - En Javaklass som används för att köa operationer.

JSON - JavaScript Object Notation. Ett format som beskriver objekt med hjälp av text.

JVM - Java Virtual Machine. En process som kan exekvera Java-byte kod. Krävs för att köra Javaprogram.

Microsoft IIS - Microsoft Internet Information Services. Webservermjukvara från Microsoft.

nginx - Engine-X. Webservermjukvara.

NoSQL - Not Only SQL. Ett sätt att lagra data på andra sätt än det som används i en relationsdatabas.

PHP - PHP: Hypertext Preprocessor. Ett skriptspråk som används för webutveckling på serversidan.

PNG - Portable Network Graphics. Ett digitalt bildformat som inte komprimerar bilden.

POI - Poor Obfuscation Implementation. Ett projekt som tillhandahåller Javabibliotek för diverse Microsoft Office-format.

SQL - Structured Query Language. Ett programspråk designat för att hantera data i en relationsdatabas.

SWT - The Standard Widget Toolkit. En verktygslåda för att bygga grafiska användargränssnitt.

UI - User Interface. Användargränssnitt.

URL - Uniform Resource Locator. En sträng som representerar en resurs på internet.

XAMPP - Cross-Apache HTTP Server-MySQL-PHP-Perl. En webserverlösning som innehåller Apache HTTP-server, MySQL och diverse skriptspråk.

XLS - Excel Binary File Format. Ett filformat för Microsoft Excel.

XML - Extensible Markup Language. Ett märkspråk.

1. Inledning

1.1 Bakgrund

Människor är beroende av att vara friska och välmående för att de skall kunna arbeta och driva samhället framåt. Psykisk ohälsa är enligt socialstyrelsens folkhälsorapport en av de vanligaste anledningarna till sjukersättning och frånvaro från jobb eller skola [1]. Detta är direkt skadligt för individen och kostar samhället och arbetsgivare mycket pengar i form av minskad arbetskraft. Empaticus är ett företag som med hjälp av socialt och emotionellt lärande vill få människor att öka sin förståelse och insikt i sitt inre, och på det viset skapa bättre relationer till omvärlden. Ett verktyg som Empaticus vill använda i sitt arbete är en mobilapplikation där det ska vara möjligt att lära sig att känna igen sina känslor och förstå varför de uppstår genom att meditera och dagligen rapportera sina känsloupplevelser. Genom att människor lär sig detta tror Empaticus att många av de problem som stress medför kan förebyggas. Applikationen kommer härmed att refereras till som *Emotivation*.

Det har tidigare gjorts andra försök till att bidra med liknande lösningar på problemet. Svagheten hos många av dessa är att de endast täcker en del av behovet men inte alltid kan fungera som en helhetslösning. Det finns sedan tidigare en påbörjad applikation med en del av den funktionalitet som önskas, och detta projekt har för avsikt att vidareutveckla denna applikation med de funktioner som beskrivs i följande avsnitt.

I den existerande applikationen finns det möjlighet för användarna att svara på frågeformulär som berör hur de upplevt sina känslor den senaste tiden. Dessa frågeformulär hämtas lokalt på telefonen och kan därför inte uppdateras utan att användaren tvingas installera en ny version av applikationen. Frågor besvaras under en tiodagarsperiod som kallas *batch*. Varje sådan batch inleds med en speciell typ av frågor som sedan återkommer den sista dagen i samband med att batchen avslutas. Däremellan erbjudes en annan uppsättning frågor vid två tillfällen varje dag. Det finns också en ljudspelare som kan spela upp ljudfiler som även dessa är lagrade lokalt. Utöver det finns en meditationstimer som kan användas till att meditera själv utan hjälp av ljudfiler. Användarens statistik i form av svarsalternativ och meditationstider sparas i en lokal databas. Denna statistik är till för att användas av Empaticus partners i

forskningssyfte men även för att användare skall ha möjlighet att följa sina framsteg. Statistiken är kopplad till ett specifikt användarkonto som i den befintliga applikationen är begränsad till den telefon som applikationen är installerad på. Det är en begränsning som gör att det inte är möjligt att nyttja sitt konto på en annan telefon än den där kontot skapades.

1.2 Syfte

Syftet med arbetet är att vidareutveckla en befintlig applikation med nya funktioner för företaget Empaticus som kan bidra med att hjälpa människor att lära sig att känna igen sina känslor och förstå varför de uppstår.

1.3 Mål

Efter genomfört examensarbete ska:

- Applikationen använda sig av en central databas för autentisering, lagring av frågor och sparande av data.
- Applikationen ha möjlighet att användas offline och då låta användare få tillgång till begränsad funktionalitet av applikationen.
- Applikationen vara försedd med en funktion i form av en känslokarta där det ska vara möjligt att fritt markera ut var i kroppen förnimmelser har upplevts. Detta ska visuellt se ut som en värmekarta där markeringen skiftar färg beroende på hur länge användaren fokuserat ett visst område.
- En desktopapplikation avsedd för administratören finnas med stöd för:
    - Administrering av frågor och svar i den centrala databasen.
    - Hämtning av användares meditations- och frågestatistik.

1.4 Avgränsningar

Utvecklingen av själva mobilapplikationen begränsas till att endast inkludera Android och låter utveckling till andra operativsystem vara möjligt i framtida projekt.

2. Metod

Inledningsvis skall det för varje delmål inhämtas generell information och kunskap för att komma fram till tekniker som är lämpliga för att uppnå målet. Med hjälp av denna kunskap ska det uppskattas hur lång tid det bör ta för varje delmål att implementeras. Innan implementationen av varje del kommer en fördjupningsfas att inledas, där ytterligare kunskap om de aktuella teknologierna inhämtas. Därefter fattas beslutet om vilken teknik som skall användas vid implementation. Efter att detta beslut tagits påbörjas implementationen av det aktuella målet. Det första delmålet som behöver implementeras är den centrala databasen, detta eftersom att den ligger till grund för att de efterkommande delarna skall kunna implementeras. De resterande delmålen är inte beroende av vilken ordning de implementeras i och kan således utföras i godtycklig ordning.

Inhämtning av den teoretiska kunskap som behövs kommer att ske med hjälp av Chalmers bibliotek samt Internet.

3. Teknisk Bakgrund

3.1 SQL

SQL är ett programspråk som är till för att skapa och redigera relationsdatabaser. De fördelar som finns med att använda en relationsdatabas är att duplicering av data minimeras genom att entiteter inte behöver lagras flera gånger, utan tabellkolumner kopplas ihop med hjälp av relationer. Det blir även enklare att modifiera data i tabeller då en ändring i en tabell ändrar samma värde i alla tabeller som den är relaterad till [2].

3.1.1 MySQL

MySQL är en relationsdatabas som är öppen källkod. En stor fördel med MySQL är att det är en väletablerad databashanterare och den har således en stor användarbas. Det innebär att det finns väldokumenterad information att tillgå.

3.1.2 Oracle SQL

Oracle SQL är en relationsdatabas som utvecklas av företaget Oracle. En nackdel med Oracle SQL är att den inte är öppen källkod.

