Bachelor's Thesis

Stressreducerande applikation för Androidbaserade system

Nikolaj Delvin

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, 2016.

Stressreducerande applikation

för Androidbaserade system

Av

Nikolaj Delvin

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Abstract

This thesis has been performed at MediYoga International. Company's concept is centered on education of instructors and other personnel in stress reducing methodology. The methodology is based on scientific studies of yoga and has well documented effects.

The main reason for long-term sick leave in Sweden today is burn-out, as a direct consequence of continuous stress witch is supported by many scientific studies. The company has an ambition to develop a stress reducing application suited for modern phones and tablets witch could target a broad audience. The choice of platform is based on devices availability to the user, because the frequent use of the application is essential to the effective stress reduction. The constant availability could be vital when stress symptoms make a sudden appearance.

In the given study we analyze the company's methodology and documents to get a better understanding for their concept behind reduction of the negative effects of long term stress on the individual. Similar solutions are investigated and interviews are carried out with focus groups to get a better understanding for the task and to create a requirements specification. Based on the aforementioned analysis a prototype has been designed and developed for Android based systems for further evaluation.

Keywords: stress, yoga, application, Android, Java, XML, Agile

Sammanfattning

Examensarbetet har utförts på MediYoga International. Företagets verksamhet centrerar sig kring utbildning av instruktörer och annan personal inom stressreducerande metodik. Metodiken baseras på vetenskapliga studier av yoga och har väl dokumenterade effekter och framgång.

Den största anledningen för långvarigsjukskrivning i Sverige idag är utbrändhet som är en direkt konsekvens av långvarigstress, vilket även beskrivs i många vetenskapliga studier. Företaget har en ambition att skapa en stressreducerande applikation för moderna telefoner och andra mobila enheter som skall kunna användas av en bred målgrupp. Val av plattform baseras på dess konstanta tillgänglighet för användare då ett frekvent användande av applikationen bör förhöja dess effekt. Tillgängligheten kan även vara av stor vikt vid en plötslig uppkomst av stressrelateradesymptom.

Under examensarbetet analyseras företagets metodik och dokumentation för att närmare förstå deras koncept kring reducering av de negativa effekterna av långvarigstress hos individen. Liknande lösningar undersöks och intervjuer bedrivs med fokusgrupper för att ta fram en kravspecifikation samt skapa en djupare förståelse för uppgiften. Utifrån den ovannämnda analysen designas och implementeras en prototyp för Androidbaserade system för ytterligare utvärdering.

Nyckelord: stress, yoga, applikation, Android, Java, XML, Agile.

Innehåll

1. Inledning

Kapitlet som följer beskriver förutsättningarna i början av projektet som är baserade på en kort intervju med Mediyoga och en ytlig analys av företagets verksamhet. Vissa förändringar tillkommer under projektets förlopp baserat på eliciteringsprocessen som beskrivs i detalj i de kommande kapitlen.

1.1. Bakgrund

Projektet skall göras på uppdrag av Itbox AB för Mediyoga International AB i Stockholm.

Mediyoga har utvecklats genom kliniskt och terapeutiskt arbete med människor sedan början av 1997 Mediyogas metodik är baserad på resultat av hundratals vetenskapliga studier på olika typer av yoga över hela världen, inklusive ett halvdussin svenska studier. Idag har Mediyoga över 1400 utbildade instruktörer i Sverige och Norge, de flesta inom hälso- och sjukvården. Deras verksamhet består huvudsakligen av utbildning av yogainstruktörer och riktar sig mot stress och utbrändhet i alla åldrar. Mediyoga distribuerar även material som består av instruktionshäfte och CD (ljudspår med instruktioner och musik) via sin hemsida (Mediyoga, 2016).

Nu vill man nå slutanvändare via en ”digitalinstruktör” i form av en applikation för Android och iOS. Materialet som distribueras idag är kostsamt att både producera och förmedla samt saknar kopieringsskydd. Distributionssättet och formatet hindrar även företaget att nå en yngre generation. En applikation kommer att kunna användas på moderna, bärbara enheter som mobiltelefoner samt surfplattor och finnas tillgängligt via webbaserade distributionslösningar. Android och iOS enheter har ett flertal givare som kan förbättra upplevelsen ytterligare för Mediyogas kunder. Externa givare i form av exempelvis smartklockor kan användas för pulsmätning, även andningsmätning kan implementeras. Kamera och GPS kan användas under loggning av de utförda passen. Inloggning kan ske via fingeravtrycksläsare som finns i de moderna telefonerna/läsplattorna.

Tillsammans med Mediyoga ska ett nytt lämpligt koncept tas fram som lämpar sig för moderna mobila enheter samt behandlar den ovannämnda problematiken.

1.2. Syfte

Bibehålla effekten av Mediyogas metodik (hälsovinster) i den nya applikationen, samt minimera riskerna av illegal spridning av materialet i samband med inträdet på den breda marknaden. Göra applikationen användbar för alla åldrar inom målgruppen (18-65). Den sekundära ambitionen är att försöka använda den befintliga utrustningen hos enheterna för en förbättrad upplevelse.

1.3. Mål

Analysera Mediyogas koncept som består av situationsstyrda meditationssekvenser (CD och häfte), skapa djupare förståelse för deras domän (yoga, meditation). Ta fram ett nytt koncept tillsammans med Mediyoga som lämpar sig för mobila enheter och återspegla denna i en kravspecifikation för applikationen. Modellera applikationen i UML för vidare implementering av en testbar prototyp för Androidplattformen.

1.4. Problemformulering

- Ta fram en kravspecifikation i samarbete med företaget. Kravspecifikationen ska följa någon standard till exempel från Kravkursen. I specifikationen ska det framgå vilka sensorer som prototypen ska använda och vilken funktionalitet som ska ingå. Önskvärt är minst en sensor, möjlighet att strömma media från databas, möjlighet att skicka data från sensorn till databas.
- Göra ett antal “mock-ups” på papper av applikationen för att hitta en lämplig utformning. Utvärdera dem genom en undersökning på företaget och eventuellt dess kunder. Utvärderingen ska göras med hjälp av intervjuer.
- Välja protokoll/format för strömmande media.
- Välja en databas/server där man kan lägga mediefiler och hålla reda på kunderna.
- Välja utvecklingsverktyg för applikationen.

- Utveckla en Androidapplikation utifrån resultaten i de föregående punkterna. Applikationen ska kunna kommunicera med en server så att någon form av media kan strömmas till applikationen och data från applikationen ska kunna skickas till servern. Observera att både filer och data bara behöver vara korta exempel som visar att konceptet fungerar.
- Testa och utvärdera prototypen. Utvärderingen ska göras i samarbete med företaget och ska göras på ett systematiskt sätt.

1.5. Avgränsningar

Ingen färdig produkt kommer att levereras under projektet utan en kravspecifikation och design samt en testbar prototyp för Android. Endast en del av de valda givarna kommer att implementeras i prototypen. Preliminärt leveransdatum augusti-september 2015.

2. Metod

Detta kapitel kommer att beskriva metoder som har används under examensarbetet. Vi kan dela in processen i följande delar: planering, rapportskrivning, studier, analys, design, implementation, test och validering. Metodiken under utvecklingsfasen följer en traditionell iterativ modell.

2.1. Planering

Under planeringsstadiet definierades alla arbetsmoment och strukturerades kronologiskt med hjälp av Gantschema. Tidsintervallerna för de olika momenten visade sig vara ganska svåra att bestämma när man saknar både kunskap och erfarenhet. Däremot definitionen av de olika arbetsmomenten gav upphov till en struktur som har gett stöd under hela processen samt definierat en viss parallellism i arbetet.

2.2. Rapportskrivning

Rapporten har upprättats parallellt med resterande arbetet i form av anteckningar och dokument relaterade till sina arbetsmoment. En tidig observation som gjordes under rapportskrivningen var dess bidrag till en djupare förståelse av problemet.

2.3. Analys

Processen utformades med stöd av (Lauesen, 2002) och har följande struktur: analys av kundföretaget som inkluderar dokumentstudie och analys av nuvarande arbetssätt, analys av liknande lösningar och avslutningsvis en gruppintervju. Analysen av problemet beskrivs ingående under ”Elicitering”.

2.4. Studier

Inför studier definierades kompetenssvagheter inom ramar för Androidplattformen i form av XML och funktion av ett grafiskt gränssnitt. Inledningsvis studerades XML med hjälp av litteratur (Harwani, 2013) och

internetresurser vilket visade sig vara ganska ineffektivt men gav däremot en bättre bild av nuvarande trender inom GUI design. Istället för att studera abstrakta exempel och övningar behövdes verkliga mål som var relevanta för projektet. Omstruktureringen resulterade i arbete parallellt med studier vilket snabbt gav resultat.

2.5. Design

Endast i en perfekt linjär värld är studier, design och implementation tre separata faser. Projektet dikterade sin verklighet vilket innebar en konstant återkoppling mellan faserna. Designen gjordes först med papper och penna i form av klassdiagram för att sedan realiseras i kod. Även andra diagram som sekvens och kontextdiagram användes för att ge ytterligare stöd under modelleringen av applikationen. Klassdiagrammen skapades med hjälp av simpleUML, främst för att få en bättre överblick över koden samt kunna presentera designen i rapporten. Ambitionen har varit att använda sig av lämpliga mönster och principer för att göra koden både gedigen och underhållningsbar. Abstraktionen är inte fullständig vilket är i linje med prototyputveckling.

2.6. Implementering

All implementering gjordes i Android Studio och bestod av Java och XML kod. Viss uppenbar refaktorering gjordes löpande till en grad som ansågs lämplig eller snarare tillräcklig under ramarna för en prototyp.

2.7. Test och validering

Inget utförligt test av applikation har gjorts under projektet då målet är en prototyp. I samband med varje avslutad iteration testades applikationen på målmaskin av kund samtidigt som funktionaliteten validerades. Feedback från kund presenterades oftast muntligt och bristerna antecknades. För underlättad kommunikation med kund upprättades en Cloudlösning från Dropbox (Dropbox, 2016) en molntjänst för informationsdelning. I test och valideringssyfte införskaffades en Android enhet av kund, för att underlätta

installation och justering av Android studio användes fjärrstyrning med hjälp av Team Viewer (Teamviewer, 2016).

2.8. Källkritik

Android Programming är skriven av B.M. Harwani och är en pedagogist bok som behandlar stegvis inlärning av Android programmering vilket i sin tur har gett bra stöd under både design och implementeringsstadiet. Harwani är en känd pedagog inom programmering med ett flertal titlar hos stora förlag.

Software Requirements, Styles and Techniques är skriven av Lauesen, S och används i kursen Kravhantering (ETS 672) på Lunds tekniska högskola. Boken har gett ett starkt stöd under de initiala eliciteringsfaserna och varit ett underlag för alla beslut avseende val av strategi.

Den officiella sidan för Android utvecklare är en resurs som riktar sig mest till utvecklare och innehåller detaljerade beskrivningar av systemet samt kodexempel. Resursen är upprättad av Google och håller en väldigt hög kvalité både vad gäller layout och innehåll. Informationen här har bidragit främst för en närmare förståelse av samspelet mellan hård och mjukvara i Android system.

Kompendium från Medi Yoga International AB av E. Engqvist belyser företagets koncept och synsätt på stressreducering och stress. Analysen av dessa anses vara av stor betydelse för förståelsen av företagets arbetsmetoder samt skapar en grund för intervjuerna som finns beskrivan i kap. 4.5.

3. Teknisk bakgrund

Kapitel som följer beskriver de tekniska förutsättningarna under projektet som involverar både mjuk och hårdvara. Här förklaras även vissa principer som har används under implementering.

3.1. Hårdvara

All implementering har skett på en stationär Intelbaserad PC med operativsystemet Windows 10 Pro, någon djupare beskrivning av maskinen saknar relevans. Som målmaskin under projektet för test och felsökning samt kvalitetssäkring av exekvering användes en Androidbaserad enhet, Sony Xperia Z3 Compact med Androidversion 6.0.1, Marshmallow. Enheten som är lanserad 2014 kan ses som medelmåttlig på dagens telefonmarknad och har följande projektrelevanta specifikation (Sony, 2016):

CPU

- Snapdragon 2,5 GHz fyrkärnig Qualcomm
- Adreno 330 GPU

Minne och lagring

- 2 GB RAM
- 16 GB flashminne

Ljud och ljudformat

- Sonys teknik för 3D-surroundljud (VPT)
- Clear Audio+ – ljudförbättringsprogramvara
- Upplev xLoud™
- DSEE HX
- Högupplöst ljud
- Ljudinspelning, filformat som stöds: 3GPP, MP4, AMR

- Ljuduppspelning, filformat som stöds: 3GPP, MP4, ADTS, AMR, FLAC, Matroska, SMF, XMF, Mobile XMF, OTA, RTTTL, RTX, iMelody, MP3, WAV, OGG, ASF

Skärm

- 4,6-tums display (1280 x 720 bildpunkter)
- TRILUMINOS™-display
- X-Reality™-bildmotor

Anslutning

- aGPS
- Trådlös Bluetooth® 4.0-teknik
- 3,5 mm hörlursuttag med teknik för digital brusreducering (DNC)
- DLNA-certifierad®
- NFC
- Inbyggd USB-sammanlänkning
- Synkronisering via Exchange ActiveSync®, Facebook™, Google™ och SyncML™
- Stöd för höghastighets-USB 2.0 och Micro USB
- Wi-Fi®- och Wi-Fi-hotspot-funktionalitet
- Trådlös ANT+-teknik

Sensorer

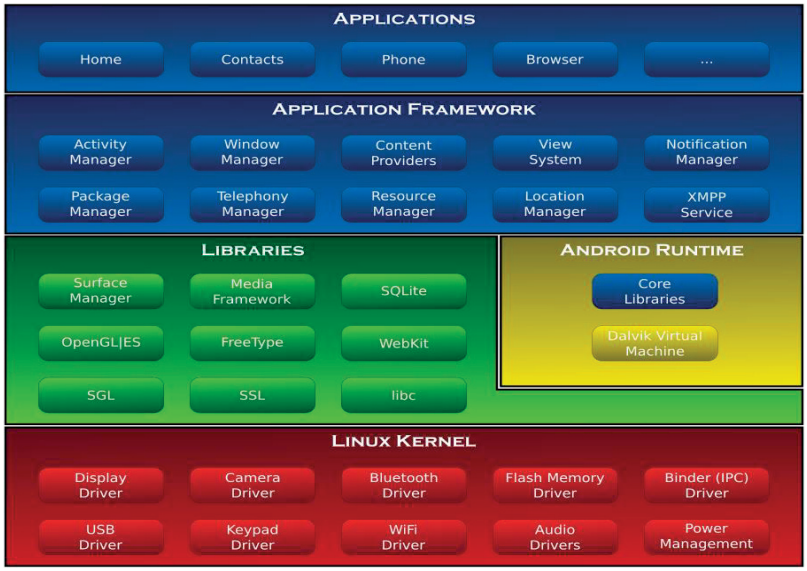
- Accelerometer
- Gyroskop
- Avståndssensor
- Kompass
- Barometer

3.2. Mjukvara

Här kommer vi att titta på mjukvaran som har används under projektet med fokus på implementering och test, närmare bestämt mjukvaran på arbetsdatorn och målmaskinen.

3.2.1. Android

Ett operativsystem som förnärvarande utvecklas av Google och är baserat på Linux. Systemet är primärt skapat för mobilaenheter med en touchskärm som telefoner och surfplattor men har idag blivit utökat för stöd av bland annat fordon (Android auto), TV (Android TV) och externa enheter som klockor (Android Wear). Sedan 2013 är Android utan tvekan en marknadsledande plattform med 81.3% av smartphonemarknaden i tredje kvartalet samma år.



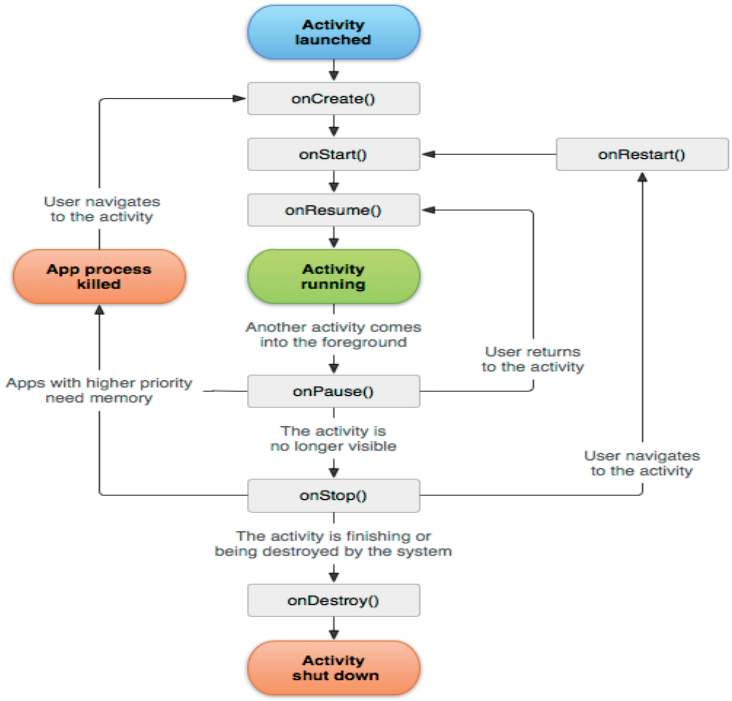
Figur 3.1 Android arkitektur (The official site for Android developers, 2016)

Som vi nämnt tidigare så är Android primärt riktat till batteridrivna enheter vilket resulterar i jakten på att minimera elkonsumtion hos enheterna. Detta åstadkommer man via minimering av CPU användandet av pausade applikationer, för att kringgå detta finns speciella mekanismer. Minneshanteringen har också sina överraskningar, när enheten får slut på minne raderar OS pausade applikationer från primärminnet enligt FIFO principen vilket kan vara frustrerande för en tredjepartutvecklare.