3.1.3 SQLite

SQLite är en serverlös relationsdatabas som läser och skriver direkt till en fil på disk. Alla Android- och iPhone-enheter använder sig av en SQLite-databas. Koden för SQLite är fri att användas i både privata och kommersiella projekt. [3]

3.2 NoSQL

NoSQL betyder Not Only SQL. En fördel som finns med att använda en NoSQL-databas är att alla scheman inte behöver skapas på förhand utan de kan skapas dynamiskt. Det finns flera olika typer av NoSQL-databaser som var och en är bra på lagring av olika typer av entiteter. Prestanda kan optimeras genom att välja rätt typ av NoSQL-databas medan i fallet med SQL finns det endast en typ av databas som då ger färre möjligheter till specialisering. NoSQL-databaser använder sig av objektorienterade API:n för kommunikation med databasen vilket möjliggör direkt kommunikation med

databasen [4]. Bland de olika NoSQL-databaser som existerar är MongoDB och Cassandra de mest populära [5].

3.2.1 MongoDB

MongoDB är en dokument-databas med hög prestanda och automatisk skalbarhet. Dokument motsvarar existerande datatyper i de flesta programspråk[6]. MongoDB använder BSON som är likt JSON för att lagra data [7].

3.2.2 Cassandra

Cassandra är inriktat på att skala riktigt bra i stora projekt och lämpar sig därför bättre för system med många servrar och extremt mycket data att lagra [8].

3.3 JSON

JSON är ett dataformat som är lättläst för människor men även enkelt att förstå för datorer och lämpar sig bra att använda för att skicka data mellan databas och applikation. JSON är anpassat för att vara oberoende av programspråk, vilket passar bra till detta projekt då det bidrar till att underlätta framtida projekt, projekt som inte nödvändigtvis använder samma språk som detta [9].

3.4 Webservrar

3.4.1 Apache HTTP Server

Apache stöder Windows, Unix, Linux och Mac OSX och är öppen källkod [10]. Apaches källkod är väldokumenterad vilket ger värdefulla och tillförlitliga supportalternativ [11].

3.4.2 Microsoft IIS

Microsoft IIS är endast möjligt att använda på Microsofts operativsystem. Denna begränsning är också en styrka i och med att den är optimerad för ett visst operativsystem. En nackdel med IIS är att den är stängd källkod [11].

3.4.3 nginx

nginx optimerar prestanda genom att inte behöva skapa en ny process för varje förfrågan. nginx använder lite minne och kan användas på ett stort antal operativsystem [11]. nginx har inte lika väldokumenterad kod som dess konkurrenter [12].

3.5 PHP (PHP Hypertext Preprocessor)

PHP är ett skriptspråk som främst är avsett för att köras på en webserver och kan användas med alla vanliga, men även många mindre vanliga operativsystem och programmeringsspråk. PHP kan användas för att skapa Internetsidor med dynamiskt innehåll och låta applikationer kommunicera med databaser. PHP reducerar kopplingen mellan logiken i applikation och server genom att lägga till ett extra lager för kommunikation med server [13].

3.6 JDBC

JDBC är ett API som är specifikt för programmeringsspråket Java och definierar hur en klient kan få tillgång till en databas. API:t innehåller färdiga Javametoder för att att skicka SQL-anrop direkt från Javakod till databaser. För att använda JDBC krävs även en JDBC-drivrutin för den databasen som är tänkt att anslutas till [14].

3.7 Jsoup

Jsoup är ett Javabibiliotek som används för att bearbeta HTML. Biblioteket ger ett mycket bekvämt API för att extrahera och manipulera data utifrån HTML-dokumentet [15].
Jsoup har stöd för att:

- Parsa HTML från en URL, fil, eller sträng.
- Hitta och extrahera data, med hjälp av DOM-traversering eller CSS-väljare.
- Manipulera HTML-element, attribut och text.

3.8 ID3-tagg

ID3-taggar är metadatabehållare för mp3-filer som kan innehålla information om filen såsom titel, artist, album, spårnummer med mera. En ID3-tagg möjliggör att denna information kan lagras inuti själva mp3-filen [16].

3.9 Canvas

Canvas är en klass i Androids SDK. Klassen innehåller metoder för att rita objekt på skärmen [17].

3.10 Grafiska gränssnitt i Java

3.10.1 Swing

Swing implementerar en uppsättning komponenter för att bygga grafiska användargränssnitt och lägga till rik grafikfunktionalitet och interaktivitet till Java-program [18].

Fördelar med Swing:

• Välbeprövad teknik för att bygga grafiska användargränssnitt till Javaapplikationer.

• Det finns mycket information på nätet om Swing.

• Inbyggt i programspråket Java.

• Plattformsoberoende.

3.10.2 JavaFX

JavaFX är en uppsättning grafik- och mediapaket som gör det möjligt att designa, skapa, testa, felsöka och driftsätta klientprogram som arbetar konsekvent över olika plattformar. JavaFX är planerad att ersätta Swing som UI-bibliotek för Java [19].

Fördelar med JavaFX:

• JavaFX kan använda CSS för att separera utseende och stil från implementering.

• Plattformsoberoende.

Nackdelar med JavaFX:

• Bristfälligt med information och dokumentation.

3.10.3 SWT

SWT är ett grafiskt användargränssnittbibliotek för Java som är designat för att tillhandahålla effektiv och portabel åtkomst till användargränssnittskomponenter på de operativsystem där SWT finns implementerat [20].

Fördelar med SWT:

• Använder sig av plattformsspecifika element där det är möjligt.

• Finns mycket information om SWT online.

• Plattformsoberoende.

Nackdelar med SWT:

• Kräver plattformsspecifika bibliotek för varje operativsystem som stöds.

3.11 Begrepp

3.11.1 Parsing

Uttrycket parsing innebär att analysera en sträng med symboler för att använda dem på ett sätt som definieras innan man parsar [21].

3.11.2 Mockup

En mockup är en skiss eller modell över hur ett slutgiltigt system skall se ut [22].

3.11.3 Mutex (Mutual Exclusion)

Mutual exclusion-algoritmer används inom programmering för att undvika att två trådar samtidigt exekverar en kritisk region. En kritisk region är en bit kod i vilken en process eller tråd hanterar en gemensam resurs [23].

4. Genomförande

4.1 Planering

Projektet påbörjades med att det genomfördes en efterforskning av de verktyg och teknologier som skulle komma att behöva användas för att slutföra projektet. Efter genomförandet av denna efterforskning fanns det tillräckligt med information för att uppskatta hur lång tid varje del av projektet borde ta. Det skapades då ett Gantt-schema där det planerades hur arbetsuppgifterna skulle delas upp. Detta schema finns i appendix 1. Det avsattes tid för ytterligare efterforskning innan implementeringen av varje enskild del för att fördjupa kunskapen innan arbetet med just den delen påbörjades.

4.2 Databas, Server och Onlineläge

Det inledande steget i projektet var att skapa den centrala databasen som ligger till grund för hela applikationen. Orsaken till att projektet påbörjades i denna ände var att de efterföljande delarna av projektet är beroende av att denna del finns implementerad. Valet stod mellan att använda en databas av typen SQL eller NoSQL. Efterforskning utfördes beträffande de två alternativen för att välja den teknik som var mest gynnsam. En svaghet som observerades med NoSQL-databasen MongoDB var att problem med duplicering kan uppstå, då en entitet ofta kan behöva lagras i flera instanser vilket är ett återkommande problem i den typen av databaser [24]. I övrigt poängterades det inte några tillräckligt stora fördelar eller nackdelar för att det ena alternativet skulle väga tyngre än det andra. Valet föll därför på att använda det alternativ där mest information och bakgrundsfakta fanns, det vill säga SQL.

De SQL-mjukvaror som var aktuella var MySQL och Oracle. Oracle har mer alternativ och möjligheter vad gäller partitionering och replikation och är i allmänhet mer anpassat för större företag och projekt. MySQL innehåller de vanligaste och mest använda funktionerna för ett genomsnittligt projekt. Oracle erbjuder endast en strikt begränsad version utan att man betalar för mer avancerade utgåvor [25]. Egenskaperna hos detta projekt gjorde att valet föll på MySQL.