Android Open Source Project är öppen källkod som kan modifieras och ändras av entusiaster efter deras behov med en tät återkoppling till Google. Ofta baseras nya uppdateringar på just analys av deras arbete. För ytterligare fördjupning besök (The official site for Android developers, 2016).

3.2.2. Java för Android

Den eventuellt största skillnaden ur programmeringssynpunkt från Java är Androids egna bibliotek som bland annat möjliggör kommunikationen mellan OS och applikationen samt integration av XML. Basen i en Android applikation utgörs av Activity klasser som måste deklareras i AndroidManifest.xml och saknar någon main metod. Activity klassen implementerar ett interface som stödjer applikationens livscykel som är dikterad av OS se figur 3.2 nedan.



Figur 3.2 Activity, livscykel (The official site for Android developers, 2016)

Som framgår utav figuren 3.2 så kan applikationen bli avbruten av både användare och OS i stort sett när som helst vilket medför en viss komplexitet. En annan skillnad är OS versionshantering avseende målmaskin så kallad API nivå som uttrycks i build.gradle filen, valet präglar de tillgängliga biblioteken som är knutna till SDK. Här handlar det om att göra en avvägning tidigt i processen mellan modernitet och tillgänglighet dvs desto senare API nivå desto färre enheter som kan använda sig av applikationen. Värt att tillägga att API nivåer är bakåtkompatibla.

För övrigt är det samma syntax och skapar inga större hinder för en Java programmerare.

3.2.3. Android Studio

All utveckling under projektet har gjorts i Android Studio 2.1.1 med JRE 1.7 och JVM Hotspot 64-bit. Utvecklingen av prototypen har i huvudsak involverat två kodhanteringsverktyg inom den givna plattformen, nämligen Java och XML editor. XML delen i Android ansvarar för layout av användargränssnittet och tillför en tydlig separation mellan just layout och kod samt ger en mer överskådlig bild. Värt att nämna att layouten i Android kan implanteras helt i Javakod utan XML och tvärtom. För att uppnå den önskevärda separationen kan vi redan nu tilldela ansvarsområde som följande: XML för layout och kod för funktion av layout. Editorn har även ett grafiskt hjälpmedel som endast har används vid granskning av layoutelement under projektet.

För att uppnå en överskådlighet av designen användes simpleUML som är en plugin till Android Studio. SimpleUML levererade överskådliga klassdiagram med utsatta beroende som både fungerade som stöd under implementering och presentation i denna rapport. Verktyget kan både generera diagram utifrån färdig kod och tvärtom samt är enkelt och gratis.

4. Elicitering

Eliciteringsprocessen kommer att centrera sig kring intervjuer med intressenter som består av både användarmålgruppen och experter på företaget. För att effektivisera detta inleds processen med domänkännedom, nämligen: dokumentstudier, analys av liknande lösningar och nuvarande arbetssätt (Lauesen, 2002). Extraherad funktionalitet skall senare användas under intervjuer för utvärdering. Syftet med eliciteringsprocessen är att skapa förståelse för Mediyogas metodik samt extrahera funktionalitet för applikationen.

Definierade nyckelpunkter för elicitering:

- Nuvarande arbetssätt
- Målsättning med projektet
- Funktionalitet
- Test av koncept

Baserat på de ovannämnda punkterna samt begränsningar dikterade av MediYoga gjordes följande val av eliciteringsmetoder:

- Analys av intressenter
- Gruppintervju
- Dokumentstudier
- Analys av liknande lösningar
- Analys av nuvarande arbetssätt
- Brainstorming
- Utveckling av prototyp

4.1. Mediyoga

Metodiken centrerar sig kring stress och utbrändhet men behandlar även fysiska hälsoproblem. Den baseras på holistiska discipliner i form av olika yogainriktningar som med hjälp av hundratals vetenskapliga studier gjorda i samarbete med bl.a. Karolinska Institutet, Danderyds sjukhus och Stockholms Universitet omvandlats till en vetenskaplig teknik. Metodiken består av fysiska övningar som kombineras med andning och koncentrationsövningar som anpassas individuellt efter varje patientens unika behov. Tillstånd som behandlas omfattar: astma, ryggproblem, utbrändhet, sömnstörningar, ätstörningar, migrän och även Parkinsons syndrom m.fl.

4.2. Dokumentstudie

Det primära syftet med studien är att skapa förståelse för Mediyogas metodik samt förstå orsakerna bakom stress och utbrändhet, följande dokument analyserades: ”Instruktörsutbildningskompendium” (Elisabeth Engqvist, 2015) och ”Stresskompendium” (Elisabeth Engqvist, 2015) båda används vid utbildning av instruktörer.

4.2.1. Stress

Stress är en reaktion på fara eller ett behov av prestationsökning som katalyseras av vår omgivning. Vid uppkomst av omedelbar fara försätts den mänskliga kroppen med hjälp av hormonell styrning i ett tillstånd som är avgörande för överlevnaden. De så kallade stresshormoner bidrar till reducerad smärtkänslighet, ökar muskulär prestation och skärpt kognitiv förmåga samtidigt som behovet av mat, sömn och reproduktion minskar. Tillståndet i sitt ursprung är kortvarigt och åtföljt av återhämtning.

Stress i det moderna samhället

Idag är det i högsta grad ovanligt att vi blir attackerade av ett vilt djur eller behöver en plötslig prestationsökning i samband med jakt eller krig men trotts det så ligger stress och utbrändhet högst på sjukskrivningsstatistiken, vad är orsaken? Enligt författaren så är det kraven som dikteras från individens omgivning, krav som ofta är tidsbundna. Som resultat så hamnar individen i ett långvarigt stresstillstånd utan möjlighet till återhämtning, vilket orsakar både psykiska och fysiska skador på organismen. Under stress produceras i huvudsak alltid tre hormoner: adrenalin, noradrenalin och kortisol. Det är deras inverkan på nervsystemet under en längre period som ger upphov till utbrändhet och försätter kroppen i ett långvarigt stresstillstånd.

Stressens skadeverkningar/symptom

Pulsfrekvensen ökar för att kunna förse muskler, lungor och hjärna med blod samtidigt som blodkärlen drar ihop sig och blodets koagulationsförmåga ökar vilket på sikt kan resultera i hjärtklappning, arytmi och högt blodtryck. Som ledning så ökar även andningsfrekvensen, endast de översta delarna av lungorna aktiveras och ger upphov till en försämrad syresättning av blodet per andetag. Den här typen av andning är väldigt påfrestande för muskulaturen kring hals, nacke, axlar och käkar då

diafragmans (muskulatur som dominerar under normal andning) aktivitet minimeras. Konsekvensen utav konstant spänd muskulatur är smärta som oftast är lokaliserad i rygg, axlar och nacke som även indirekt ger upphov till dålig kroppshållning och förkortade muskler. Då blodflödet koncentreras kring muskulaturen i anslutning till rörelseapparaten och andning utsätts andra organ för svält. Matsmältningssystemets aktivitet halveras och funktion förändras vilket leder till magkatarr, sura uppstötningar, kramp, matöverkänslighet, förstoppning och irriterade tarmar, för att nämna några. Därav sjunker även näringsupptagningen och det leder till tömning av kroppens vitamin, mineral och spårämne depåer. Kompensationen för den nedsatta näringsupptagningen och musklernas behov av snabb energi kompenseras med frisättning av fettsyror och socker i blodet vilket leder till ökning av det skadliga kolesterolet och ökar risken för diabetes 2. Även immunförsvaret anpassas till att skydda mot ett fysiskt angrepp och koncentreras kring hud, benmärg och lymfkörtlar. Vid hög aktivitet i immunförsvaret under längre perioder utan återhämtning kan försvaret kollapsa och även börja angripa sig själv. Vid långvarig stress ökar halten av stresshormoner men däremot köns och tillväxthormoner minskar, vilket på sikt bidrar till en reducerad sexlust, ökade PMS-smärtor, nedsatt ämnesomsättning samt påverkar känslan av välbefinnande på ett starkt negativ sätt. Även mun och svalg blir bort prioriterade under tillståndet man kan uppleva torrhetskänsla i munnen och försämrad tandhygien, de omdisponerade kroppsvätskorna kan även utlösa spasmer i hals- och svalgmuskulaturen. För att blodförlusten skall minimeras vid penetrering så minimeras även blodflödet i huden vilket bidrar till uttorkning och i vissa fall även till håravfall uppe på skalpen.

Behandling/stressreducering

MediYoga har utvecklat ett antal verktyg som kan användas för att reducera effekten av stress och den viktigaste av dem är baserad på andning, ett verktyg som även ger en omedelbar effekt. Författaren påstår att genom att styra andningen så kan kroppen försättas i ett vilotillstånd och påbörja återhämtning. ”*Tre andetag kan göra skillnad. Tre riktigt långa, djupa andetag kan få en situation att förändras” -Elisabeth Engqvist.* Vid djup och lugn andning är det omöjligt för kroppen att övergå i ett stresstillstånd, författaren rekommenderar att styra andningen under hela dagen eller så ofta som det är möjligt. MediYoga har utvecklat ett antal övningar som kan tillämpas i alla vardagsmiljöer med 3-5 minuters varaktighet som ger en omedelbar stressreducerande effekt.

Metodiken behandlar hela levnadssätt och inkluderar även kost och tillvaro som har en positiv inverkan på nervsystemet, här följer några exempel:

Försvagar nervsystemet:

- Tobak, även som indirekt rökare
- Alkohol
- Socker, även råsocker. Honung och fruktsocker är att föredra. Vitt socker är effektivt för kroppen som näringsbränsle och kan orsaka eller förstärka ångest.
- Vitt mjöl, kan liksom socker, förstärka ångestkänslor.
- Koffein, kaffe, läsk, energidrycker m.m.
- Choklad, kakao
- Långa eller många mobilsamtal
- Tv-tittande
- Datoranvändning

Stärker nervsystemet:

- Stärkande mat: sellerijuice, färsk ingefära, bananer, oliver, Yogi te
- Dagsljus
- Frisk luft
- Vistas i naturen, särskilt skogen
- Tänka positiva, upplyftande tankar
- Yoga
- Meditation
- Iskall dusch på morgonen. Bidrar till dopaminutsöndring i hjärnan. -Dopamin är en belönings signalsubstans i hjärnan.

Författaren sätter även vikt på tankemodulering ”Tänka positiva, upplyftande tankar”, det innebär att man inte tänker negativa utfall i diverse kommande scenario utan styr tankeverksamheten till nuet. Minimera rädslan genom att låta kognitiv verksamhet dominera över känslorna, analysera och planera istället för en redundant känslomässig reaktion på kommande händelser som endast skapar ångest och saknar konstruktiv substans. Analys av varningssignalerna som manifesterar sig i form av depression och ångest kan skapa förståelse för orsakerna bakom,

viktigt att inte nonchalera dessa då de endast växer sig starkare. En annan stresskatalysator i det moderna samhället är gapet mellan där man är och där man vill vara, det är inget fel i ambitionen att utvecklas och sätta upp livsmål utan problemet ligger i den bristande hanteringen av vägen dit, menar författaren. Närvaroövningar behandlar just den typen av stress och bygger på meditation. Idag finns det väldigt mycket vetenskapligt belägg för meditationens positiva effekter på vår hjärna som består av förbättrad fokus, uppmärksamhet och koncentration. Meditation påverkar även hypotalamus som kontrollerar aptit, sömncykler och det emotionella tillståndet.

4.2.2. Riktlinjer och funktionalitet för applikationen

Syftet med applikationen centrerar kring stressreducering, utifrån stressorsaker kan vi forma följande riktlinjer: minimera notifikationer, minimera uppmaningar till användaren (registrering, inköp, dela innehåll i sociala nätverk, m.fl.), avstå från att placera användaren inom några som helst tidsramar, retoriken i applikationen ska vara mjuk och inbjudande. Försöka på ett diskret sätt att ge rekommendationer för en sundare livsstil för en ökad effekt. Låta användaren att sätta upp konkreta mål i form av reducerade stressymptom och möjlighet att följa upp dessa.

Tekniska tillbehör

Först ska vi titta på symtomen som kan läsas av med modern teknik samt formulera krav på den typen av utrustning. Marknaden erbjuder idag ett väldigt brett sortiment av utrustning som främst riktar sig mot idrottare, här följer en lista på utrustningstyper som kan mäta relevanta symptom i form av andnings och pulsfrekvens, samt blodets syresättning. Utvärderingen är gjord i sammarbete med Mediyoga, fakta om enheterna är hämtad från (Vandrico Inc, 2015):

- Smarta klockor med pulsmätare och armband
- Väst med integrerade givare för både puls och andningsfrekvens mätning
- Hörlurar som mäter både pulsfrekvens och syresättning

Efter en analys av möjligheterna, formulerades följande krav på utrustning:

- Utrustningen måste vara lätt att ta med sig överallt dvs. lätt

tillgänglig
- Låg kostnad
- Kunna läsa så många symptom som möjligt för en mer korrekt bild

Vi börjar med att titta på klockorna som erbjuder avläsning av ett symptom av tre, vilket kan vara vilseledande vid analys och därav svårt att implementera på ett relevant sätt.

Västen som är visserligen tunn och åtsittande kräver underhåll i form av tvätt och är framtagen för att bäras under träningspass dvs. endast under kortare perioder (1-3 timmar) vilket kan leda till att den blir obekväm för användaren. Symptombilden blir ganska bra då den uppfyller två av tre möjliga avläsningar. Västen är ganska dyr (fr. 2000kr) och svårtillgänglig på marknaden.

Hörlurarna kan mäta två av tre stressymptom, är relativt billiga (fr. 1500kr) samt enkla att ha med sig oberoende av sysselsättning samtidigt som de faktiskt är hörlurar med handsfree dvs. utrustning som många redan använder sig av idag. Sistnämnda typen anses som mest lämplig och därav ska prioriteras under implementering.

4.3. Analys av nuvarande arbetssätt

Förnärvarande bedriver Mediyoga försäljning av sitt material bestående av ljud CD samt instruktionshäfte via sin hemsida. CD innehåller ljudspår bestående av muntliga instruktioner med backgrundsmusik. Instruktionshäftet innehåller instruktioner till ljudspåren som består av bilder och text som behandlar varje övningsmoment.

4.3.1. Brister i det nuvarande arbetssättet

Bristerna definierades vid ett telefonmöte med Mediyoga och är följande:

- Föråldrad media (CD), kunder har varken vilja eller möjlighet att använda CD-spelare

- Otympligt distributionssätt, hantering av försändelser tar mycket arbetstid
- Logistiska omständigheter som sändningsfördröjning, lagring och kostnad
- Materialet kan inte finnas konstant tillgängligt för användaren, svårt att ta med sig överallt
- Saknar kopieringsskydd
- Produktionskostnad
- Dyrt

4.4. Liknande lösningar

Listan på applikationer baseras på förslag från Mediyoga och är en produkt av en analys av mer än 60 relevanta applikationer. För att uppnå en objektiv bedömning gjordes först analysen av applikationen och sedan undersöktes försäljningsstatistiken, värt att tillägga att analysen aldrig avvek från statistiken.

4.4.1. Bedömningskriterier

Bedömningarna baseras på statistik och observation i de flesta fall sammanföll det ganska väl som framkommer av noteringarna se Appendix A. Förutom statistiska kriterier, infördes flera subjektiva som kvalité och engagemang. Engagemang syftar på viljan hos användaren att använda applikationen och ska inte förväxlas med behovet. Kvalité är betydligt mindre subjektivt och är en samling av faktorer: layout, design, implementering (buggar, fördröjningar), detaljrikedom. Beskrivningen ger en kort översikt och skapar en bra referens. Allt under rubriken ”Negativt” är ganska självförklarande. Rubriken ”Positivt” syftar inte bara till utvunnen funktionalitet utan definierar även riktlinjer för den kommande applikationen.

4.4.2. Fördjupad granskning av ledande applikationer

Som framgår utav analysen i Appendix A har vi två ledande applikationer nämligen Headspace och OfficeYoga med 500.000 nedladdningar respektive 1.000.000. Båda applikationerna realiserar sin försäljning via prenumeration, Headspace kostar 12.95$/mån och OfficeYoga 4.99$/mån. Applikationerna är välgjorda och är av hög kvalité, det som enligt min

uppfattning skiljer de åt är att Headspace är utvecklat med ett stort engagemang från experter inom Mindfulness och OfficeYoga är mer en samling av yogaövningar ordnade och presenterade på ett snyggt sätt. Med hög sannolikhet har Headspace använt sig av en iterativmodell där konstant feedback från kunden varit tillgänglig men däremot OfficeYoga är eventuellt en tolkning av en kravspecifikation. Något som stödjer den ovannämnda tesen är den aggressiva marknadsföringen som OfficeYoga utsätter sina användare för, vilket i sin tur strider mot stressreducering. OfficeYoga använder även foto på människor i sina guidningar till övningssekvenser vilket enligt Elisabeth Engqvist strider mot en neutral och avslappnat sinnestillstånd hos användaren och som resultat motverkar reducering av stress. Headspace däremot bygger sin presentation på neutrala teckningar och animationer vilket kan tyda på en expert inblandning. Det är just den röda tråd som en applikation av detta slag måste implementera enligt min uppfattning vilket i sin tur skapar en lust hos användaren.

4.4.3. Utvärdering av applikationsanalysen

Förutom att extrahera funktioner och skapa förståelse för domänen så har undersökningen definierat fallgropar. Applikationen skall vara inbjudande eller engagerande för användaren vilket man uppnår via design, relevant detaljrikedom och användarvänlighet. Däremot ett uppfyllt syfte eller ett tillfredsställt behov skulle garantera ett fortsatt användande. Den röda tråden eller syftet skall framgå i både design och funktionalitet speciellt i det givna fallet som skall bidra till stressreducering hos användare.

4.5. Gruppintervju

Inför intervjun slutfördes dokumentstudier och analysen av liknande produkter för att skapa grundläggande förståelse för domänen och kunna relatera till gruppen. Intervjun dominerades av öppna frågor med en viss minimal ledning. Stängda frågor baserades till största del på analysen av liknande applikationer. Gruppen delades in i två delar för att nybörjare inte skulle känna press från experternas auktoritet. Varaktighet 1,5 - 2 timmar per intervju, totalt två genomförda gruppintervjuer.

4.5.1. Planering

- Presentation av deltagare: Namn, yoganivå och datoranvändning.
- Nuvarande situation: Hjälpmedel, var, när, svårigheter
- Beskriva typiska scenario i sitt utövande
- Diskussion av funktionalitet i liknande applikationer
- Brainstorming kring ny funktionalitet

4.5.2. Utvärdering

Båda grupperna kände sig avslappnade vilket ledde till djupa och långvariga diskussioner och en del brainstorming. Avancerade yogautövare kunde tidigt exkluderas från den preliminärt definierade målgruppen baserat på bristande behov av stöd men däremot nybörjare eller användare utan ambition att avancera stärkte sitt behov. Gruppen saknade däremot medlemmar som inte utövar någon form av yoga vilket resulterade i beslut om att involvera även den målgruppen under prototyputvärdering samt bedriva en ny intervju som skall rikta sig mot skeptiker och icke yogautövare.

För detaljerad intervjubeskrivning se Appendix B.

4.6. Resultat av analysen

Eliciteringen har levererat en del konkreta funktioner och en del övergripande riktlinjer som skall följas under projektet en del av dessa kommer att användas vid prototyp framställning som kulminerar eliciteringsprocessen. Vid telefonmöte med Mediyoga framställdes följande lista som baseras på extraherad funktionalitet och som skall ligga till grund för kravspecifikationen. Listan är preliminär och skall främst användas för referens.

Funktionella:

1. Färdiga övningssekvenser modulerade av experter.
2. Musik under övningar.

3. Möjlighet att stänga av det muntliga stödet. Grundar sig på betinget (Pavlov) till musiken.
4. Behov av att se sitt tillstånd (fysiska, psykiska) förändras över tiden. Subjektiv bedömning i flera kategorier (sömn, stress, mediciner, osv). Grundar sig på en läkares rekommendation att föra dagbok.
5. Varierande övningslängd fr. 3 min.
6. Nivågradering av övningar.
7. Flexibel storlek på applikationen, grundar sig på minnesbrist på enheterna.
8. Schemalagda sekvenser som är målorienterade (bättre sömn på två veckor, bli kvitt ryggsmärtor på en månad, osv.).
9. Möjlighet till korta pauser 1-2 min mellan övningarna, pausen skall initieras av applikationen.
10. Instruktioner till övningar skall bestå av ljud, text och stillbilder.
11. Möjlighet att skicka information till användare via applikationen, exempelvis inbjudningar till live events, lyssna av ljudblogg.
12. Föra någon form av statistik över användaren.
13. Ge användare möjlighet att vara sociala via applikationen.
14. Ingen pausfunktion under övningarna, endast start och stop.
15. Applikationen skall kunna läsa av användarens fysiska tillstånd som färd (GPS), störande ljud (mikrofon).
16. Applikationen skall utifrån användarens tillstånd (se 18) ge rekommendationer (sätta in hörlurar, övningar).
17. Applikationen skall uppmana användaren att försätta enheten i flygplansläge innan påbörjad övning.
18. Användargränssnittet skall ändras beroende av tiden på dygnet, 3 tillstånd: morgon, dag och kväll

Övergripande:

- Behov av att utöva när som helst var som helst
- Inga störningar eller avbrott
- Applikationen skall behandla både psykisk och fysisk hälsa med tyngdpunkt på stressreducering

Grafisk design (GUI):

- Visuellt: Natur, vågrörelse, behagliga färger. Materialdesignkoncept
- Ljud: Naturtema (hav, regn, fågelkvitter, osv).
- Foto på människor skall vara svartvita

Konceptuellt:

- Inga måste (jag måste = stress)
- Bryta det negativa tankemönstret så tidigt som möjligt
- Applikationen skall främst centrera kring stressreducering
- Andningen är av högsta vikt

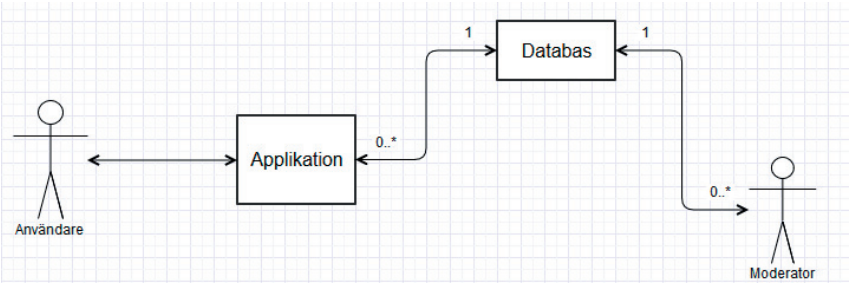
5. Kravspecifikation

Främst skall en målsättning definieras för specifikationen. Specifikationen skall ge en övergripande bild av hela systemet och samtidigt vara bitvis enkel för validering av icke tekniskt kunnig personal på Mediyoga. För underlättad läsning kommer Mediyoga att hädanefter refereras till som kund. Namn av typen ”AssetRetriever” hänvisar till klasser men däremot ”assetRetriever” hänvisar till variabeler eller metoder om inte kontexten antyder något annat.

Följande avgränsningar anses vara lämpliga (Lauesen, 2002):

- Kontextdiagram över gränssnitt se figur 5.1
- Användningsfall och stöd

Kontextdiagramet kommer att ge stöd under designfasen och användningsfallen kan däremot bidra till valideringen och är lättolkade. Här görs även valet av SDK nivå och sätts till 21 vilket motsvarar Android 5.0 Lollypop (senaste 6.0), enligt Android Studio svarar det mot ca 85 % av enheterna på marknaden.



Figur 5.1: Kontextdiagram

Användningsfall och stöd (Tasks and support)

Specifikationerna upprättas enligt en iterativmodell och beskrivs i detta kapitel i turordning. Kravmodellen som används gör det möjligt för kunden som saknar teknisk kompetens att vara delaktig i processen under planeringsstadiet. Modellen saknar teknisk terminologi samt har en

förenklad struktur. Specifikationen omfattar beskrivning av arbetsområdet, användare, målsättning och möjliga lösningar. Beskrivningen av den ovannämnda kravmodellen är tagen ur (Lauesen, 2002).

5.1. Första iterationen – GUI

Arbetsområde

Användargränssnitt

Gränssnittet skall följa de riktlinjer som är satta i det föregående kapitlet. Syftet med iterationen är att ge en grafisk bild på hur gränssnittet kan se ut utan att gå in på funktionaliteten i applikationen samt engagera kunden på ett tidigt stadie.

Användare

Applikationens målgrupp.

Målsättning

Att skapa ett gränssnitt som följer riktlinjer som är satta under elicitering.

Förslag på lösning

Gränssnittet skall domineras av naturbilder och innehålla behaglig och avkopplande retorik. Designen skall följa ”material design” koncept.

5.2. Andra iterationen – Spelare för övningssekvenser

Arbetsområde

Uppspelning av övningssekvenser bestående av musikspår, röstspår, bilder och text. Övningssekvenserna består av övningar vars längd bestäms av röstspåret.

Användare

Applikationens målgrupp.

Målsättning

Skapa en androidaktivitet utgående från beskrivningen av arbetsområdet. Spelaren anses vara en central punkt i applikationen och därav skall implementeras tidigt i processen för att kunna granskas och justeras av kund under de kommande iterationerna så att en viss parallellitet kan uppnås i den iterativa processen. Spelaren skall även skapa en diskussion kring övningssekvensernas utformning på ett tidigt stadie.

Förslag på lösning

Varje röstspår har tillhörande text och bildsekvenser som ändras beroende av tiden men oberoende av varandra. Musiken ska repeteras under hela övningssekvensen. Musik och ljudspåret skall oberoende av varandra kunna sättas till ljudlöst av användare. Sekvenserna skall även kunna pausas och stoppas av användare. Återstående och total (övningssekvens) tid skall visas under uppspelning.

5.3. Tredje iterationen – Presentation av övningssekvenser

Arbetsområde

Presentation av övningssekvenser för användare. En sorterad lista med relevant information om övningssekvenser som är kopplad till uppspelning av dessa. Sekvenserna är indelade i följande mängder: nödvändiga tillbehör (chair, mat, none) som är ömsesidigt uteslutande och lämpliga utövnings platser (work, public, free) som kan vara överlappande.

Användare

Applikationens målgrupp, kunden (modulering av övningar)

Målsättning

Skapa en androidaktivitet som visar en lista av övningssekvenser för användare samt underlättar val av lämplig sekvens.

Förslag på lösning

Visa tre listor i samma aktivitet som sorteras efter ömsesidigt uteslutande element, varje fragment i listan skall motsvara en övningssekvens och innehålla namn, varaktighet och kort beskrivning. För att underlätta navigering kan en sökfunktion implementeras.

5.4. Fjärde iterationen – Revision av feedback av tidigare iterationer

Arbetsområde

Revision av tidigare iterationer baserat på noterade brister i kapitel 7.2.1 och 7.3.1

Användare

Applikationens målgrupp.

Målsättning

Utför en detaljerad revision av listan på negativ feedback i kap. 7.2.1 och 7.3.1. Implementera föreslagna lösningar efter en diskussion med kunden då funktionella förändringar kan förekomma.

Förslag på lösning

Uppspelningen fungerar inte när applikationen körs i backgrunden

För att Android skall kunna fortsätta exekvering när en applikation flyttas till backgrund antingen av systemet eller av användaren så måste en Service implementeras. En Service kan i sin tur använda sig av en Notification som utgör ett interface mot användaren när applikationen befinner sig i backgrunden.

Hanterar inte Audio Focus

Avsikten med Audio Focus är att hantera interferens med ljuduppspelning i applikationen. När en annan entitet behöver tillgång till ljuduppspelnings hårdvaran kan Android notifiera applikationen om en sådan förfråga. Den enklaste lösningen är att försätta uppspelningen av övningssekvensen i paus tillstånd.

Utseendet av gränssnittet är provisoriskt

Endast efter en fastställning av funktionalitet skall det grafiska interfacet få sitt slutliga utseende. Spelaren har en central roll i applikationen och borde få en komplett utvärdering under prototyptest.

Eventuella stabilitetsproblem

Med stöd av analysen i kap. 7.2 har en diskussion bedrivits med kunden, där man har bestämt följande: avstå från två parallella ljudtrådar och låta användaren göra ett val innan uppspelningen startar, antingen en musiktråd eller en rösttråd där den nuvarande röstljudfilen ersätts med en ljudfil som innehåller både backgrundsmusik och röst.

Hanterar inte ljudutgångar på ett sakligt sätt exempelvis växling mellan hörlurar och högtalare

Via AudioManager kan man avgöra vilka ljudutgångar används och vidta nödvändiga åtgärder. Efter en diskussion med kunden, flyttas kravet till fullversionen istället. Då det saknar direkt relevans vid prototyputvärdering.

Funktionen nästa/föregående motverkar syftet med applikationen. För att uppnå en stressreducerandeeffekt måste användren uppleva en oavbruten övningssekvens

Ta bort icke önskevärd funktionalitet.

Sorteringen av övningssekvenslistan är inte användarvänlig

Inför förändringar enligt rekommendation i 7.3.1

Ge användaren möjlighet att bläddra genom övningar i sekvenserna

Skapa en ny Activity mellan SessionListActivity och SessionActivity som uppfyller kravet.

6. Design

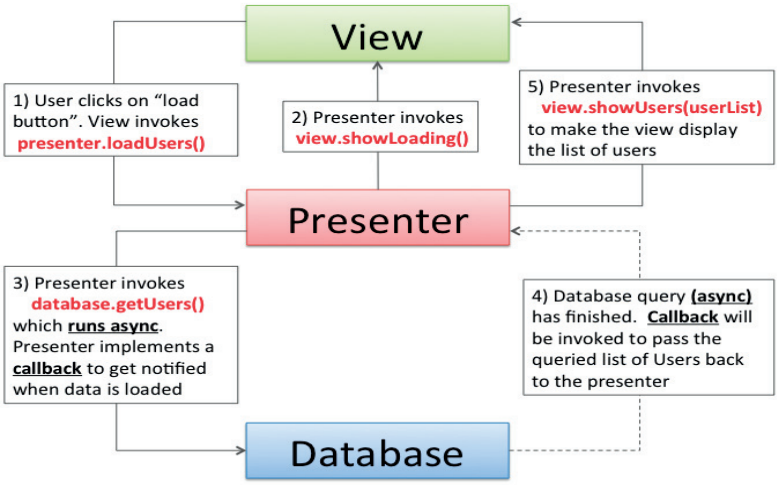
Designstadiet har varit direktkopplat till implementeringsstadiet då möjligheterna och begränsningarna i Android varit föga kända. Varje iteration har inletts med en skiss på papper till den grad som det var möjligt för att sedan implementeras i kod och återkopplas till skissen. Alla klassdiagram samt kodutdrag som refereras i kapitlett hänvisar till slutversionen för att ge läsaren en komplett bild av systemet. Klassdiagrammen är genererade med simpleUML utifrån färdig kod.

6.1. Första iterationen – GUI

Designen av gränssnittets utseende är direkt knutet till dess implementering, först görs en skiss med papper och penna för att sedan uttryckas i XML. Färgpaletten för gränssnittet följer Googles riktlinjer för material design enligt (Google, 2015) och är framtagna av kunden.

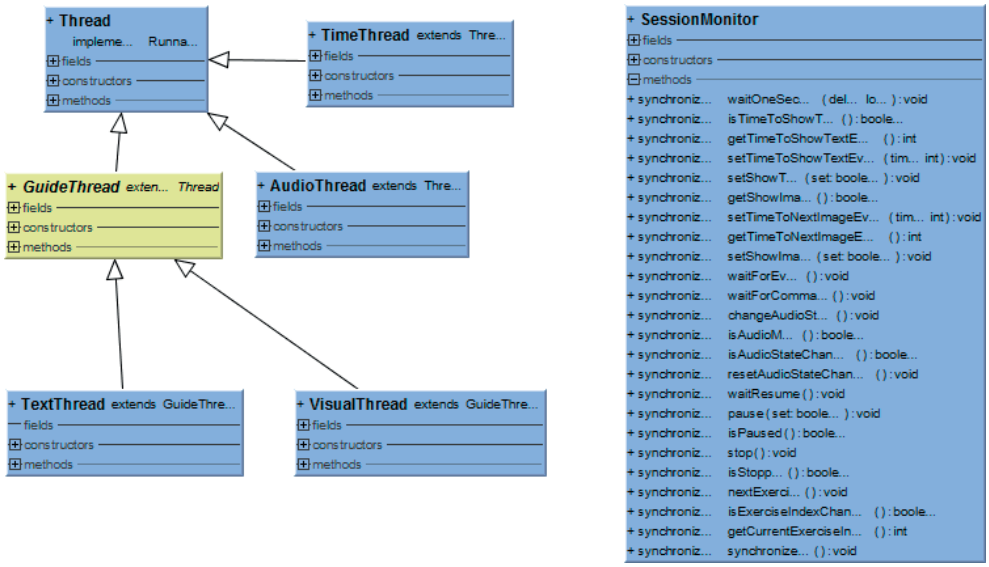
6.2. Andra iterationen - Spelare för övningssekvenser

Inledningsvis analyserades MVP Androids motsvarighet till MVC för att åstadkomma en separation av modellen från användargränssnittet. Den fundamentala skillnaden mellan dessa är att i fallet med MVP så anropas Presenter vid tillståndsförändringar i Model till skillnad från MVC där Model observeras direkt av View se figur 4.1. MVP principen kommer att gälla för hela applikationen.