För skapandet av en effektiv relationsdatabas underlättar det om relationer och tabellstrukturer planeras i ett ER-diagram [26]. Till följd av detta reserverades tid till att skapa ett bra och effektivt ER-diagram.

I samband med att ER-diagrammet färdigställts konverterades det till SQL-kod för skapande av tabeller och relationer.

Nästa steg i utvecklingen bestod av att skapa en server med databas och fastställa ett tillvägagångssätt för att kommunicera med den. Valet för kommunikationsteknologi med server stod mellan PHP och JDBC. Fördelen av att använda PHP är att de skript som skrivs i programspråket kan återvändas i desktopapplikationen eller vid eventuell framtida utveckling av applikationer till andra plattformar. Fördelen med att använda JDBC är att det finns färdiga metoder för kommunikation med databasen direkt ifrån Javakod, vilket innebär att behovet av ett extra programspråk för kommunikation kan undvikas. Valet mellan de två teknologierna föll i slutändan på PHP-skript som anropas från applikationen.

Efter kommunikationsteknologi valts återstod att besluta vilken servermjukvara som bäst passar specifikationerna. I samband med den förberedande fördjupningen inför uppgiften hittades en servermjukvarusvit kallad XAMPP. XAMPP innehåller både en MySQL-databas och en Apache Web Server vilket är precis vad som behövdes för uppgiften. Mjukvaran dokumenteras även som enkel att komma igång med vilket är en fördel i denna sortens projekt [27].

För att kommunicera med databasen skapades ett REST-API som använder sig av PHP-skript. Ett API skapades på grund av att det hjälper oss att hålla låg koppling mellan server och applikation, vilket möjliggör återanvändning av skripten vid framtida utbyggnad. Det skapades skript för att lägga till, ta bort, modifiera och hämta alla databasentiteter. Vid användandet av skripten skickas parametrar med i ett GET- eller POST-anrop till skriptet som finns tillgänglig på webservern. Skriptet utför sedan olika databasoperationer beroende på vilka parametervärden som skickades med. Resultaten packas därefter in i en array och returneras till klienten som JSON. För att utföra anropen från applikationen användes en klass som tar emot tre parametrar. En parameter är URL:en till det skript som tänkts användas. Den andra parametern talar om huruvida POST eller GET skall användas. Den tredje parametern är en lista innehållandes namn/värdepar som ska läggas till URL:en. Som svar på ett anrop fås ett JSON-objekt som innehåller information om anropet var lyckat, hur många resultat som hittades

och de faktiska resultaten. Användandet av JSON och JSON-objekt gör det enkelt att hämta data från databasen och konvertera till Java-klasser för användning inuti applikationen [28].

4.3 Offlineläge

För att implementera offlineläget skapades en planering som innehöll information om vad som var planerat att fungera offline och hur det isåfall skulle fungera. Från början var tanken att applikationen skulle använda Androids inbyggda stöd för SQLite för att på så sätt spegla data på servern mot telefonens databas. En nackdel med denna metod var att för att uppnå optimal prestanda krävs det vetskap om exakt vilken data som behövs speglas. Det kan bli väldigt mycket onödiga synkroniseringar om all data som finns på den centrala servern ska speglas. Efter fördjupning inom alternativa lösningar till spegling hittades ett antal andra möjliga lösningar.

Av de lösningar som fanns ansågs ett system där alla serveranrop köas när Internet inte är tillgängligt som det mest lämpliga. Genom denna metod märker inte användaren av att den är offline vid uppladdningar, då applikationen beter sig på samma sätt mot användaren som om det hade funnits Internetåtkomst. En nackdel med metoden är att de funktioner som behöver hämta data från servern inte går att lösa med samma metod, eftersom det då uppstår långa väntetider för att få Internetåtkomst innan datan blir hämtad. Det beslutades istället att när data hämtas online sparas denna data lokalt på telefonen i själva applikationens minne och när användaren senare försöker komma åt data när den befinner sig offline hämtas datan därifrån istället. Begränsningar som införts i offlineläget är att användaren inte kan starta en ny batch utan Internetuppkoppling, men kan däremot fortsätta och slutföra en redan påbörjad batch.

För att köa dataupppladdning mot servern valdes att använda biblioteket som heter Android Priority Job Queue [29]. Detta bibliotek gör det möjligt att köa olika uppgifter i bakgrunden på en applikation vilket gör att de inte låser UI-tråden och andra vitala delar av applikationen. Det mest intressanta med biblioteket är att uppgifter som kräver Internetåtkomst kan läggas i en kö vilket innebär att de utförs när Internetåtkomst åter blir tillgängligt. Det går även att spara uppgifterna som lagts i kön för att de inte ska försvinna när applikationen termineras och rensas från minnet.

För att köa uppgifter behövs en JobManager. Via en statisk metod kan alla andra klasser i projektet nå en och samma instans av JobManagern för att se till att alla uppgifter läggs i samma kö. När JobManagers instansieras finns möjlighet att i koden ange en del konfigurationsparametrar såsom minimalt och maximalt antal konsumenter och hur många jobb varje konsument kan åta sig samt hur länge konsumenterna skall hållas vid liv då det inte finnas några lediga jobb kvar att tilldela. Konsumenter är i detta fall detsamma som trådar.

Efter att ett fungerande kösystem blivit implementerat i applikationen behövdes något att placera i den. I Android Priority Job Queue används klasser som ärver egenskaper från en klass som heter Job. När ett Job-objekt instansieras kan ett antal konfigurationsparametrar definieras. Dessa parametrar är prioritet, nätverkskrav, persistering, gruppering (används för mutex på delade resurser) och jobbfördröjning. De relevanta parametrarna för jobben i detta projekt är nätverkskrav och persistering av jobben. Sedan tidigare hade ett REST-API i PHP för alla databasanrop skapats, nu användes dessa och det implementerades det att API:t anropas från Jobs och placeras i kön.

4.4 Ljudspelare

I den befintliga applikationen fanns det en ljudspelare vilken innehöll stöd för att spela upp ljudfiler som låg lagrade lokalt på telefonen. I och med att onlineläget implementerades var det ett naturligt steg att lagra ljudfilerna på servern och istället strömma dem därifrån. Fördelarna med att strömma ljudfilerna från servern är att applikationen inte tar upp lika mycket utrymme på enheten i fråga och att applikationens innehåll kan anpassas dynamiskt, utan att användaren behöver göra en ominstallation av applikationen.

Eftersom applikationen skall innehålla ett offlineläge fanns det krav på att selektivt spara ner ljudfiler för uppspelning offline för de tillfällen då användaren inte har tillgång till Internetanslutning eller har en begränsad mängd mobildata att tillgå. Implementationen påbörjades genom att en helt ny ljudspelare utvecklades då det ansågs mer fördelaktigt än att återanvända och modifiera den ursprungliga. Den nya ljudspelaren har funktioner för strömning av ljudfiler från webservern samtidigt som användaren kan välja vilka av dessa filer som skall laddas ned för att vara tillgängliga offline. Strömmningen går till på det

sättet att ljudspelaren hämtar en lista innehållande URL:erna till alla ”.mp3” filer i mappen “audio” på webservern. För hämtning av denna lista används ett bibliotek kallat jsoup [15]. jsoup gör det möjligt att parsa HTML-filer och dess taggar. Webservern som används i projektet generar automatiskt indexfiler (.HTML) för varje katalog i dess filsystem, vilka innehåller information om vad som finns i den aktuella katalogen. Genom att i HTML-filen parsa de <a>-taggar med href-attribut som slutar på ”.mp3” från audio-katalogen på webservern kan en lista över alla ljudfiler genereras.

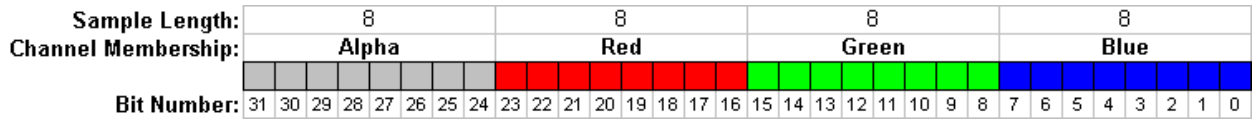
För att visa vilka ljudfiler som finns tillgängliga för uppspelning skapades en listvy (ListView) i Androidaktiviteten som använder sig av en SongAdapter (ärvd från ArrayAdapter) för att representera data och en skräddarsydd listrad (song_row.xml). Syftet med SongAdapter är att hantera en lista och hur datan i vyn representeras på skärmen. En instans av SongAdapter behöver en lista med information om alla ljudfiler (artist, album, spårnamn, längd) och listradslayouten (song_row.xml) för att kunna göra detta. För att få tag på information om en ”.mp3” är det vanligt att tagga ljudfilen med ID3-taggar och sedan läsa dessa. Problemet är att vid strömning av ljudfilerna istället för att läsa dem lokalt kan inte ID3-taggarna utläsas. Därför skapades en speciallösning för att få tag på denna information. För att ta reda på spårnamn, album, artist och spellängd på ljudfilerna skapades ett system där denna information skrivs inom []-taggar, (ex. [artist]-[album]-[spårnamn]-[xxlxx].mp3) i filnamnet. Spellängden anges inom den sista []-taggen. För att skriva “:” skrivs istället “l” på grund av att det inte går att använda “:” i filnamn. Denna lösning gjorde det möjligt att använda listan med URL:er för alla ljudfiler för att ta fram artist, album, spårnamn och spellängd på filerna. Informationen sparas sedan i en klass Song, som lagrar följande information: url, artist, album, spårnamn, spellängd. Alla Song objekt som utvinns ur listan placeras sedan i en ny lista som SongAdaptern kan använda. Listradslayouten (song_row.xml) specificerar hur varje rad i en lista skall se ut. När SongAdapter-objektet är initierat med all nödvändig data sätts det som listvyns adapter. En ljudfil börjar strömmas genom ett tryck på en listrad i vyn på sin enhet, och då skickas URL:en till Androids API:s mediaspelare som sätter igång uppspelning.

För att spara ljudfiler lokalt på enheter skapades en ny Androidaktivititet som hämtar och visar listan med ljudfiler på servern. Skillnaden i detta fall är att ljudfilerna inte spelas upp vid tryck på ett objekt i listvyn, istället sparas de ned i Applikationens privata lagringsutrymme. Om applikationen sedan befinner

sig i offlineläge och meditationsspelaren startas kommer endast en lista med de ljudfiler som sparats lokalt att visas och kunna spelas upp.

4.5 Känslokarta

Nästa steg enligt planeringen var implementation av känslokartan, för att skapa denna är tanken enligt förundersökningen att skapa en skräddarsydd vy som använder en canvas och översätter touch-interaktioner till värmekartsliknande visualisering av touch-datan. Efter ytterligare efterforskning hittades en färdig skräddarsydd vy namngiven HeatView [30]. Vyns underliggande logik fungerar genom att en bitmap i ARGB-format skapas. I figur 4.1 illustreras hur ARGB fungerar. Varje pixel tar upp 4 bytes och varje kanal (A = alfa = genomskinlighet, R = röd, G = grön, B = blå) har 8-bitars precision det vill säga 256 olika värden per kanal. HeatView ärver från klassen View vilken innehåller funktionen onTouchEvent, denna metod exekveras varje gång en användare vidrör vyn. I onTouchEvent hämtas information om antalet fingrar som vidrört vyn och deras x- och y-koordinater sparas. En metod (addPoint) anropas därefter som tar emot koordinaterna och målar en radial gradient runt varje position med en viss radie i Alfa kanalen med värdet 10 additivt (10/256). Detta innebär att om det målas på positioner som redan har blivit tilldelade värden adderas det nya värdet till det gamla, istället för att det gamla värdet skrivs över. När detta steg är slutfört anropas metoden colorize för varje punkt, som hämtar alla pixlar runtom den punkten gradienten målades in på. Metoden går sedan igenom alla pixlar och bitskiftar dem 24 bitar åt höger för att komma åt alfa-kanalen (som kan ha 0-255 som värde). Detta värde översätts sedan till färger i ett färgspektra liknandes en värmekartas spektra. Vilket sker genom att bestämda numeriska intervall motsvarar olika färger. Exempelvis kan alfa-värden mellan 0-80 ge nyanser av blått, alfa värden mellan 81-160 ge nyanser av grönt och alfa värden mellan 161-255 ger nyanser av rött. Bitmappens pixlar ersätts sedan med alla de nya färglagda varianterna.



Figur 4.1 Karta över ARGB [31]

Efter att klassen HeatView blivit inlagd i projektet och dess funktionalitet verifierad, behövdes ett sätt att spara det användaren markerar. På grund av detta skapades en metod för att returnera bitmappen. Det fanns även krav på att spara denna data på servern och länka till användaren. För att åstadkomma detta i en SQL-databas stod valet mellan att ladda upp bitmappen som text, image, eller blob. Valet föll på textformat. För att kunna ladda upp bitmappen som text komprimerades den till en bild i PNG-format som därefter sparades i en bytearray. Bytearrayen översattes till en String i Base64-format som därefter enkelt kunde lagras i databasen som vanlig text (Base64-format innebär att binär data översätts till en ASCII-sträng). Om behovet att åter få bilden i PNG- eller bitmap-format skulle uppstå kan en omvänd process användas.

4.6 Desktopapplikation

För att skapa desktopapplikationen behövdes först ett beslut tas angående vilken teknik som passade för programmeringen av applikationens grafiska gränssnitt. Valet stod mellan SWT, Swing och JavaFX. Valet föll på Swing på grund av att det var där det fanns mest erfarenhet sedan tidigare. Funktionaliteten desktopapplikationen skall innehålla är möjlighet att visa, ändra, ta bort och lägga till nya frågor och svar i databasen samt hämta användarstatistik för användning i forskningssyfte. För att bestämma hur statistiken skall sparas var diskussioner med uppdragsgivaren nödvändig. Alternativen var att antingen hämta all användarstatistik från databasen och visa inuti applikationen med hjälp av någon grafisk representation eller att spara ner informationen i tabeller i något format kompatibelt med Excel. Efter samtal med uppdragsgivaren beslutades Excel-lösningen som lämpligast, då personerna intresserade av denna statistik har vana av Excel.

För att implementera applikationen påbörjades en designfas där en mockup över hur det grafiska användargränssnittet skulle se ut skapades. Ur mockupen identifierades vilka Java Swing klasser och layouter som mockupelementen motsvarade. Efter detta skapades en version av applikationen med alla gui-element på plats men utan någon bakomliggande logik för databasoperationer. För att testa att gui-elementen fungerade korrekt fylldes de med påhittad data för att verifiera funktionaliteten. Logik implementerades sedan för hur alla operationer på datan skulle fungera, dock lokalt utan koppling mot

databas i detta steg. I nästa steg återstod att ersätta den lokala logiken mot funktioner kopplade mot databasen.