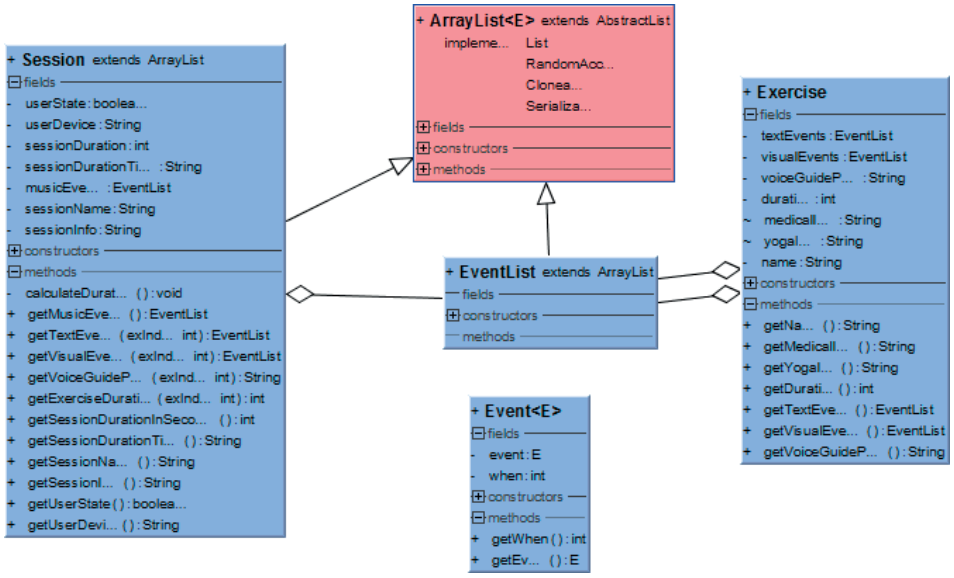
Figur 6.1 exempel på MVP. *Database motsvarar Model i texten*.

Spelarens karaktär förestår en parallell exekvering som kan implementeras via multipla trådar. En tråd för varje ljudspår samt två trådar som kan användas för visning av text och bildsekvenser. För att synkronisera trådarna skall en monitorklass användas och en femte tråd introduceras för att hålla reda på tiden, en sådan uppdelning uppfyller även principen för enkelt ansvar se figur 4.3 nedan.



Figur 6.3 Trådar och monitor

Första utdraget baserades på Observer mönstret, Model skulle observeras av Presenter och notifiera vid förändringar i sitt tillstånd, i sin tur skulle Presenter uppdatera View. Under implementeringen visade det sig vara felaktigt, då Android inte tillåter uppdatering av View från externa trådar vilket delvist framgår utav följande exception (*ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views*) mer om detta kan läsas här (The official site for Android developers, 2016). Lösningen blev en Handler klass som möjliggör kommunikationen mellan externa trådobjekt och UI tråden. För test av spelaren designades även ett Event, Exercise och Session objekt som används vid trådstyrning se figur 4.4. Vi får en följande struktur: Exercise innehåller listor med Event objekt som styr trådarna, varje Session byggs upp av en eller fler Exercise objekt. Val av datastrukturen för att lagra Event objekt baseras på dess flexibilitet och kan ses som temporär.



Figur 6.4 Session, Exercise och Event objekt

Under den givna iterationen designades även ett hjälpverktyg AssetRetriever för att hantera applikationens resurser som exempelvis bilder, textfiler och ljudfiler då Android erbjuder en förbestämd katalogstruktur som saknar en nödvändig överskådlighet och flexibilitet.

6.3. Tredje iterationen – Presentation av övningssekvenser

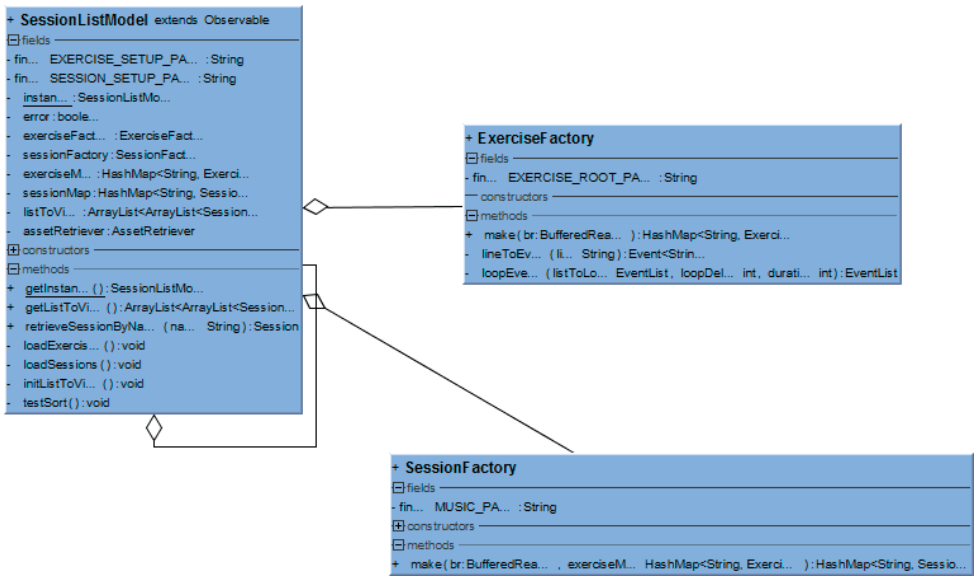
Primärt analyserades två olika möjligheter att lagra övningar och övningssekvenser lokalt på enheten. Det första och antagligen mest uppenbara är en lokal databas på enheten och det andra är en textfil. Preferensen hamnar hos textfilen baserat på följande: lätt att tolka och modifiera för kunden och enklare att implementera. Databaslösningen kan däremot bli aktuell vid vidareutveckling av applikationen.

Här tas även beslutet om att använda två olika textfiler en för att modulera övningar (exercise_setup.cnf) och en för sekvenserna (session_setup.cnf) se figur 6.5 nedan.

```
Neck roll
neck_roll.mp3
235
#end
Stimulates thyroid gland, releases jaws, releases
headaches, decreases blood pressure
#end
For inner peace, sleep, deep relaxation
#end
Sit with your back straight. Place your hands on knees,
palms facing down.#0
Lower your chin to your chest and let the tip of the
chin follow the collarbone up to the left shoulder.#10
Lift the chin and make a demi circle with over to the
right shoulder.#20
Let the chin follow the right collarbone down to the
chest.#30
When you lift the chin, open your mouth thoroughly#40
and when you lower the chin, let your jaw relax#50
Breath in when you lift the chin and breath out when you
lower#60
Perfect exercise 2 min each way, before bedtime if you
grind your teeth at night#70
#loop#10
neck_roll_1#0
#end
#endfile
```

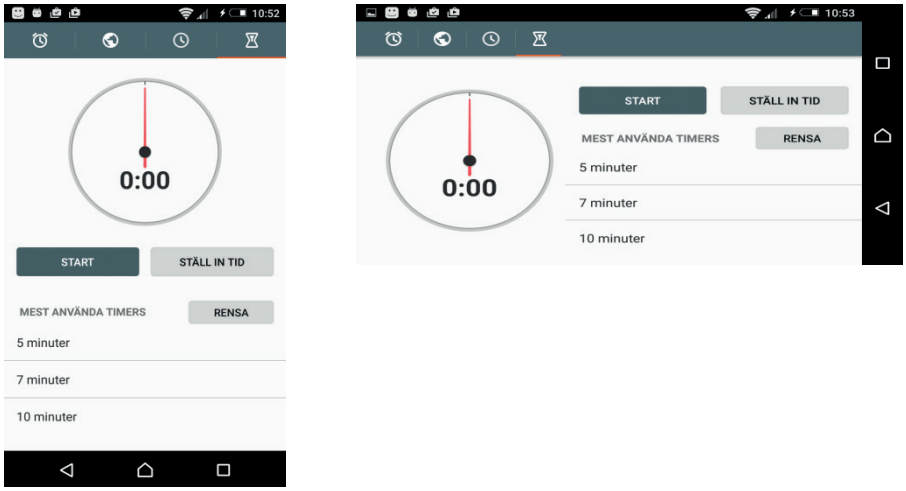
Figur 6.5: Utdrag ur exercise_setup.cnf

Filerna skall läsas in av respektive klass SessionFactory och ExerciseFactory och motsvarande objektlistor skapas. För hantering av listorna introduceras SessionListModel som ansvarar för sortering och lagring av listobjekten se figur 6.6 nedan.



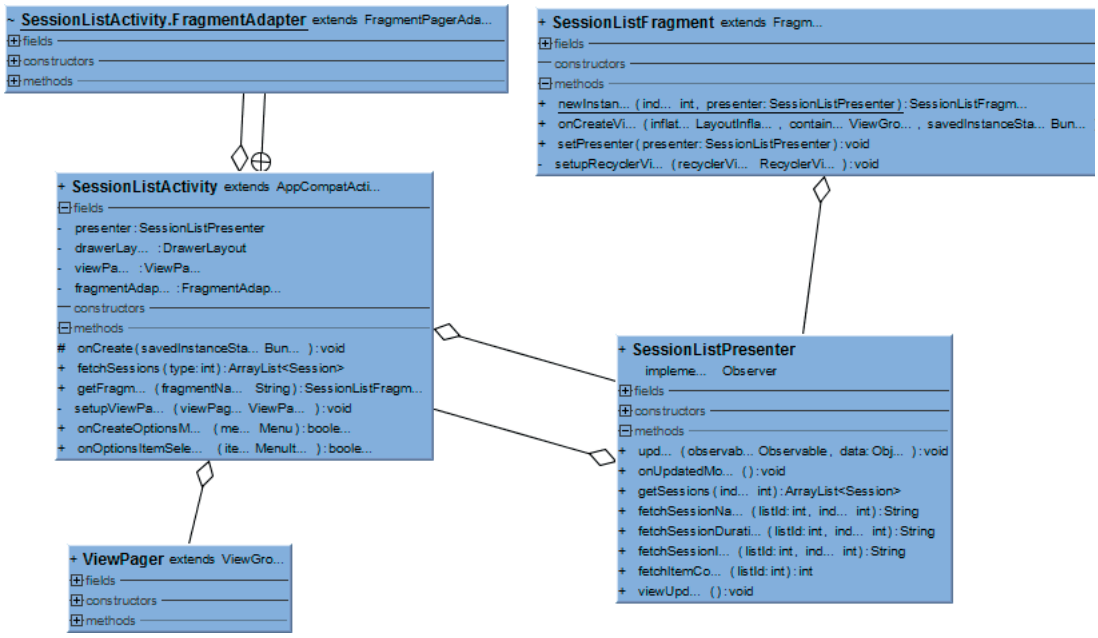
Figur 6.6 SessionListModel, ExerciseFactory och SessionFactory

SessionListModel observeras av Presenter men under nuvarande implementation så utsätts inte modellen för någon förändring så att mönstret är av en antydande karaktär. Här utförs även en primär analys av den grafiska presentationen, valet hamnar på fragment. Fragment kan förklaras som multipla Activity objekt inne i en parent Activity. Varje fragment implementerar en egen livscykel som är endast beroende av parent, konstruktionen kan ses som modulbaserad och öppnar upp nya möjligheter. Fragment kan uppdateras oberoende av varandra utan att starta om parent samt återanvändas, men den eventuellt mest spännande egenskapen är deras anpassningsförmåga till användarens hårdvara och skärmorientering se figur 6.7.



Figur 6.7 Samma applikation vid lodrätt respektive vågrätt skärmorientering

Inför designen analyserades Fragmentavsnitt i (Harwani, 2013) och (The official site for Android developers, 2016). Resultatet kan ses i fugur 6.8 nedan.



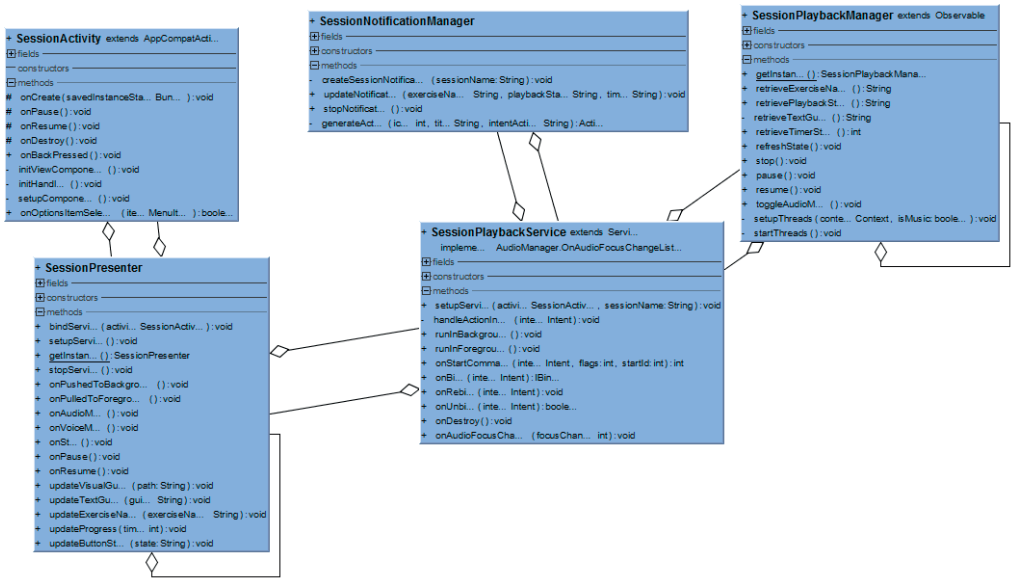
Figur 6.8: Fragmentbaserat GUI för visning av övningssekvenser

6.4. Fjärde iterationen – Svar på feedback i kap. 7.2.1, 7.3.1

Avsikten med iterationen som följer är att analysera feedback i kap. 7.2.1, 7.3.1 och baserat på slutsatser införa korrigeringar i den nuvarande implementationen. Slutsatserna skall baseras på rekommendationer och riktlinjer som är satta i (The official site for Android developers, 2016). I fall att ändringarna påverkar funktionaliteten hos spelaren skall kunden involveras.

6.4.1. Uppspelningen fungerar inte när applikationen körs i backgrunden

Efter fördjupade studier i beteendet av Service och Notification kan följande klassdiagram skapas, se figur 6.9. En sådan design underlättar en vidare utökning av spelaren, man kan på ett enkelt sätt utöka SessionPlaybackService till att stödja andra användargränssnitt så som Android Wear eller Android TV.



Figur 6.9 Klassdiagram av sekvensspelaren

Tanken bakom designen är följande: SessionPlaybackService har en SessionNotificationManager som hanterar notifikationer när applikationen befinner sig i backgrunden och en SessionPlaybackManager som hanterar uppspelningen av övningssekvenser via trådar. SessionPlaybackService är bunden till motsvarande Activity via en Presenter enligt MVP och växlar in/utmatning mellan View och Notification beroende på applikationens tillstånd (background/foreground).

6.4.2. Hanterar inte Audio Focus

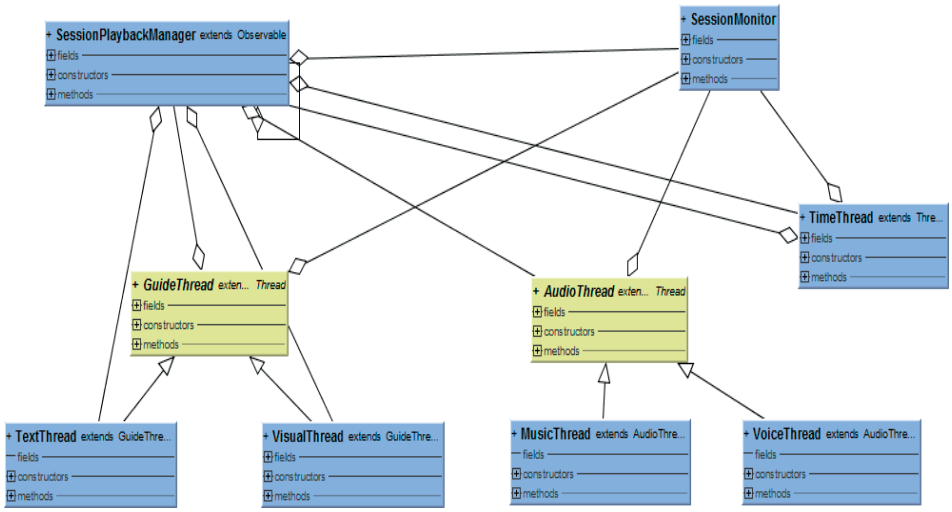
Som framgår utav figuren 6.8 så implementerar SessionPlaybackService OnAudioFocusChangeListner vilket löser problemet.

6.4.3. Utseendet av gränssnittet är provisoriskt

Denna punkt flyttas direkt till implementeringsfasen och skall hanteras sist i kapitel 7.4 då designen skall baseras på funktionaliteten hos spelaren. Tanken är att först göra en skiss på papper för att sedan uttryckas i XML vilket kommer att resultera i en kontinuerlig återkoppling mellan de två faserna och feedback från kunden.

6.4.4. Eventuella stabilitetsproblem

Här följer vi rekommendationerna från kap 5.4 och ersätter parallell exekvering av ljudtrådar med en tråd i taget, valet som användaren skall göra inför uppspelning placeras i en ny klass SessionInfoActivity se figur 6.10. Den slutliga versionen av övningsspelaren presenteras nedan i figur 6.91.



Figur 6.91: Övningsspelaren

6.4.5. Funktionen nästa/föregående motverkar syftet med applikationen. För att uppnå en stressreducerandeeffekt måste användren uppleva en oavbruten övningssekvens

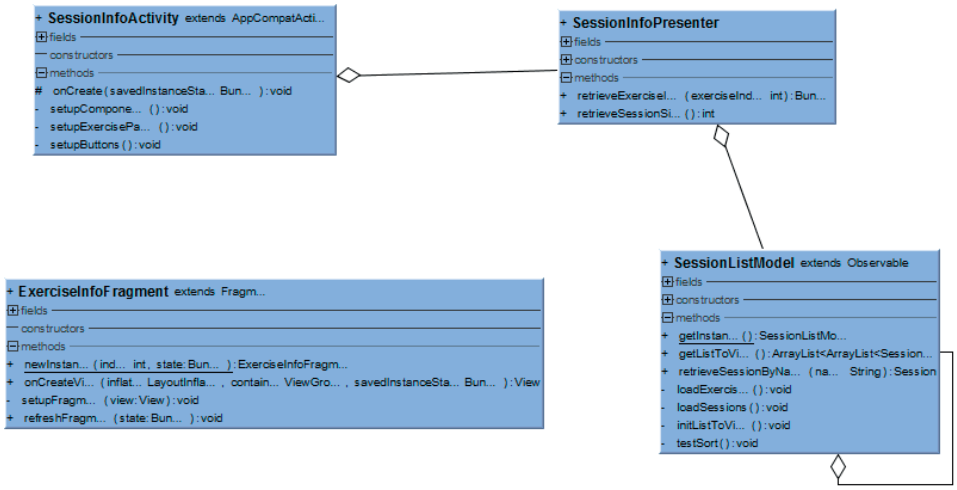
Här går vi direkt till implementeringsstadiet och rensar bort både XML och klassfiler på icke önskevärd funktionalitet.

6.4.6. Sorteringen av övningssekvenslistan är inte användarvänlig

Även detta krav går direkt till nästa stadie då det handlar om mindre förändringar som inte involverar applikationens arkitektur.

6.4.7. Ge användaren möjlighet att bläddra genom övningar i sekvenserna

Denna funktionalitet implementeras tillsammans med föregående i SessionInfoActivity, som använder sig av SessionListModel se figur 6.92 nedan.



Figur 6.92: SessionInfoActivity

7. Implementering

All implementering under projektet är gjord i Android Studio. Valet av verktyg baseras på följande: framtaget av Google som Android, endast riktat till Android vilket ofta ger utökad funktionalitet och stabilitet. Verktyget är ofta refererat till både på webben och i litteraturen samt är gratis. Varje iteration avsluttas med ett feedbackkapitel som baseras på en analys av det implementerade momentet tillsammans med kunden. All kod som anses relevant presenteras i Appendix C.

7.1. Första iterationen – GUI

Inledningsvis analyserades möjligheterna av de grafiska biblioteken framtagna av Google som är inkluderade i Android Studio samt riktlinjer för Materialdesign för att sedan återvända till designstadiet för en skiss. Fjärde utdraget presenterades för kund.

7.1.1. Feedback – GUI

Positivt:

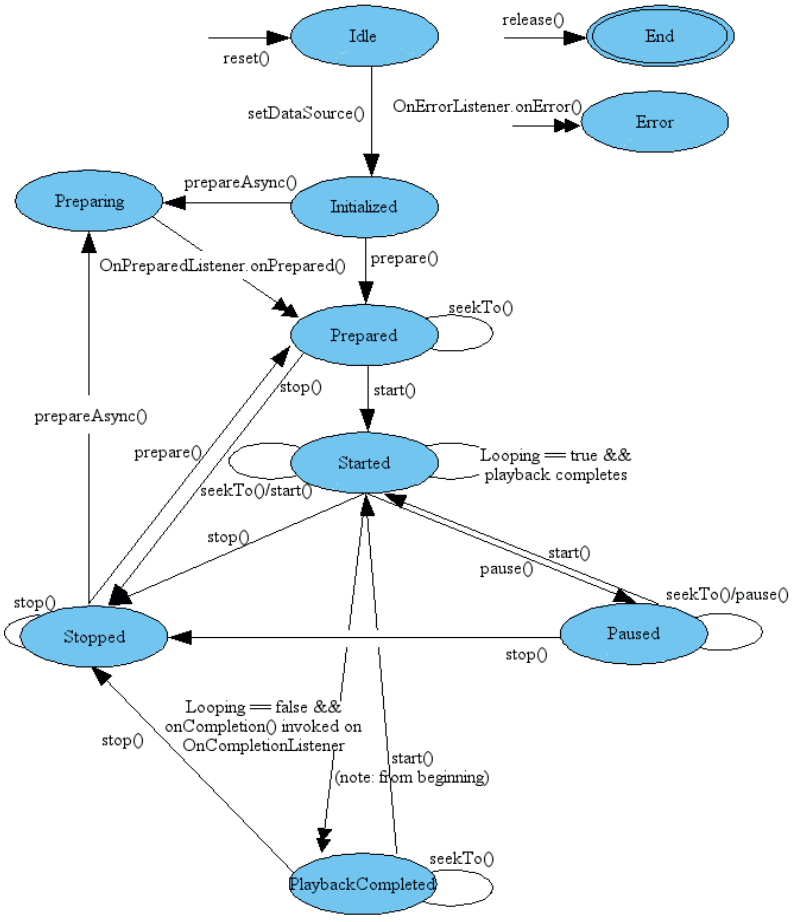
- Dominerande naturbilderna som följer applikationens stressreducerandekoncept.
- Retoriken i applikationen är behaglig och mer antydande än direkt.
- Nertonade och transparenta element som knappar och menyer.

Negativt:

- Texterna är något svåra att urskilja mot en varierande natur
- Retoriken kan skapa användningsproblem då den kan bli svårtolkad
- Dominant menyrad som skär sig med omgivande design

7.2. Andra iterationen - Exekvering av övningssekvenser

Som stöd under hela implementeringen användes ett tillståndsdiagram för Androids ljudspelare MediaPlayer i något förenklad form se figur 5.1 nedan med tillhörande dokumentation (The official site for Android developers, 2016). Implementerade funktioner består av följande: spela, pausa, nästa/föregående övning, stumt tillstånd för varje ljudtråd.



Figur 7.1 Tillståndsdiag. MediaPlayer

Processen inleddes med att skapa ett enkelt gränssnitt för spelaren med tillhörande Presenter klass endast i testsyfte. Presenter möjliggör kommunikation med trådobjekten på följande sätt:

- View initierar WeakReference objekt som kan liknas vid en pekare och refererar till GUI element som textfält och bildfält.
- För att hantera kommunikationen från modellens trådar med UI tråden skapas två Handler klasser. Klasserna har tillgång till WeakReference objekten och kan modifiera GUI via dessa.
- Handlerobjekten skickas till modellen och distribueras till respektive tråd

SessionPlayer klassen initierar och startar trådarna samt hanterar kommunikationen med dessa och Presenter.