För att koppla applikationen mot databasen stod valet mellan att återanvända PHP-skripten eller att använda JDBC. Valet föll på JDBC på grund av att det ger färre lager mellan klient och server (Java<->Databas) medan PHP ger fler lager (Java<->PHP<->Databas). För att använda JDBC mot en server krävs det att en särskild drivrutin för JDBC importeras till projektet. Det verifierades att det gick att upprätta en anslutning mot databasen med drivrutinen. Till sist skapades ett nytt API med hjälp av JDBC för de operationer som tänkt vara tillgängliga i desktopapplikationen. Efter att API:t färdigställts ersattes den lokala logikens metoder med API:ts metoder. För nersparandet av statistiken användes ett bibiliotek som heter poi för att skapa HSSF-dokument (Excel dokument) av SQL-tabellernas värden [32]. Genom att bearbeta resultaten av ett SQL-anrop som returnerar all statistik kan poi infoga tabellvärdena rad för rad i en klass HSSFSheet. Detta visas i form av ett kodavsnitt i figur 4.2, där resultatet av SQL-anropet stegas igenom och infogas i Exceldokumentet. För att skapa en HSSFSheet behövs en annan klass vid namn HSSFWorkbook, som används efter att alla värden har blivit infogade för att spara ett HSSFSheet som en .xls fil.

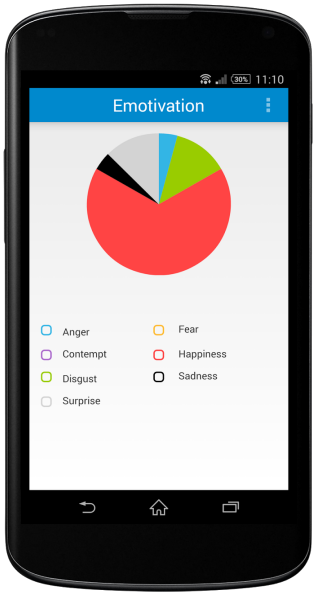
```
//Create headers for the column names
HSSFRow rowhead=   sheet.createRow(0);
for(int i=0 ; i < columnNames.length ; i++) {
    rowhead.createCell(i).setCellValue(columnNames[i]);
}

//Fill the rows with data from the resultset
int i=1;
while(rs.next()){
    HSSFRow row=   sheet.createRow(i);
    for(int j=0 ; j < columnNames.length ; j++) {
        row.createCell(j).setCellValue(rs.getString(columnNames[j]));
    }
    i++;
}
```

Figur 4.2 Kodavsnitt över infogning av tabellvärden till ett HSSF dokument.

5. Resultat

5.1 Databas, Server och Onlineläge

Den existerande applikationen som endast fungerade offline har byggts ut med ett onlineläge. Onlineläget möjlliggör dynamisk hämtning av innehåll till applikationen från en server, registrering av användare och sparande av dess statistik från besvarade frågeformulär i serverns databas. Ett exempel på hur en användares statistik kan se ut visas i figur 5.1. Det innehåll som nu går att redigera utan att uppdatera själva applikationen är olika typer av frågor samt meditationsspår. Dessa uppdateras av administratören och är inget som den enskilda användaren kontrollerar. Data som tidigare gick förlorad vid ominstallation av applikationen finns nu kvar även efter att användaren tagit bort applikationen i och med att allt lagras på servern. Onlineläget gör det möjligt för uppdragsgivaren att ta del av statistik över hur applikationen används för att använda denna i forskningssyfte. Genom analys av datan kan exempelvis statistik utvinnas om hur användarbasens känslomässiga tillstånd förändras över tid genom användning av applikationen. Detta kan ge viktig information om vad som kan vara effektiva behandlingsmetoder. På så sätt kan applikationen bidra till att förebygga många av de problem som uppstår när fler människor blir konstant stressade i vardagslivet.



Figur 5.1 Exempel på hur frekvent en användare känt var och en av de vanligaste känslorna.

Servern är skapad med den plattformsoberoende mjukvaran XAMPP. Det är en mjukvarussvit som innehåller stöd för bland annat MySQL-databaser och en Apache-webserver. MySQL-databasen används av applikationen för att spara användardata och statistik i samt hämta frågor ifrån. Apache-webservern har som syfte att göra de PHP-skript som används för att kommunicera med databasen nåbara och även för att streama/ladda ned meditationsspår till ljudspelaren.

5.2 Offlineläge

För att underlätta för användare som har begränsad tillgång till Internet har ett offlineläge implementerats som gör att applikationen beter sig på liknande sätt oavsett om Internetanslutning är upprättad eller inte. För att börja använda offlineläget krävs det att användaren loggar in och autentiserar sitt konto mot servern, något som inte är möjligt att göra offline. Offlineläget fungerar på det viset att om användaren försöker använda en funktion som skickar data till servern kommer serverförfrågringar att läggas till i en kö istället för att försökas skickas direkt till servern. Det innebär att det inte märks någon distinkt skillnad för användaren då hon är offline jämfört med online, då gränssnittet fungerar på samma sätt i båda fallen. I och med att Internetanslutning åter etableras kommer kön att börja bearbetas automatiskt och skicka data till servern. Den data som hämtas från servern sparas också internt i telefonens minne och används då en användare försöker hämta data offline. Det innebär att den data som visas i applikationen, till exempel statistik, inte garanterat är den allra senaste då applikationen befinner sig offline.

5.3 Ljudspelare

Under arbetets gång beslutades att ljudspelarens funktionalitet skulle utökas, något som inte var med i planeringen från början men ändå kändes som ett naturligt steg i applikationens övergång till att fungera online. Resultatet blev att det nu finns en ljudspelare, med möjlighet att spela upp förinspelade meditationsspår, som dels kan strömma filer lagrade på server dels låter användaren spara ner de filer hon vill ha möjlighet att lyssna till offline till telefonens minne. Ljudspelaren respektive statistik över en användares meditation kan ses i figur 5.2 nedan. Studier visar att meditation medför att hjärnans

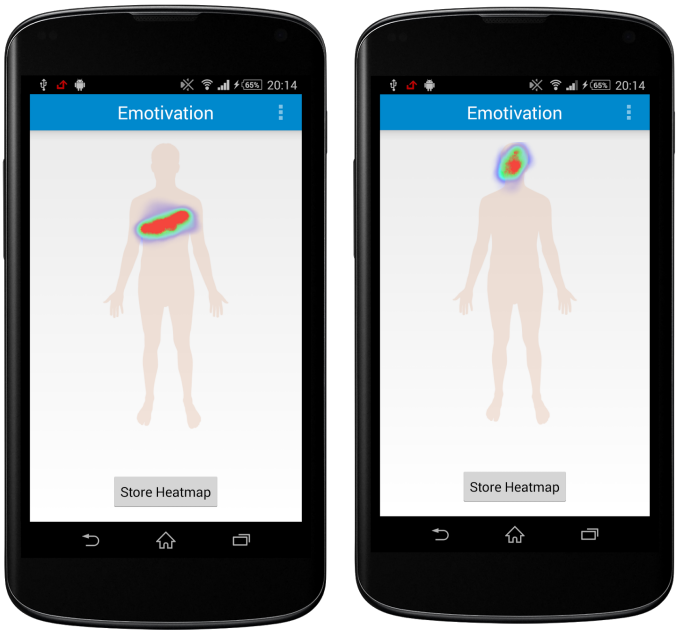
EEG-aktivitet minskar under meditation, vilket ökar avslappning och ger ett allmänt lugnare tillstånd [33]. Detta bidrar enklare bibehållning av den lugna känslan även efter meditation. Det är precis det som Empaticus tror kan bidra till den förbättring av hälsan som på sikt är bra för samhället.