För att undersöka beteendet hos trådar skapades först en tråd för uppdatering av textfält och en för uppspelning av ljudfiler samt en monitor klass för synkronisering, för att underlätta ytterligare ritas tillståndsdiagram för varje tråd. För att bibehålla principen om enkelt ansvar introducerades en tredje tråd som håller reda på tiden under exekveringen och triggar resterande trådar. Systemet fungerade korrekt tills introduktionen av en andra ljudtråd, vid samtidig uppstart av ljudtrådarna spelade enheten endast upp en av ljudfilerna. Om vi tittar närmare på tillståndiagrammet för MediaPlayer figur 5.1 och tillhörande dokumentation (The official site for Android developers, 2016) kan vi utläsa en fördröjning mellan Initialized och Prepared tillstånd (ca 200ms, enhetsberoende) men ingen fördröjning är angiven för övergången mellan Prepared och Started. För att kringgå detta problem infördes en fördröjning på 300ms mellan trådarna vid tillståndsövergången som realiseras via en sorts FIFO sluss se figur 5.2, längden på tidsintervallet baseras på Androids standard GUI fördröjning som är 250ms.

```
private void initializeAudio() {//see commentary in the MusicThread
    mon.startPreparingAudio();
    audio = assetRetriever.loadMediaFromAsset(session.getVoiceGuidePath(0))
    try {
        audio.prepare();
    } catch (IOException e) {
        e.printStackTrace();
    }
    audio.setVolume(0, 0);//mutes the audio glitch
    mon.preparingDone();
    mon.waitUntilAllPrepared();

    mon.startAudio();
    audio.start();
    mon.waitForStartedState();

    mon.doneStarting();
    audio.pause();
    mon.synchronizeAll();
}
```

```
private boolean preparingAudio = false;
private boolean preparingMusic = false;
private int preparedThreads = 0;

//synchronize on prepared (audio)
synchronized public void waitUntilAllPrepared() {
    try {
        while (preparingAudio && preparingMusic) {
            wait();
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

synchronized public void preparingDone() {
    preparedThreads++;
    if (preparedThreads == 2) {
        preparedThreads = 0;
        preparingAudio = false;
        preparingMusic = false;
        notifyAll();
    }
}

synchronized public void startPreparingAudio() {
    preparingAudio = true;
}

synchronized public void startPreparingMusic() {
    preparingMusic = true;
}

//synchronize on started (audio)
private boolean audioStarting = false;

synchronized public void startAudio() {
    if (!audioStarting)
        audioStarting = true;
    else {
        try {
            while (audioStarting) {
                wait();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

Figur 7.2 Övre bild visar uppstartsekvensen för ljudtrådarna. Nedre bild visar motsvarande process i monitorn.

Trots att systemet fungerar kan vi observera följande problematik. En entitet som MediaPlayer är relativt resurskrävande samtidigt som fördröjningsintervallet på 300ms är godtyckligt d.v.s. kan variera baserat på målmaskin. Utifrån dessa observationer kan vi förutse stabilitetsproblem dels i form av hantering i OS exempelvis Audio Focus och även problem med tung belastning av både minne och CPU vilket eventuellt kommer att leda till att Android avslutar en av MediaPlayer instanser. Även en kvalitetsaspekt skall betraktas i form av smidig exekvering då vi får en ökad CPU belastning. Ett annat argument är att det inte finns något färdigt system inom Android som hanterar den typen av parallell exekvering vilket kan tyda på att det inte är välkommet, inte heller något exempel på liknande lösning har kunnat hittas. För en komplett och slutgiltig implementering av spelaren se kapitel 7.4.

7.2.1. Feedback – Exekvering av övningssekvenser

Positivt:

- Det fungerar.

Negativt:

- Uppspelningen fungerar inte när applikationen körs i backgrunden.
- Hanterar inte Audio Focus.
- Utseendet av gränssnittet är provisoriskt.
- Eventuella stabilitetsproblem.
- Hanterar inte ljudutgångar på ett sakligt sätt exempelvis växling mellan hörlurar och högtalare.
- Funktionen nästa/föregående motverkar syftet med applikationen. För att uppnå en stressreducerandeeffekt måste användren uppleva en sekvens i sin helhet.

7.3. Tredje iterationen - Presentation av övningssekvenser

Iterationen inleds med implementering av modellen enligt klassdiagrammet i respektive designkapitel. Först skapas en HashMap<String, Exercise> med hjälp av ExerciseFactory som läser in en fil med övningsbeskrivning, nyckeln i mappen motsvarar namnet på övningen. I nästa steg skapas motsvarande struktur HashMap<String, Session> som innehåller övningssekvenser eller sessions. Utifrån den sistnämnda mappen skapas en lista för visning av View, ArrayList <ArrayList<Session>> där index motsvarar ömsesidigt uteslutande tillbehör närmare bestämt {all, mat, chair, none} med {all} som undantag.

Nästa fas är implementering av det grafiskagränssnittet som kommer att vara fragmentbaserad. Vi skall försöka föreställa oss ett sådant system via olika lager, första lagret består av en Activity som har en motsvarande layout som innehåller en ViewPager med tillhörande FragmentPagerAdapter. FragmentPagerAdapter är ett interface för att kunna kommunicera mellan Fragment och ViewPager. Fragment eller i vårt fall SessionListFragment har en egen layout bestående av endast en RecyclerView. SessionListFragment har en liknande Adapter för att manipulera RecyclerView som i sin tur innehåller ett antal ViewHolder objekt som slutligen representerar våra övningssekvenser se klassdiagrammet i figur 4.7. Då det endast finns ett litet antal övningssekvenser som skall utvärderas under prototyptest och sorteringen ut av dessa är inte definitiv finns det ingen anledning att implementera någon sökfunktion i nuläge.

7.3.1. Feedback - Presentation av övningssekvenser

- Sorteringen av övningssekvenser enligt ömsesidigt uteslutande är inte användarvänligt, däremot en sortering enligt användares tillstånd {work, public, free} (se figur 4.4, userState) skulle vara mer lämplig trots att det skapar en redundans i listan då vi har att göra med överlappande mängder.
- Använda funktionaliteten i den fragmentbaserade designen på ett sakligt sätt i fullversionen.

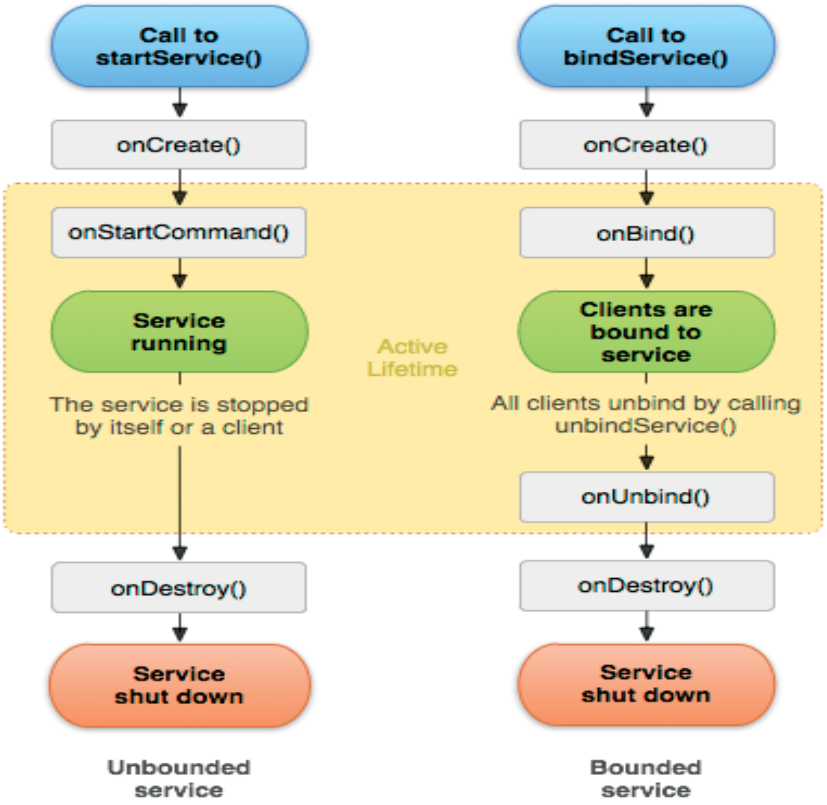
- Ge användaren möjlighet att bläddra genom övningar i sekvenserna.

7.4. Fjärde iterationen - Svar på feedback i kap. 7.2.1 och 7.3.1

Beskrivningen av processen kommer att följa designen i kapitel 6.4 med samma indelning för att underlätta för läsaren.

7.4.1. Uppspelningen fungerar inte när applikationen körs i backgrunden

En Service klass är väldigt snarlik en Activity och måste deklareras separat i AndroidManifest.xml och har en egen livscykel. För att göra kommunikationen möjlig mellan Service och Activity måste vi binda Service via Presenter till Activity vilket hanteras av ett ServiceConnection objekt som initieras lokalt i Presenter. Service implementerar ett interface som innehåller onStartCommand(Intent intent,…..) metod och som det framgår utav namnet tar metoden emot kommando från andra klasser. Det innebär inte att man inte kan anropa andra metoder i Service men är däremot rekommenderat att använda sig av, dels beroende på hanteringen utav OS när applikationen körs i backgrunden. Här använder vi oss av en bunden Service baserat på att den inte behöver leva utan applikationen, för en bättre översikt av livscykeln se figur 7.3 nedan.



Figur 7.3: Service livscykel (The official site for Android developers, 2016)

Service håller reda på sitt background/foreground tillstånd via en flagga och dirigerar om utmatning och inmatning baserat på denna. Utmatning precis som inmatning kan ske på två sätt, antingen via View eller via en Notification som hanteras av en NotificationManager. Uppspelningen av övningssekvenser hanteras av SessionPlaybackManager och sker på följande sätt: Trådarna initieras och startas, styrkommando förmedlas till trådarna via monitor, all utmatning sker antingen på begäran eller baserat på tidsmarkeringar som är lagrade i Session objektet. När uppspelningen avslutas dödas trådarna via monitor och vi får en komplett livscykel.

Uppspelningen kan avslutas på två sätt antingen via användaren eller när övningarna är slut, först stoppas uppspelning sedan Service och slutligen SessionActivity.

7.4.2. Hanterar inte Audio Focus

För att hantera AudioFocus låter vi SessionPlaybackService implementera AudioManager.OnAudioFocusChangeListener interface som består av en metod nämligen onAudioFocusChange(int focusChange) där focusChange beskriver tillståndet hos AudioFocus. Uppspelningen försätts i Paused tillstånd i alla utfall förutom när applikationen återfår fokus.

7.4.3. Utseendet av gränssnittet är provisoriskt

Utseendet av gränssnittet anpassas enligt riktlinjer i kapitel 5.1, för slutversion se kapitel 9.

7.4.4. Eventuella stabilitetsproblem

Här låter vi användaren göra valet i SessionInfoActivity som beskrivet i kapitel 6.4.4 och skickar en boolean variabel via Intent till SessionActivity. Själva valet av ljudtråd hanteras i SessionPlaybackManager vid initiering av trådar.

7.4.5. Funktionen nästa/föregående motverkar syftet med applikationen. För att uppnå en stressreducerandeeffekt måste användren uppleva en oavbruten övningssekvens

Tar bort onödig funktionalitet, den stora omstruktureringen av övningsspelaren minskar arbetsbelastning i detta steg. Ingen djupare beskrivning anses nödvändig.

7.4.6. Sorteringen av övningssekvenslistan är inte användarvänlig

Här ändras sorteringen i ArrayList<ArrayList<Session>> listToView i SessionListModel enligt förslaget i avsnitt 7.3.1 och fliknamnen i SessionListActivity.

7.4.7. Ge användaren möjlighet att bläddra genom övningar i sekvenserna

Inledningsvis skapas en skiss på papper för att sedan uttryckas i XML, layouten innehåller en ViewPager som i sin tur består av Fragment som motsvarar varje enskild övning. Samtidigt innehåller den fullständig beskrivning av hela sekvensen och två knappar som startar nästa Activity nämligen SessionActivity. SessionListModel används även här via Presenter.

8. Test och validering

Test och validering har skett löpande enligt den iterativa processmodellen som kan följas i feedbackavsnitten efter varje avslutad iteration som beskrivs i kapitel 5. Kunden har varit direktinvolverad i båda faserna och lämnat en muntligredogörelse utifrån egna iakttagelser. Inga testscenario har skapats då den ovannämnda metoden antas vara tillfredsställande inom ramar för prototyputveckling.

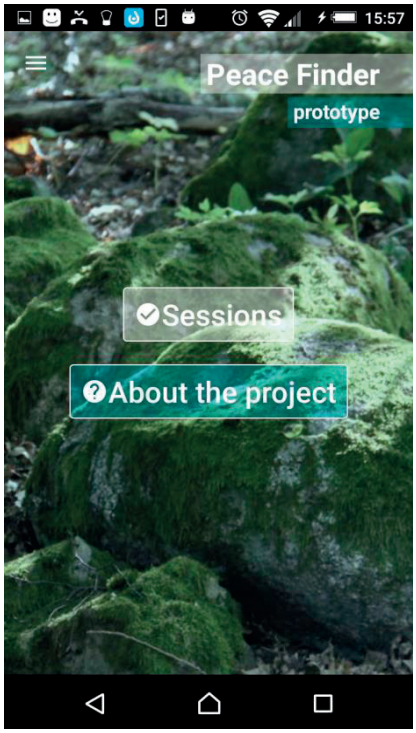
9. Resultat

Som resultat av arbetet har vi en färdig prototyp för utvärdering. Utvärderingen kan ske obevakat och under en obegränsad tid för att skapa underlag för en intervju med testpersoner. Protypen ger möjlighet att utvärdera följande moment:

- Layout av interface
- Effekten av övningar
- Användarvänlighet

Prototypen består av följande delar:

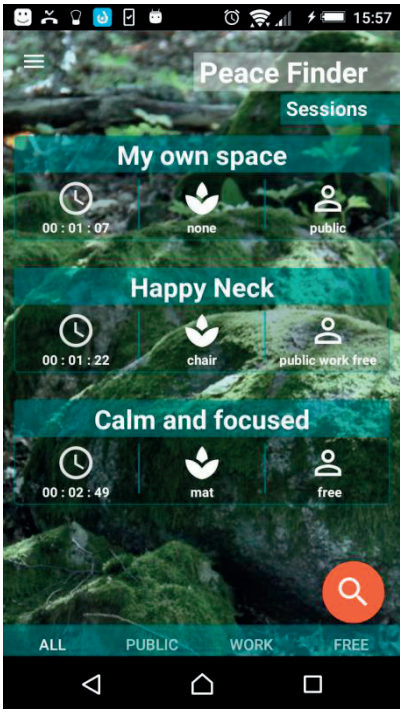
- Introduktion, fig 9.1



Figur 9.1: MainActivity

Här har användaren två möjligheter, antingen läsa om projektet eller fortsätta till övningslistan.

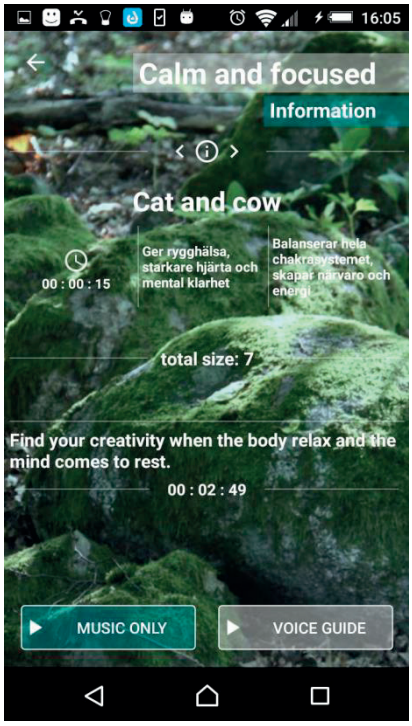
- Övningslista, fig 9.2



Figur 9.2: SessionListActivity

Här listas övningssekvenserna baserat på användarens tillstånd (All, Public, Work, Free), PageViewer ger användaren möjlighet att traversera listorna i en och samma Activity och välja en övningssekvens.

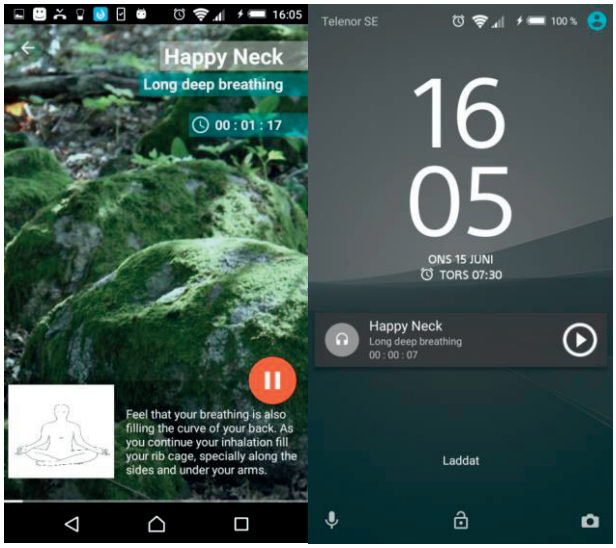
- Information om övningar, fig 9.3



Figur 9.3: SessionInfoActivity

Baserat på valet i föregående steg listas information om den valda övningen samt ges en möjlighet för användaren att gå igenom de olika momenten i övningssekvensen. I botten av skärmen finns två knappar som ger användaren möjlighet att välja om sekvensen ska ha en röstguide eller endast bakgrundsmusik.

- Övningsspelare (SessionActivity), fig 9.4



Figur 9.4: SessionActivity (vänster/höger, foreground/background)

Slutligen hamnar vi i uppspelningsdelen av applikationen. Användaren har möjlighet att pausa eller återuppta uppspelning samt spela upp sekvensen medan telefonen befinner sig i standby tillstånd via SessionPlaybackService. Uppspelningen avslutas när sekvensen är slut eller på användarens begäran och återvänder till föregående steg.

Kunden har även en möjlighet att införa förändringar via cnf filer som formar övningsförloppet vid en tidig återkoppling med testpersoner. Här kan även fler övningar läggas till av kunden under utvärderingsfasen på ett relativt enkelt sätt. Den typen av löpande justering skulle eventuellt kunna bidra till optimering av den stressreducerande effekten.

Utvärdering av prototypen kommer att ske under bevakning av kunden med löpande feedback från testpersoner under 1-2 månaders period, därefter kommer flera strukturerade intervjuer bedrivas med testpersoner. Intervjustrukturen kommer att baseras på feedback som mottas av kunden och det redan extraherade funktionaliteten.

Kunden har redan idag bestämt sig för att slutföra projektet baserat på detta arbete vilket i sin tur garanterar stöd under hela testfasen.

10. Slutsats

Processen som är beskriven i denna rapport har varit strukturerad och följt en standard som är beskriven i (Lauesen, 2002). Valet av en iterativutvecklingsmodel visade sig vara lämpligt för den typ av applikation där funktionen och syftet förfinades med kundens engagemand i varje iteration. Återkopplingen mellan design och implementeringsstadiet minskade i takt med inlärning och de senare iterationerna var linjära. MVP strukturen gav stora fördelar när det grafiska interfacet skulle förändras då en total separation av modellen var uppnådd. Även resterande arkitektur bestående av designprinciper som singelton, factory och principen för enkelt ansvar levererade sina fördelar under hela förloppet. Som resultat får vi en prototyp som fungerar utan större anmärkningar.

10.1. Svar på frågeställningen

I detta avsnitt kommer det att ske en utvärdering av den frågeställning eller problemformulering som är gjord i kapitel 1.4. För att underlätta för läsaren listas svaren i samma ordning som i problemformuleringen.

- Kravspecifikation: Arbetet med kravspecifikationen under projektet följer den iterativa modellen, vilket innebär att en ny kravspecifikation upprättas för varje iteration baserat på funktionalitetslistan i kapitel 4.6 och följer Task and support standarden som är beskriven i (Lauesen, 2002).
- Sensorer: I kapitel 4 genomförs en analys av den önskevärda och möjliga funktionaliteten hos applikationen som på vissa punkter strider mot den initiala problemformuleringen. Trots att inget direkt behov av att utvärdera användarenstillstånd med hjälp av sensorer har uttryckts, har följande analys bedrivits. I rapporten har fastställts vilken symptombild som kan läsas med de sensorer som finns på marknaden idag, se kapitel 4.2.2.1. Samtidigt inser man att komplexiteten i en sådan algoritm kräver experter som inte finns på företaget. Eventuellt skulle man kunna ta fram en lösning i samarbete med stressexperter inom medicin.
- Strömma media: Behov av tillgänglighet och avbrottsfri uppselning strider mot strömning av media som innehåller övningssekvenser.

Däremot strömning av hjälpvideo och annat material som inte inverkar direkt på övningsförloppet skulle kunna implementeras men hamnar utanför avgränsningen för prototypen.

- Lämplig utformning: Inför intervjuer gjordes en analys av nuvarande arbetssätt och liknande produkter samt dokumentsstudie på företaget. Detta utgjorde basen för intervjuerna utan att ge förutfattade beslut om produkten.
- Välja protokoll/format för strömmande media: Den iterativa modellen strider mot att ta tidiga beslut då ingen strömning ingår i prototypen.
- Välja en databas/server för media filer: se föregående svar.
- Välja utvecklingsverktyg för applikationen: Android Studio blev prioriterad, för motivering se kapitel 3.2.3.
- Androidapplikation bunden till databas: Eliciteringsprocessen har ändrat den initiala uppfattningen om produkten. Som resultat har vi en prototyp som med hjälp av en flertrådig process möjliggör uppspelning av expertmodulerade övningssekvenser. Prototypen ger även möjlighet för kunden att med ett mindre stöd från utvecklaren modifiera övningssekvenser via cnf filer. En annan viktig del är att testpersonen inte behöver en ständig bevakning utan kan använda sig av prototypen under en längre tid för en mer saklig återkoppling, vilket även önskas av kunden.
- Kopieringsskydd: Applikationsresurserna är inte tillgängliga för läsning eller kopiering.

10.2. Utvecklingsmöjligheter

Här kommer vi endast att diskutera vissa möjligheter eller snarare idéer kring den redan implementerade funktionaliteten.

10.2.1. Redundans i övningssekvenser

För att användaren skall uppnå en önskevärd effekt behövs ett frekvent användande av applikationen vilket i sin tur kan leda till frekvent upprepning av övningssekvenserna och som resultat ett minskat användande. Som en möjlig lösning på problemet skulle man kunna ändra datastrukturen i sekvenserna till en graf istället för en lista. De alternativa

vägarna skulle minska redundansen i övningarna och motivera till användning.

10.2.2. Retoriken i applikationen

Reducera yoga-terminologin i applikationen för att nå en bredare målgrupp.

10.2.3. Underlätta för kunden att modulera övningar

Skapa ett webbaserat verktyg för att göra det möjligt för kunden att underhålla en databas med övningssekvenser. En sådan lösning kan även gynna andra önskemål från kapitel 4.6.

10.2.4. Alternativa gränssnitt

Ge användaren en möjlighet att använda en tv eller Android Wear för att förbättra upplevelsen.

11. Terminologi

JVM – Java Virtual Machine

JRE – Java Runtime Environment

API – Application Programming Interface

SDK – Software Development Kit

MVC – Model View Control

MVP – Model View Presenter

UI – User Interface, refererar oftast till huvudprocess/tråd inom Android

Handler – Används vid kommunikation mellan trådar/processer i ett Androidsystem

SimpleUML – UML plugin för Android Studio

Audio Focus – Mekanism för hantering av multipla ljudkällor i Android

Intent – En abstrakt data struktur som ofta används inom Android

XML - Extensible Markup Language

12. Källförteckning

Mediyoga, 2016. *https://www.mediyoga.se.* [Online]
Available at: https://www.mediyoga.se
[Använd 01 04 2016].

Dropbox, 2016. *https://www.dropbox.com.* [Online]
Available at: https://www.dropbox.com
[Använd 01 06 2016].

Team Viewer, 2016. *https://www.teamviewer.com.* [Online]
Available at: https://www.teamviewer.com
[Använd 01 03 2016].

Sony, 2016. *https://www.sony.com.* [Online]
Available at: https://www.sony.com
[Använd 01 01 2016].

Vandrico Inc, 2015. *http://vandrico.com.* [Online]
Available at: http://vandrico.com
[Använd 02 10 2015].

Engqvist, E., 2015. *Instruktörsutbildningskompendium*. Mediyoga International AB.

Engqvist, E., 2015. *Stresskompendium*. Mediyoga International AB.

Google, 2015. https://material.google.com/. [Online]
Available at: https://material.google.com/
[Använd 02 12 2015].

Harwani, B., 2013. *Android Programming.* u.o.:Pearson Education, Inc.

Lauesen, S., 2002. *Software Requirements, Styles and Techniques.* Storbritanien: Pearson Education Ltd..

The official site for Android developers, 2016. *http://developer.android.com.* [Online]

Available at: http://developer.android.com
[Använd 2016].

Saagara, 2015. http://www.saagara.com/. [Online]
Available at: http://www.saagara.com/
[Använd 02 10 2015].

Yoga Relaxation Apps, 2015. http://www.yogarelaxapps.com/. [Online]
Available at: http://www.yogarelaxapps.com/
[Använd 07 10 2015].

Headspace Inc, 2015. http://www.headspace.com/. [Online]
Available at: http://www.headspace.com/
[Använd 06 10 2015].

Simha-studios, 2015. http://www.simha-studios.com/. [Online]
Available at: http://www.simha-studios.com/
[Använd 07 10 2015].

Mind Apps, 2015. http://www. mindapps.com/. [Online]
Available at: http://www.mindapps.com/
[Använd 07 10 2015].

Meditation Oasis, 2015. http://www.meditationoasis.com/. [Online]
Available at: http://www.meditationoasis.com/
[Använd 09 10 2015].

Daily Yoga, 2015. http://www. dailyyoga.com/. [Online]
Available at: http://www.dailyyoga.com/
[Använd 09 10 2015].

Passion Software, 2015. passion.software2013@gmail.com.
Available at: passion.software2013@gmail.com
[Använd 02 10 2015].

13. Appendix A

Applikationsutvärderingstabeller

Pranayama, (Saagara, 2015) Pris: 37kr	Nedladdningar: 5.000 Betyg: 4,3 (167) Senaste uppdateringen: 7 nov. 2014
Beskrivning Centrerat kring andning. Visuella instruktioner till musik med rörliga 3D figurer. Negativt • Designen känns föråldrad • Kvalitén är låg • Engagerar inte användaren Positivt • Andningscirkel som ett visuellt hjälpmedel • Utförlig hjälpsektion Försäljningsteknik • Gratis introduktion (andning på nybörjarnivå)	

YogaNidra, (Yoga Relaxation Apps, 2015) Pris: 13,34kr	Nedladdningar: 1.000 Betyg: 4,1 (53) Senaste uppdateringen: 25 nov. 2013
Beskrivning Meditation till muntliga instruktioner och musik Negativt Låg kvalité Tråkig design Engagerar inte användaren	

Saknar hjälpavsnitt Positivt Valfri/egen musik Försäljningsteknik Gratis introduktion 7 min meditationssekvens, fullversion inkluderar 18 och 25 minuter

Chakra Yoga (Simha-studios, 2015) Pris: Köp i app (7kr - 64,12kr)	Nedladdningar: 10.000 Betyg: 3,9 (306) Senaste uppdateringen: 23 nov. 2013
Beskrivning Yoga övningar bestående av muntliga instruktioner samt rörliga 3D figurer. Övningarna kan sättas i sekvenser. Finns möjlighet att logga sina övningar i en graf. Korta beskrivningar av alla övningar. Utvärdering av ens psykiska/fysiska tillstånd i form av formulär. Väldigt yogainspirerat. Negativt Låg-medel kvalité Icke homogen design, yoga tema korsas med annat. Inget tydligt syfte Reklamfält Positivt Överskådliga övningsexempel Frihet att modellera egna meditationssekvenser Möjlighet att anpassa bakgrund Försäljningsteknik Köp i applikationen bestående av antingen enstaka övningar eller hela sekvenser Reklam i applikationen (3e part)	

Headspace (Headspace Inc, 2015) Pris: prenumeration 12,95$/mån	Nedladdningar: 500.000 Betyg: 4,3 (12434) Senaste uppdateringen: 1 juni 2015
Beskrivning Erbjuder ett komplett levnadskoncept bestående av sekvenser i mindfulness och meditation. Känns modernt och vetenskapligt inga direkta anknytningar till yoga. Funktionell och användarvänlig design. Korta texter med mycket animationer (tecknade). Behandlar hälsa, relationer och intellektuell prestation. Designen i applikationen är riktad mot avslappning och stressreducering, runda hörn, behagliga färger osv. Organisationen donerar en prenumeration för varje såld. Negativt Dyrt Positivt Hög kvalité Kompetent utformning, kort och koncist Fantastisk design Möjlighet att hålla reda på minnesåtgång genom selektiv nerladdning. Konstanta användningstips (icke störande). Möjlighet att bjuda in vänner Försäljningsteknik Prenumeration upp till två år Donerar en prenumeration för varje såld	

Mindfulness (Mind Apps, 2015) Pris: 15kr	Nedladdningar: 10.000 Betyg: 4,0 (282) Senaste uppdateringen: 19 feb. 2015
Beskrivning Väldigt enkel applikation bestående av 6 meditationssekvenser, sorterade	

efter längd med möjlighet att sätta meditationspåminnelse samt föra statistik.

Negativt

Tråkig design

Låg kvalité

Få meditationssekvenser

Positivt

Bra hjälpavsnitt

Enkelt att använda

Försäljningsteknik

Direktförsäljning

Take a Break (Meditation Oasis, 2015) Pris: Gratis	Nedladdningar: 100.000 Betyg: 4,2 (1082) Senaste uppdateringen: 30 juni 2014
Beskrivning Gratisversionen består av 2 meditationssekvenser, 7 respektive 13 minuter. Negativt Dålig design, både grafiskt och strukturellt. Låg kvalité Tråkigt Positivt Utförligt hjälpavsnitt Försäljningsteknik Köp i applikationen länkar till två andra applikationer iSleep, Meditation Complete Course.	

Office Yoga (Daily Yoga, 2015) Pris: prenumeration 4,99$/mån	Nedladdningar: 1.000.000 Betyg: 4,1 (28320) Senaste uppdateringen: 14 juni 2015
Beskrivning Välbyggd applikation med bra design. Innehåller yogaövningar med tal och videoinstruktioner som är uppdelade i övningar, ca 10-15 min (1-2 min/övning). Yogaprogram som är målcentrerade (gå ner i vikt, bättre sömn osv.) och sträcker sig över flera veckor. Negativt Dyrt Positivt Hög kvalité	

Många övningar
Utförliga videoinstruktioner
Engagerande

Försäljningsteknik
Prenumerationen leder till fler ”pro” övningssekvenser och fler musik bibliotek.
Reklam i gratisversionen som dyker upp i varje övningssekvens. Cirka 20% av applikationen är gratis.

Yoga for Beginners (Passion Software, 2015) Pris: Gratis	Nedladdningar: 10.000 Betyg: 4,2 (87) Senaste uppdateringen: 11 feb 2014
Beskrivning En samling videoinstruktioner för yoga utövande. Väldigt enkel implementation, består av sorterade listor av länkar till videoinstruktioner. Negativt Måste vara uppkopplad Tråkigt Positivt Gratis Försäljningsteknik Reklam i applikation	

14. Appendix B

Sammanfattning av gruppintervjuer från 23.06.2015

Fokusgrupp 1

Deltagare: 1 yogainstruktör (YI), 1 avancerad yogautövare(YA), 2 nybörjare yogautövare(YN)

Nuvarande situation

Hjälpmedel:

- Timer (samtliga)
- Musik (samtliga)
- Mantra (YI, YA)

Tekniska hjälpmedel:

- youtube.com -övningsexempel (YN)
- ”Mindfullness” smartphoneapplikation (YN)
- yogobe.com -internetresurs för yogautövare (YN, YA)
- bodyfi.com -träning och hälsoresurs, (YA)

Var:

- Hemma (samtliga)
- Klasser (samtliga)
- Ute i naturen (samtliga)
- Där man finner ro (samtliga)

När:

- Vid behov (samtliga)
- Planerad aktivitet (samtliga)

Svårigheter/problem:

- Svårt att få stöd när man behöver det (YN)
- Minnesbrist på enheterna (samtliga)

Beskrivna scenario:

- Kör förbi ett vackert skogsparti, vill stanna och meditera (samtliga)
- Befinner mig på semester (samtliga)
- Sitter på bussen/tåget upplever ångest, stress eller andra negativa tillstånd (YN)
- Läkare rekommenderade att föra dagbok med beskrivning av mitt fysiska och psykiska tillstånd

Behov/Önskemål

Funktionella:

1. Färdiga övningssekvenser modulerade av experter. (YN)
2. Muntliga instruktioner samt musik under övningar. (YN)
3. Möjlighet att stänga av det muntliga stödet. Grundar sig på betinget (Pavlov) till musiken. (YI, YA)
4. Använda egen musik från telefonen. (YN)
5. Behov av att se sitt tillstånd (fysiska, psykiska) förändras över tiden. Subjektiv bedömning i flera kategorier (sömn, stress, mediciner, osv). Grundar sig på en läkares rekommendation att föra dagbok. (YN)
6. Varierande övningslängd fr. 3 min. (samtliga)
7. Nivågradering av övningar. (YN)
8. Flexibel storlek på applikationen, grundar sig på minnesbrist på enheterna. (samtliga)
9. Behov av att utöva när som helst var som helst. (samtliga)
10. Inga störningar eller avbrott. (samtliga)
11. Schemalagda sekvenser som är målorienterade (bättre sömn på två veckor, bli kvitt ryggsmärtor på en månad, osv)

Grafisk design (GUI):

1. Visuellt: Natur, asymmetri, vågrörelse, behagliga färger. (samtliga)
2. Ljud: Naturtema (hav, regn, fågelkvitter, regn). (samtliga)
3. Exempel på bra design: ”Green kitchen” -matlagnings applikation. (samtliga)

Fokusgrupp 2

Deltagare: Yogaexperter från MediYoga International AB

Nuvarande situation

Hjälpmedel:

- inga

Tekniska hjälpmedel:

- inga

Var:

- Hemma
- Där man finner ro, helst ute i naturen

När:

- Helst fasta tider
- Morgon
- Vid behov
- 1-2 timmar/dag

Behov/Önskemål

1. Nivåindelning bland övningar
2. Applikationen skall behandla både psykisk och fysisk hälsa med tyngdpunkt på stressreducering
3. Övningslängd 3 – 60 minuter
4. Övningstyp som är oändlig (cyklisk) t.ex. andningsövning
5. Möjlighet till korta pauser ”micropaus” 1-2 min mellan övningarna, pausen skall initieras av applikationen
6. Applikationen skall kunna föreslå övningar baserat på situationen som användaren befinner sig i. Situation: tillstånd, tid, plats
7. Möjlighet att mäta puls
8. Instruktioner till övningar skall bestå av ljud och stillbilder
9. Möjlighet att skicka information till användare via applikationen, exempelvis inbjudningar till live events, lyssna av ljudblogg

10. Föra någon form av statistik över användaren
11. Ge användare möjlighet att vara sociala via applikationen
12. Uppmana användaren att sätta telefonen i flygplansläge för att undvika störningar.

Grafisk design (GUI):

- Natur både bild och ljud
- Foto på människor skall vara svartvita
- Asymmetri

Konceptuellt:

- Inga måste (jag måste = stress)
- Bryta det negativa tankemönstret

15. Appendix C

SessionListActivity:

```
public class SessionListActivity extends
AppCompatActivity {
    //Presenter
    private SessionListPresenter presenter;
    private DrawerLayout drawerLayout;
    private ViewPager viewPager;
    private FragmentAdapter fragmentAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_session_list);


        //Activate the Toolbar
        Toolbar toolbar = (Toolbar)
findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        final ActionBar ab = getSupportActionBar();
        ab.setDisplayShowTitleEnabled(false);

        //Activate menu action button and set the icon

ab.setHomeAsUpIndicator(R.drawable.ic_menu_white_24dp);
        ab.setDisplayHomeAsUpEnabled(true);
        //setup activity description
        final TextView activityDesc = (TextView)
findViewById(R.id.toolbar_subtitle);
        activityDesc.setText("Sessions");
        activityDesc.setTypeface(null, Typeface.BOLD);

        //Drawerlayout
        drawerLayout = (DrawerLayout)
findViewById(R.id.drawer_layout);

        viewPager = (ViewPager)
findViewById(R.id.viewpager);
```

```
        //init variables
        presenter = new SessionListPresenter(this);

        if (viewPager != null) {
            setupViewPager(viewPager);
        }

        TabLayout tabLayout = (TabLayout)
findViewById(R.id.list_tabs);
        tabLayout.setupWithViewPager(viewPager);

    }

    public SessionListFragment getFragment(String
fragmentName) {
        return fragmentAdapter.getItem(0);
    }

    private void setupViewPager(ViewPager viewPager) {
        fragmentAdapter = new
FragmentAdapter(getSupportFragmentManager());

fragmentAdapter.addFragment(SessionListFragment.newInsta
nce(0, presenter), "All");

fragmentAdapter.addFragment(SessionListFragment.newInsta
nce(1, presenter), "Public");

fragmentAdapter.addFragment(SessionListFragment.newInsta
nce(2, presenter), "Work");

fragmentAdapter.addFragment(SessionListFragment.newInsta
nce(3, presenter), "Free");
        viewPager.setAdapter(fragmentAdapter);
    }

    //menu items
    @Override
    public boolean onOptionsItemSelected(MenuItem item)
{
        // Handle action bar item clicks here. The
action bar will
        // automatically handle clicks on the Home/Up
button, so long
        // as you specify a parent activity in
AndroidManifest.xml.
        int id = item.getItemId();
```

```
        //noinspection SimplifiableIfStatement
        switch (item.getItemId()) {
            case android.R.id.home:

drawerLayout.openDrawer(GravityCompat.START);
                return true;
        }

        return super.onOptionsItemSelected(item);
    }

    //FragmentAdapter class to be used with viewpager
and fragments
    static class FragmentAdapter extends
FragmentPagerAdapter {
        private final List<SessionListFragment>
fragments = new ArrayList<>();
        private final List<String> fragmentTitles = new
ArrayList<>();

        public FragmentAdapter(FragmentManager
fragmentManager) {
            super(fragmentManager);
        }

        public void addFragment(SessionListFragment
fragment, String title) {
            fragments.add(fragment);
            fragmentTitles.add(title);
        }

        @Override
        public SessionListFragment getItem(int position)
{
            return fragments.get(position);
        }

        @Override
        public int getCount() {
            return fragments.size();
        }

        @Override
        public CharSequence getPageTitle(int position) {
            return fragmentTitles.get(position);
```

```
        }
    }
```

SessionListFragment:

```
public class SessionListFragment extends Fragment {

    private RecyclerViewAdapter recyclerViewAdapter;
    private SessionListPresenter presenter;

    public static SessionListFragment newInstance(int
index, SessionListPresenter presenter) {

        Bundle args = new Bundle();
        SessionListFragment fragment = new
SessionListFragment();
        args.putInt("listId", index);
        fragment.setArguments(args);
        fragment.setPresenter(presenter);
        return fragment;
    }

    @Override
    public View onCreateView(LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
        onCreate(savedInstanceState);
        recyclerViewAdapter = new
RecyclerViewAdapter(getActivity(),
getArguments().getInt("listId"), presenter);
        RecyclerView rv = (RecyclerView)
inflater.inflate(
                R.layout.session_list_fragment_layout,
container, false);
        setupRecyclerView(rv);
        return rv;
    }

    public void setPresenter(SessionListPresenter
presenter) {
        this.presenter = presenter;
    }

    private void setupRecyclerView(RecyclerView
recyclerView) {
```

```
        recyclerView.setLayoutManager(new
LinearLayoutManager(recyclerView.getContext()));
        recyclerView.setAdapter(recyclerViewAdapter);
    }

    public static class RecyclerViewAdapter
            extends
RecyclerView.Adapter<RecyclerViewAdapter.ViewHolder> {

        private final TypedValue typedValue = new
TypedValue();
        private int listId;
        private SessionListPresenter presenter;

        public static class ViewHolder extends
RecyclerView.ViewHolder {
            public String boundString;

            private final View view;
            private final TextView sessionName;
            private final TextView sessionDuration;
            private final TextView userDevice;
            private final TextView userState;

            public ViewHolder(View view) {
                super(view);
                this.view = view;
                sessionName = (TextView)
view.findViewById(R.id.session_name);
                sessionDuration = (TextView)
view.findViewById(R.id.session_duration);
                userDevice = (TextView)
view.findViewById(R.id.user_device);
                userState = (TextView)
view.findViewById((R.id.user_state));
            }

            @Override
            public String toString() {
                return super.toString() + " '" +
sessionName.getText();
            }
        }

        public RecyclerViewAdapter(Context context, int
listId, SessionListPresenter presenter) {
```

```
context.getTheme().resolveAttribute(R.attr.selectableIte
mBackground, typedValue, true);
            this.listId = listId;
            this.presenter = presenter;
        }

        @Override
        public ViewHolder onCreateViewHolder(ViewGroup
parent, int viewType) {
            View view =
LayoutInflater.from(parent.getContext())
                    .inflate(R.layout.session_list_item,
parent, false);

            return new ViewHolder(view);
        }

        @Override
        public void onBindViewHolder(final ViewHolder
holder, final int position) {
            holder.boundString =
presenter.fetchSessionName(listId, position);

holder.sessionName.setText(presenter.fetchSessionName(li
stId, position));

holder.sessionDuration.setText(presenter.fetchSessionDur
ation(listId, position));


holder.userDevice.setText(presenter.fetchUserDevice(list
Id, position));

holder.userState.setText(presenter.fetchUserState(listId
,position));
            holder.view.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View v) {

                    Intent intent = new
Intent(v.getContext(), SessionInfoActivity.class);
                    intent.putExtra("session",
presenter.fetchSessionName(listId, position));//put
session name

v.getContext().startActivity(intent);
```

```
                }
            });
        }

        @Override
        public int getItemCount() {
            if (presenter != null)
                return presenter.fetchItemCount(listId);
            else
                return 0;
        }
    }
}
```

SessionListPresenter:

```
public class SessionListPresenter implements Observer {

    SessionListModel model; //model
    SessionListActivity view;

    public SessionListPresenter(Context context) {

        view = (SessionListActivity) context;
        model = new SessionListModel(context, this);
        //viewUpdate();
    }

    @Override
    public void update(Observable observable, Object
data) {
        int i = 0;
        //model = (SessionListModel) observable;
        viewUpdate();

    }

    public void onUpdatedModel() {

    }

    public ArrayList<Session> getSessions(int index) {
        return model.getListToView().get(index);
```

```
    }

    public String fetchSessionName(int listId, int
index) {
        return
model.getListToView().get(listId).get(index).getSessionN
ame();
    }

    public String fetchSessionDuration(int listId, int
index) {
        return
model.getListToView().get(listId).get(index).getSessionD
urationTime();
    }

    public String fetchSessionInfo(int listId, int
index) {
        return
model.getListToView().get(listId).get(index).getSessionI
nfo();
    }

    public String fetchUserDevice(int listId, int index)
{
        return
model.getListToView().get(listId).get(index).getUserDevi
ce();
    }
    public String fetchUserState(int listId, int index){
        String state = "";
        boolean[] userState =
model.getListToView().get(listId).get(index).getUserStat
e();

        if(userState[0])
            state += "public";
        if(userState[1])
            state += " work";
        if(userState[2])
            state += " free";
        return state;
    }

    public int fetchItemCount(int listId) {
        return model.getListToView().get(listId).size();
    }
```

```
    public void viewUpdate() {
        SessionListFragment slv =
view.getFragment("All");
        ;
    }
}
```

SessionListModel:

```
public class SessionListModel extends Observable {
    //Constants
    private final String EXERCISE_SETUP_PATH =
"config_files/exercise_test.cnf";
    private final String SESSION_SETUP_PATH =
"config_files/session_test.cnf";

    //Singleton
    private static SessionListModel instance = null;
    //Variables
    private boolean error = false;

    //Factories
    private ExerciseFactory exerciseFactory;
    private SessionFactory sessionFactory;

    //Data structures
    private HashMap<String, Exercise> exerciseMap;
//key: exercise name
    private HashMap<String, Session> sessionMap;
    private ArrayList<ArrayList<Session>> listToView;
//sorted list to show by the view. by index {all, mat,
chair, none}

    //Asset retriever
    private AssetRetriever assetRetriever;

    public SessionListModel(Context context, Observer
presenter) {
        assetRetriever = new
AssetRetriever(context.getAssets());
        exerciseFactory = new ExerciseFactory();
        sessionFactory = new SessionFactory();
```

```
        exerciseMap = new HashMap<String, Exercise>();
        sessionMap = new HashMap<String, Session>();
        listToView = new ArrayList<>();
        addObserver(presenter);
        loadExercises();
        loadSessions();
        initListToView();
        testSort();

        instance = this;//set singleton on creation
        //on error notify user
    }

    public static SessionListModel getInstance()
{//makes it reachable throughout the application,
eventually change the default const.!!!
        return instance;
    }

    public ArrayList<ArrayList<Session>> getListToView()
{
        return listToView;
    }

    public Session retrieveSessionByName(String name) {
        return sessionMap.get(name);
    }

    private void loadExercises() {
        BufferedReader br;
        if ((br =
assetRetriever.loadReaderFromAsset(EXERCISE_SETUP_PATH))
!= null) {
            if ((exerciseMap = exerciseFactory.make(br))
== null) {
                error = true;
            } else
                error = false;//eventually generate
exception
        }
    }

    private void loadSessions() {
        BufferedReader br;
        if ((br =
assetRetriever.loadReaderFromAsset(SESSION_SETUP_PATH))
!= null) {
```