Figur 5.2 Lista med tillgängliga meditationsspår (t.v). Statistik för en användares meditationstid (t.h).

5.4 Känslokarta

Applikationen har givits en ny funktion där användaren på en känslokarta i form av en kropp kan markera var förnimmelser uppstått och hur intensiva de har varit. Desto längre ett finger hålls på en viss position desto starkare intensitet kommer färgen på markeringen att få. Resultatet liknar en värmekarta. Detta ska bidra med lärdom om att känna igen hur en viss känsla uppenbarar sig i form av förnimmelser i olika delar av kroppen. Med hjälp av det ska det gå att förebygga oönskade känslor genom att ett visst mönster i sin kropp uppenbaras när känslan uppkommer. Det är också möjligt att känna igen i vilka situationer som en viss känsla uppstått. I figur 5.3 visas två exempel på hur användaren kan

markera hur denne känt. Det finns mycket utbyggnad som kan göras kring denna funktion. Det kan till exempel vara av värde att spara ner vilken geografisk position som en viss känsla upplevts, till exempel kanske en viss känsla oftare uppstår på jobbet än i hemmet. På det sättet skulle det gå att härleda varför en viss känsla uppkommer vid just det speciella tillfället och inte annars.

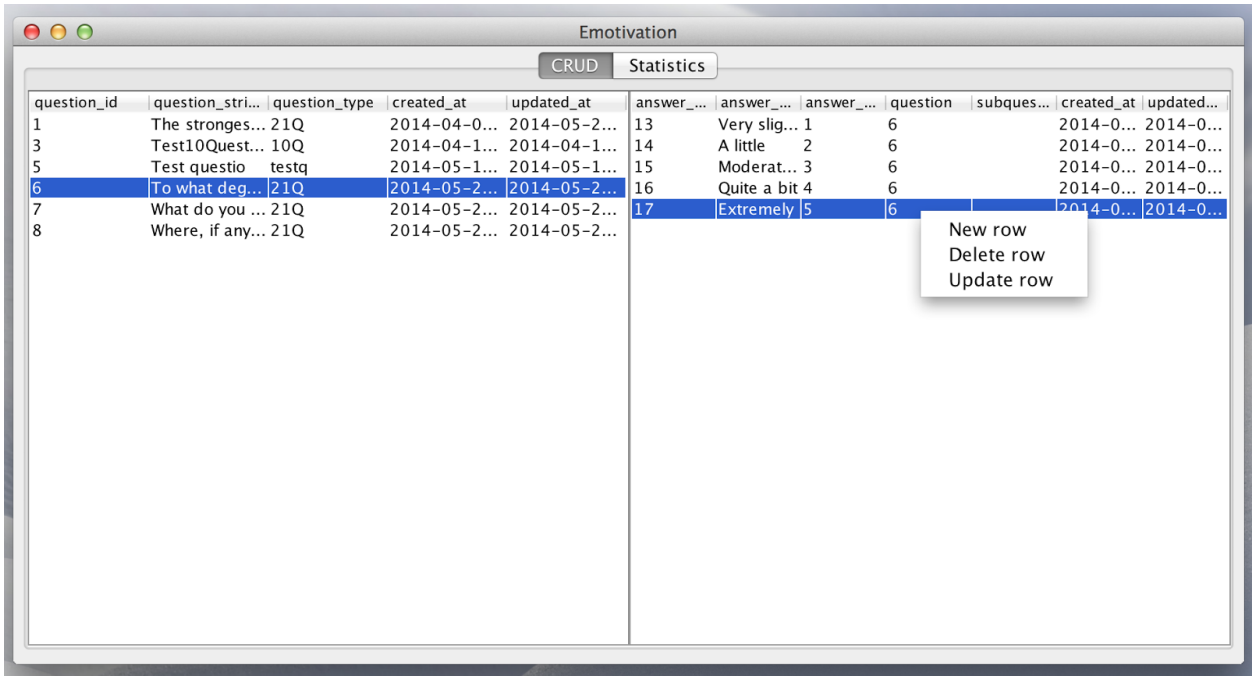


Figur 5.3 Ett exempel på två olika markeringar på värmekartan

5.5 Desktopapplikation

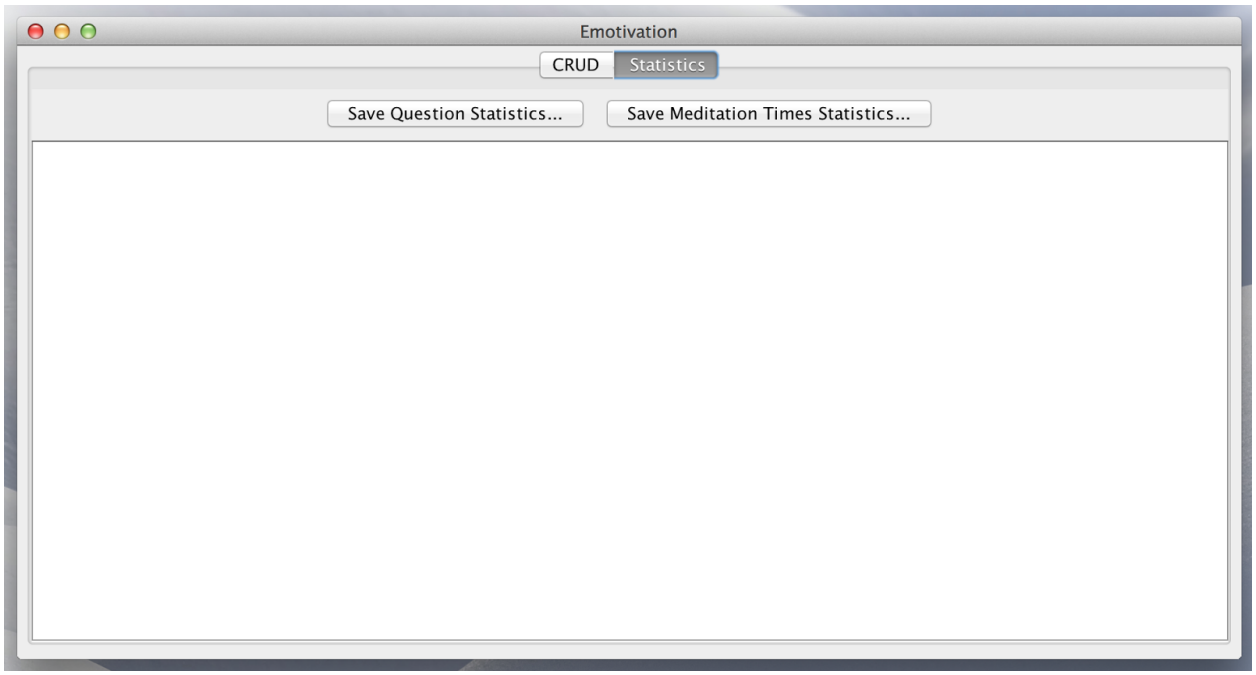
En desktopapplikation har skapats med följande funktioner:

- Stöd för att lägga till/ändra/ta bort frågor och svar till frågorna. Detta syns i figur 5.4.
- Stöd för att hämta användarstatistik. Detta syns i figur 5.5.



Figur 5.4 Översikt över hantering av innehållet i databasen från desktopapplikationen.

Desktopapplikationen är skapad i Java med hjälp av biblioteket Swing. För att kommunicera med servern använder applikationen sig av JDBC för att ansluta och skicka förfrågningar.



Figur 5.5 Knappar för att exportera statistik från databasen till Excel via desktopapplikationen.

6. Slutsats

6.1 Resumé

Projektets mål var att vidareutveckla en Androidapplikation med nya funktioner. Dessa är ett online- och offlineläge, en känslokartsfunktion samt en desktopapplikation för uppdatering av frågor i databasen och hämtning av statistik. För att implementera onlineläget skapades en server med en relationsdatabas. Kommunikation med servern sker genom att aplikationen skickar parametrar till PHP-skript, som ligger på servern, som skickar tillbaka önskad information formaterat i JSON. Offlineläget skapades genom att all datauppladdning köas när nätverk eller server ej finns tillgängligt, och all den senaste data som hämtas från servern sparas lokalt för att användas då nätverket inte är tillgängligt. Känslokartan implementerades genom användandet av en skräddarsydd vy som gör om touchhändelser till färger med olika intensitet beroende på hur länge en viss yta vidrörs. Desktopapplikationen har skapats med hjälp av Java Swing och innehåller funktioner för att uppdatera frågor och svarsalternativ i databasen samt exportering av statistik.

En funktion som implementerats, som inte var med i kravspecifikationen, är en ljudspelare som kan strömma meditationsspår som ligger lagrade på servern och också har möjlighet att låta användaren välja vilka spår den vill ha tillgång till vid användning offline.

6.2 Kritisk Diskussion

Onlinedelen av Androidapplikationen skapades med PHP och MySQL. Efter att vi fått erfarenhet av att använda JDBC i desktopapplikationen insåg vi att det var ett smidigare sätt att kommunicera med databasen än PHP skript. Därför hade det antagligen både varit fördelaktigare och tillfört färre lager att använda JDBC även i Androidapplikationen. En nackdel med användande av JDBC i Androidapplikationen hade dock varit att det endast går att använda i Java, vilket medför att utveckling av en applikation till iOS eller Windows Phone hade varit tvungen att inkludera ytterligare programmering för samma funktionalitet som kan återanvändas genom att anropa PHP-skript. Att använda PHP-skript i Androidapplikationen ger även fördelen att om ett skript på servern ändras

förändrar det funktionaliteten i alla applikationer som använder sig av skriptet och tillåter således förändring av funktionalitet i applikation eller databas utan att programmera om och kompilera nya versioner av applikationerna.

Vi valde att använda SQL på grund av att den är väletablerad och har stor användarbas samtidigt som vi har en del tidigare erfarenhet av SQL. Men i efterhand kanske det hade vart bättre att använt MongoDB då vi inte utnyttjat de finesser som en relationsdatabas medför särskilt effektivt. I projekt där datan i databasen är av en mer simpel karaktär och inte nödvändigtvis behöver innehålla många advanecrade relationer kan det vara mer fördelaktigt att använda en databas av typen dokument istället.

Den ursprungliga tanken var att skapa offlineläget genom att skapa en lokal SQLite-databas som vi kommunicerar med lokalt när applikationen är offline och vill göra förändringar och sedan synkronisera mot den centrala MySQL-databasen när Internetanslutning åter etablerats. Vi insåg dock att denna lösning skulle bli alltför svår att implementera. Därför fick vi byta metod till ett system där vi köar förfrågningar för att uppnå önskad funktionalitet.

Offlinedelen av applikationen skapades med hjälp av Android Priority Jobqueue som var mycket användbart och passade ändamålet perfekt. Men det finns en del nackdelar med att använda detta bibliotek. Att göra UI-uppdateringar beroende på om ett jobb genomförts framgångsrikt eller misslyckat blir svårare i och med att jobbet inte nödvändigtvis utförs direkt, därför har interaktionen med UI:t blivit något mindre responsiv. Innan vi införde offlineläget hade vi en metod som skickade användaren vidare till en lämplig aktivitet efter en uppgift blivit utförd med framgång eller visade ett meddelande vid misslyckande, men den biten finns inte kvar nu eftersom vi inte vet direkt om operationen är lyckad eller ej. Därför bör en del arbete läggas på att förändra arkitekturen mellan UI-uppdateringar och serverinteraktion för att få ett UI som svarar bättre på särskilda händelser.

Desktopapplikationen är skriven i Java och använder JDBC för att ansluta till servern. Till skillnad från mobilapplikationen kan desktopapplikationen köras av alla operativsystem som har stöd för en JVM. Detta gör applikationen plattformsoberoende. Tanken var från början att applikationen skulle

återanvända de PHP-skript vi skapade för Androidapplikationen. Vi valde dock att använda JDBC för att minska antalet lager mellan klient och server.

Med tanke på att projektet inte har inneburit att skapa något helt och hållet från grunden utan snarare modifiera och utöka ett existerande system bedömer vi det angreppssätt vi har valt som passande. Eftersom vi hade ett existerande projekt att utgå ifrån krävs det undersökning av den befintliga koden först för att kunna göra vissa strategiska val för att se vad som passar. Medan om projektet hade byggts från grunden hade det varit möjligt att göra en mycket mer omfattande planering att följa redan vid den initiala planeringfasen.

6.3 Samhällsaspekter

Stress är nödvändigt för att kunna prestera bättre i pressade situationer. Dock kan den långvariga typ av stress som kommer genom psykisk påfrestning från omgivningen ge upphov till flera problem, till exempel fysiska spänningar i kroppen, vilka efter tillräckligt lång tid i sin tur kan leda till depression och långvariga sjukdomar. Vi tror att genom regelbundet användande av *Emotivation* skulle många problem med stress och utbrändhet på arbetsplatser och skolor minska. Genom att utföra dagliga känslorapporter går det att kartlägga och bli mer medveten om sin känslomässiga situation och upptäcka mönster i vardagen som kan hjälpa att förstå sitt inre. Det finns möjlighet att utöva meditation med hjälp av applikationen, både genom att lyssna till inspelade guidade meditationer samt att på egen hand meditera med en timer som håller reda på tiden och kan ge signaler med jämna intervall. Resultatet kommer då bli att vi kan råda bot på de problem som vi nämnt tidigare som bland annat innefattar att sjukskrivningar minskar vilket ökar tillgänglig arbetskraft för företag och kan på det viset bidra till ökade inkomster.

6.4 Vidareutveckling

Trots att *Emotivation* nu berikats med flera användbara funktioner finns det fortfarande plats för nya, som kan förverkligas i framtida projekt. I detta kapitel kommer några av de mest signifikanta idéer som kommit fram under utvecklingen att nämnas.

Utveckling till fler plattformar är något som skulle vara positivt för att kunna nå ut till en större användarskara. Sådan utveckling har underlättats genom att databasen och servern inte är skapade för att passa endast till Android, utan kan med fördel återanvändas i ett sådant framtida projekt genom att förfrågningar till databasen besvaras i ett format som är anpassat att vara möjligt att läsas av vilket programmeringsspråk som helst.

För att föra användare av applikationen närmare varandra och uppmuntra till fortsatt nyttjande av *Emotivation* kan någon form av socialt nätverk implementeras som är begränsat till användare av just den här applikationen. Där skulle användare ha möjlighet att dela framsteg och upplevelser med vänner, klasskamrater, kollegor eller till allmänheten.

Wearables är föremål som bärs på kroppen och kopplas till en mobiltelefon, till exempel armband, klockor och pulsmätare. Marknaden för dessa föremål börjar bli större och större. Insamlad data från sådana apparater skulle kunna integreras med *Emotivation* och utifrån denna information ge respons till användaren med koppling till dennes emotionella hälsa.

Applikationen kan "spelifieras", alltså införa ett system där uppdrag kan utföras som vid fullbordande ger poäng och låser upp nya uppdrag. Uppdragen skulle till exempel kunna gå ut på att utföra goda gärningar i sin omgivning och genom att rapportera dessa som uträttade skulle användaren göra sig förtjänt av poäng eller någon annan form av belöning. Ett sätt att representera sig själv i applikationen skulle kunna vara genom att skapa en avatar som har olika sociala förmågor som kan tränas genom att använda *Emotivation*. Genom framgångsrik träning av sin avatar i en viss färdighet kan det antas att användaren själv blivit bättre på den egenskapen i det verkliga livet också. På det viset går det att följa den utveckling som sker steg för steg.

Referenser

[1] M. Danielsson et al., “Folkhälsorapport 2009: Psykologiska påfrestningar och stressrelaterade problem,” Socialstyrelsen, Stockholm, Sverige, 2009. ISBN: 978-91-978065-8-9

[2] H. Garcia-Molina, et al., “The Relational Model of Data”, *Database Systems,* second edition. Stanford, USA: Pearson 2009, ss 17-64. ISBN: 978-0-13-135428-9

[3] “About SQLite,” SQLite, [Online] Tillgänglig: http://www.sqlite.org/about.html. [Hämtad: 16 jun, 2014].

[4] “NoSQL Databases Explained,” MongoDB, 2014 [Online] Tillgänglig: http://www.mongodb.com/nosql-explained. [Hämtad: 4 jun., 2014].

[5] “DB-Engines Ranking,” DB-Engines, 2014 [Online] Tillgänglig: http://db-engines.com/en/ranking. [Hämtad: 4 jun., 2014].

[6] “Introduction to MongoDB,” MongoDB, 2014 [Online] Tillgänglig: http://docs.mongodb.org/manual/core/introduction/. [Hämtad: 16 jun 2014].

[7] “BSON - Binary JSON,” BSONSpec, mar 2014[Online] Tillgänglig: http://bsonspec.org/. [Hämtad: 4 jun., 2014].

[8] “The Apache Cassandra Project,” Cassandra, maj 2014 [Online] Tillgänglig: http://cassandra.apache.org/. [Hämtad: 4 jun., 2014].

[9] “Introducing JSON,” JSON, apr 2014 [Online] Tillgänglig: http://www.json.org/. [Hämtad: 4 jun., 2014].

[10] S. Hall, “IIS vs Apache,” ScriptRock, apr 2014 [Online] Tillgänglig: http://www.scriptrock.com/articles/iis-apache. [Hämtad: 16 jun 2014].

[11] E. Simard, “Nginx: The Best HTTP Server,” GloboTech, nov 2013 [Online] Tillgänglig: http://www.gtcomm.net/blog/nginx-the-best-http-server/. [Hämtad: 4 jun., 2014].

[12] L.Isaac, “Nginx Vs Apache: Nginx Basic Architecture and Scalability,” The Geek Stuff, nov 2013 [Online] Tillgänglig: http://www.thegeekstuff.com/2013/11/nginx-vs-apache/. [Hämtad: 16 jun 2014].

[13] “What can PHP do?,” PHP, jun 2014 [Online] Tillgänglig: http://php.net/manual/en/intro-whatcando.php. [Hämtad: 4 jun., 2014].

[14] “JDBC Overview,” Oracle, maj 2014 [Online] Tillgänglig: http://www.oracle.com/technetwork/java/overview-141217.html. [Hämtad: 4 jun., 2014].

[15] J. Hedley, “jsoup, Java HTML Parser,” jsoup, 2013 [Online] Tillgänglig: http://jsoup.org/. [Hämtad: 4 jun 2014].

[16] “ID3.org,” id3.org, 2013 [Online], Tillgänglig: http://id3.org/. [Hämtad: 16 jun 2014].

[17] “Canvas,” Android, 2014 [Online], Tillgänglig: http://developer.android.com/reference/android/graphics/Canvas.html. [Hämtad: 16 jun 2014].

[18] “Java SE 7 Swing APIs and Developer Guides,” Oracle, 2014 [Online] Tillgänglig: http://docs.oracle.com/javase/7/docs/technotes/guides/swing/. [Hämtad: 16 jun 2014].

[19] “Java FX Overview,” Oracle, 2014 [Online], Tillgänglig: http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm. [Hämtad: 16 jun 2014].

[20] “SWT: The Standard Widget Toolkit” Eclipse, 2014 [Online], Tillgänglig: http://www.eclipse.org/swt/. [Hämtad: 16 jun 2014].

[21] D. Howe, “parse”, Dictionary.com, 2010 [Online], Tillgänglig: http://dictionary.reference.com/browse/parse. [Hämtad: 16 jun 2014].

[22] “mock-up,” Dictionary.com, 2014 [Online], Tillgänglig: http://dictionary.reference.com/browse/mock-up. [Hämtad: 16 juni, 2014].

[23] “What is Mutual Exclusion?,” Techopedia, 2014 [Online], Tillgänglig: http://www.techopedia.com/definition/25629/mutual-exclusion-mutex. [Hämtad: 16 juni 2014].

[24] S. Mei, “Why you should never use MongoDB,” Sahramei, nov 2013 [Online] Tillgänglig: http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/. [Hämtad: 4 jun 2014].

[25] G. Trujillo, “MySQL vs Oracle Features/Functionality,” Oracle, maj 2008 [Online] Tillgänglig: https://blogs.oracle.com/GeorgeTrujillo/entry/mysql_versus_oracle_features_functionality. [Hämtad: 16 jun 2014].

[26] H. Garcia-Molina, et al., “High-Level Database Models”, *Database Systems,* second edition. Stanford, USA: Pearson 2009, ss 125-202. ISBN: 978-0-13-135428-9

[27] “About the XAMPP Project,” Apache Friends, 2014 [Online] Tillgänglig: https://www.apachefriends.org/about.html. [Hämtad: 4 jun 2014].

[28] R. Tamada, “How to connect Android with PHP, MySQL,” Android Hive, maj 2012 [Online] Tillgänglig: http://www.androidhive.info/2012/05/how-to-connect-android-with-php-mysql/. [Hämtad: 4 jun 2014].

[29] “Path - Android Priority Job Queue,” Path - Github, 2013 [Online] Tillgänglig: https://github.com/path/android-priority-jobqueue. [Hämtad: 4 jun 2014].

[30] F. Bob, “Heating up my new Xoom tablet,” Furious Bob , maj 2011 [Online] Tillgänglig: http://blog.furiousbob.com/2011/05/12/heating-up-my-new-xoom-tablet/. [Hämtad: 4 jun 2014].

[31] http://en.wikipedia.org/wiki/RGBA_color_space

[32] A. C. Oliver et al., “Apache POI - the Java API for Microsoft Documents,” The Apache POI Project, 2014 [Online] Tillgänglig: http://poi.apache.org/. [Hämtad: 4 jun 2014].

[33] B. Cahn, J. Polich, “Meditation states and traits: EEG, ERP, and neuroimaging studies,” American Physological Association, mar 2006 [Online] Tillgänglig: http://psycnet.apa.org/journals/bul/132/2/180/. [Hämtad: 4 jun 2014].

Appendix:

Appendix 1: Gantt-schema