```
            if ((sessionMap = sessionFactory.make(br,
exerciseMap)) == null) {
                error = true;
            } else
                error = false;//eventually generate
exception
        }
    }

    private void initListToView() {
        for (int i = 0; i < 4; i++)
            listToView.add(new ArrayList<Session>());
    }

    private void testSort() {//temporary method for
testing, sort {all,mat,chair,none}

        for (Session s : sessionMap.values()) {
            listToView.get(0).add(s);
            for (int i = 0; i < 3; i++) {
                if (s.getUserState()[i])
                    listToView.get(i + 1).add(s);
            }
        }
    }
}
```

ExerciseFactory:

```
public class ExerciseFactory {
    private final String EXERCISE_ROOT_PATH =
"exercises/";

    public HashMap<String, Exercise> make(BufferedReader
br) {
        //exercise building blocks
        String name = "";
        int duration = 0;
        String medicalInfo = "";
        String yogaInfo = "";
        EventList textEvents = new EventList();
        EventList imageEvents = new EventList();
        String voicePath = "";
```

```
        //set flags and path
        int choice = 0;//0 name, voice path, duration, 1
medical, 2 yoga, 3 text events, 4 image events
        HashMap<String, Exercise> exMap = new
HashMap<String, Exercise>();
        String line;

        try {
            while ((line = br.readLine()) != null &&
!line.equals("#endfile")) {
                if ((!line.equals("#end"))) {
                    switch (choice) {
                        case 0://name, voice path,
duration
                            name = line;
                            voicePath =
EXERCISE_ROOT_PATH + "voice/" + br.readLine();
                            duration =
Integer.parseInt(br.readLine());
                            break;
                        case 1://medical
                            medicalInfo += line + " ";
                            break;
                        case 2://yoga
                            yogaInfo += line + " ";
                            break;
                        case 3://text_guide
                            if
(line.startsWith("#loop"))
                                textEvents =
loopEvents(textEvents,
Integer.parseInt(line.substring(6)), duration);
                            else

textEvents.add(lineToEvent(line));
                            break;
                        case 4://image_guide
                            if
(line.startsWith("#loop"))
                                imageEvents =
loopEvents(imageEvents,
Integer.parseInt(line.substring(6)), duration);
                            else

imageEvents.add(lineToEvent(line));
                            break;
                    }
```

```
                }
                if (line.equals("#end") ||
(line.startsWith("#loop"))) {
                    if (choice == 4) {
                        choice = 0;
                        exMap.put(name, new
Exercise(textEvents, imageEvents, voicePath, duration,
name, medicalInfo, yogaInfo));
                        textEvents = new EventList();
                        imageEvents = new EventList();
                        medicalInfo = "";
                        yogaInfo = "";
                    } else
                        choice++;
                }
            }
            br.close();
        } catch (IOException e) {
            return null;
        }

        return exMap;
    }

    private Event<String> lineToEvent(String line) {
        return new Event<String>(line.substring(0,
line.indexOf("#")),
Integer.parseInt(line.substring(line.indexOf("#") + 1,
line.length())));
    }

    private EventList loopEvents(EventList listToLoop,
int loopDelay, int duration) {
        int startTime;
        int newStartTime;
        EventList loopList = new EventList();
        loopList.addAll(listToLoop);

        while (true) {
            startTime = loopList.get(loopList.size() -
1).getWhen() + loopDelay;
            for (Event e : listToLoop) {
                newStartTime = e.getWhen() +
startTime;//new zeroth second
                if (newStartTime < duration)
                    loopList.add(new Event(e.getEvent(),
newStartTime));
```

```
                else
                    return loopList;
            }
        }
    }
}
```

SessionFactory:

```
public class SessionFactory {

    private final String MUSIC_PATH = "music/";


    public SessionFactory() {
    }

    public HashMap<String, Session> make(BufferedReader
br, HashMap<String, Exercise> exerciseMap) {
        String name = "";
        String info = "";
        boolean userState[] = new boolean[3]; //
userState {PUBLIC, WORK, FREE}
        String userDevice = "";
        EventList musicEvents = new EventList();
        ArrayList<Exercise> exerciseArrayList = new
ArrayList<>();
        //set flags and path
        int choice = 0;
        HashMap<String, Session> sessionMap = new
HashMap<>();
        String line;

        try {
            while ((line = br.readLine()) != null &&
!line.equals("#endfile")) {
                if (!line.trim().equals("#end") &&
!line.trim().equals("#endsession")) {
                    switch (choice) {
                        case 0://name, userState
(PUBLIC, WORK, FREE)
                            name = line.trim(); //in
case of whitespace in cnf
```

```
                            for (int i = 0; i < 3; i++)
{
                                if
(br.readLine().trim().equals("1"))
                                    userState[i] = true;
                                else
                                    userState[i] =
false;
                            }
                            break;
                        case 1://userDevice
                            userDevice = line.trim();
                            break;
                        case 2://info
                            info += line.trim() + " ";
                            break;
                        case 3://exercise sequence

exerciseArrayList.add(exerciseMap.get(line.trim()));
                            break;
                        case 4://music
                            musicEvents.add(new
Event<String>(MUSIC_PATH + line.substring(0,
line.indexOf("#")),
Integer.parseInt(line.substring(line.indexOf("#") +
1).trim())));
                            break;
                    }
                }

                if (line.trim().equals("#end")) {
                    if (choice > 4) {
                        choice = 0;

                    } else
                        choice++;
                }
                if (line.trim().equals("#endsession")) {
                    sessionMap.put(name, new
Session(name, info, userState, userDevice, musicEvents,
exerciseArrayList));
                    exerciseArrayList = new
ArrayList<>();//reset data
                    userState = new boolean[3];
                    musicEvents = new EventList();
                    choice = 0;
                    info = "";
```

```
                }
            }

            br.close();
        } catch (IOException e) {
            return null;
        }
        if (line.trim().equals("#endfile")) {
            return sessionMap;
        } else
            return null;
    }
}
```

Presentation av övningssekvenser

SessionInfoActivity:

```
public class SessionInfoActivity extends
AppCompatActivity {

    private ViewPager exercisePager;
    private FragmentAdapter fragmentAdapter;
    private SessionInfoPresenter presenter;
    private Button musicButton;
    private Button voiceButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_session_info);
        presenter = new
SessionInfoPresenter(getApplicationContext(),
getIntent().getStringExtra("session"));
        setupComponents();
        setupButtons();
    }

    private void setupComponents() {
        //ActionBar
        Toolbar toolbar = (Toolbar)
findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        final ActionBar ab = getSupportActionBar();
        ab.setDisplayShowTitleEnabled(false);
```

```
        //Activate menu action button and set the icon

ab.setHomeAsUpIndicator(R.drawable.ic_arrow_back_white_2
4dp);
        ab.setDisplayHomeAsUpEnabled(true);
        //subtitle
        final TextView title = (TextView)
findViewById(R.id.toolbar_title);

title.setText(getIntent().getStringExtra("session"));
        final TextView subTitle = (TextView)
findViewById(R.id.toolbar_subtitle);
        subTitle.setText("Information");

        //view pager
        exercisePager = (ViewPager)
findViewById(R.id.exercise_pager);
        if (exercisePager != null) setupExercisePager();
        //Setup tabs
        TabLayout tabLayout = (TabLayout)
findViewById(R.id.exercise_tabs);
        tabLayout.setupWithViewPager(exercisePager);

        //Buttons
        musicButton = (Button)
findViewById(R.id.music_btn);
        voiceButton = (Button)
findViewById(R.id.voice_btn);
    }

    private void setupExercisePager() {
        if (presenter == null) return;

        fragmentAdapter = new
FragmentAdapter(getSupportFragmentManager());
        int sessionSize =
presenter.retrieveSessionSize();

        for (int i = 0; i < sessionSize; i++) {
            Bundle info =
presenter.retrieveExerciseInfo(i);
            ExerciseInfoFragment fragment =
ExerciseInfoFragment.newInstance(i, info);
            fragmentAdapter.addFragment(fragment,
info.getString(SessionInfoPresenter.EXERCISE_NAME));
        }
```

```
        exercisePager.setAdapter(fragmentAdapter);
    }

    private void setupButtons() {
        voiceButton.setOnClickListener(new
View.OnClickListener() {//voice button
            @Override
            public void onClick(View v) {
                Intent intent = new
Intent(v.getContext(), SessionActivity.class);
                intent.putExtra("session",
getIntent().getStringExtra("session"));//put session
name
                intent.putExtra("music", false);//music
flag
                v.getContext().startActivity(intent);
            }
        });
        musicButton.setOnClickListener(new
View.OnClickListener() {//music button
            @Override
            public void onClick(View v) {
                Intent intent = new
Intent(v.getContext(), SessionActivity.class);
                intent.putExtra("session",
getIntent().getStringExtra("session"));//put session
name
                intent.putExtra("music", true);
                v.getContext().startActivity(intent);
            }
        });
    }

    //FragmentAdapter class to be used with viewpager
and fragments
    static class FragmentAdapter extends
FragmentPagerAdapter {
        private final List<ExerciseInfoFragment>
fragments = new ArrayList<>();
        private final List<String> fragmentTitles = new
ArrayList<>();

        public FragmentAdapter(FragmentManager
fragmentManager) {
            super(fragmentManager);
        }
```

```
        public void addFragment(ExerciseInfoFragment
fragment, String title) {
            fragments.add(fragment);
            fragmentTitles.add(title);
        }

        @Override
        public ExerciseInfoFragment getItem(int
position) {
            return fragments.get(position);
        }

        @Override
        public int getCount() {
            return fragments.size();
        }

        @Override
        public CharSequence getPageTitle(int position) {
            return fragmentTitles.get(position);
        }
    }
}
```

ExerciseInfoFragment:

```
public class ExerciseInfoFragment extends Fragment {
    private TextView showExerciseName;
    private TextView showMedInfo;
    private TextView showYogaInfo;

    public static ExerciseInfoFragment newInstance(int
index, Bundle state) {
        Bundle args = new Bundle();
        ExerciseInfoFragment fragment = new
ExerciseInfoFragment();
        args.putInt("listId", index);//itemId
        args.putBundle("content", state);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public View onCreateView(LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
```

```
        onCreate(savedInstanceState);

        View fragmentView = (View) inflater.inflate(
                R.layout.exercise_info_fragment_layout,
container, false);

        setupFragment(fragmentView);

refreshFragment(getArguments().getBundle("content"));
        return fragmentView;
    }

    private void setupFragment(View view) {
        showExerciseName = (TextView)
view.findViewById(R.id.exercise_fragment_exercise_name);
        showMedInfo = (TextView)
view.findViewById(R.id.med_info);
        showYogaInfo = (TextView)
view.findViewById(R.id.yoga_info);
    }

    public void refreshFragment(Bundle state) {

showExerciseName.setText(state.getString(SessionInfoPres
enter.EXERCISE_NAME));

showMedInfo.setText(state.getString(SessionInfoPresenter
.EXERCISE_MED_INFO));

showYogaInfo.setText(state.getString(SessionInfoPresente
r.EXERCISE_YOGA_INFO));
    }

}
```

SessionInfoPresenter:

```
public class SessionInfoPresenter {

    private SessionListModel sessionListModel;
    private String sessionName;
    //Bundle keys
    public static final String EXERCISE_NAME = "name";
```

```
    public static final String EXERCISE_DURATION =
"duration";
    public static final String EXERCISE_YOGA_INFO =
"yoga_info";
    public static final String EXERCISE_MED_INFO =
"med_info";

    public SessionInfoPresenter(Context context, String
sessionName) {
        sessionListModel =
SessionListModel.getInstance();
        this.sessionName = sessionName;
    }

    public Bundle retrieveExerciseInfo(int
exerciseIndex) {
        if (sessionListModel == null) return null;

        Exercise exercise =
sessionListModel.retrieveSessionByName(sessionName).get(
exerciseIndex);
        Bundle exerciseInfoBundle = new Bundle();

        if (exercise == null) return null;

        exerciseInfoBundle.putString(EXERCISE_NAME,
exercise.getName());
        exerciseInfoBundle.putString(EXERCISE_DURATION,
TimeHelper.secondsToTimeString(exercise.getDuration()));
        exerciseInfoBundle.putString(EXERCISE_YOGA_INFO,
exercise.getYogaInfo());
        exerciseInfoBundle.putString(EXERCISE_MED_INFO,
exercise.getMedicalInfo());

        return exerciseInfoBundle;
    }

    public int retrieveSessionSize() {
        if (sessionListModel != null)
            return
sessionListModel.retrieveSessionByName(sessionName).size
();
        return 0;//something went wrong
    }
}
```

SessionActivity:

```
public class SessionActivity extends AppCompatActivity {
    //Weak references for the Handlers in the Presenter
    private TextViewHandler showTextHandler;
    private TextViewHandler showTimeHandler;
    private TextViewHandler showExerciseHandler;
    private ImageViewHandler showImageHandler;
    private FAButtonHandler play_pauseButtonHandler;
    private ProgressHandler progressHandler;

    WeakReference<TextView> wrShowText;
    WeakReference<TextView> wrShowTime;
    WeakReference<ImageView> wrShowImage;
    WeakReference<TextView> wrShowExercise;
    //WeakReference<ToggleButton> wrPlayPauseButton;
    WeakReference<FloatingActionButton> wrPlayPauseFAB;
    WeakReference<ToggleButton> wrAudioToggleButton;
    WeakReference<ProgressBar> wrShowProgress;

    //Views
    private TextView showText;
    private TextView showTime;
    private TextView showExercise;
    private TextView showSession;
    private Button stopButton;
    private ToggleButton muteAudioToggleButton;
    //private ToggleButton pauseToggleButton;
    private FloatingActionButton playFAB;
    private ProgressBar showProgress;
    //private ToggleButton wakeLockToggleButton;
    private ImageView showImage;

    private SessionPresenter presenter;
    private PowerManager powerManager;
    PowerManager.WakeLock wakeLock;

    @Override
    protected void onCreate(Bundle savedInstanceState)
{//SET ALL VIEW VAR HERE inc duration, setup handlers
here
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_session);

        //ActionBar
        Toolbar toolbar = (Toolbar)
findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        final ActionBar ab = getSupportActionBar();
```

```
        ab.setDisplayShowTitleEnabled(false);

        //Activate menu action button and set the icon

ab.setHomeAsUpIndicator(R.drawable.ic_arrow_back_white_2
4dp);
        ab.setDisplayHomeAsUpEnabled(true);
        //powerManager = (PowerManager)
getSystemService(POWER_SERVICE);


        //Initialize components
        initViewComponents();
        initHandlers();

        //Component functions
        setupComponents();
        //Presenter
        presenter = new SessionPresenter(this,
getIntent().getStringExtra("session"), showTextHandler,
showImageHandler, play_pauseButtonHandler,
showExerciseHandler, progressHandler);

    }


    //Pause on pushed to background
    @Override
    protected void onPause() {
        super.onPause();
        presenter.onPushedToBackground();
    }

    @Override
    protected void onResume() {
        super.onResume();
        presenter.onPulledToForeground();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
    }

    @Override
    public void onBackPressed() {
        presenter.onStop();
```

```
    }

    private void initViewComponents() {
        //Input
        playFAB = (FloatingActionButton)
findViewById(R.id.play_FAB);

        //Output
        showText = (TextView)
findViewById(R.id.showText);
        showTime = (TextView)
findViewById(R.id.showTime);
        showImage = (ImageView)
findViewById(R.id.showVisual);
        showExercise = (TextView)
findViewById(R.id.toolbar_subtitle);
        showSession = (TextView)
findViewById(R.id.toolbar_title);
        showProgress = (ProgressBar)
findViewById(R.id.showProgress);
    }

    //Help methods for the constructor
    private void initHandlers() {
        showTextHandler = new TextViewHandler(new
WeakReference(showText));
        showExerciseHandler = new TextViewHandler(new
WeakReference(showExercise));
        showImageHandler = new
ImageViewHandler(getApplicationContext(), new
WeakReference(showImage));
        play_pauseButtonHandler = new
FAButtonHandler(getApplicationContext(), new
WeakReference(playFAB));

        if (SessionListModel.getInstance() !=
null)//direct model access, should be through presenter
            progressHandler = new ProgressHandler(new
WeakReference(showTime), new
WeakReference(showProgress),
SessionListModel.getInstance().

retrieveSessionByName(getIntent().getStringExtra("sessio
n")).getSessionDurationInSeconds());
    }

    private void setupComponents() {
```

```
        //Views

showSession.setText(getIntent().getStringExtra("session"
));

        if (SessionListModel.getInstance() != null)
{//no knowledge of session, done onCreate to avoid the
menu glitch (wrap_content)
            int duration =
SessionListModel.getInstance().retrieveSessionByName(get
Intent().getStringExtra("session")).getSessionDurationIn
Seconds();
            showProgress.setMax(duration);//set total
duration

showExercise.setText(SessionListModel.getInstance().retr
ieveSessionByName(getIntent().getStringExtra("session"))
.get(0).getName());

showTime.setText(TimeHelper.remainingTimeString(duration
, 0));
        }

        playFAB.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if
(playFAB.getDrawable().getConstantState().equals(getDraw
able(R.drawable.ic_pause_white_36dp).getConstantState())
) {//checked = paused
                    presenter.onPause();

playFAB.setImageDrawable(getDrawable(R.drawable.ic_play_
arrow_white_36dp));
                } else {
                    presenter.onResume();

playFAB.setImageDrawable(getDrawable(R.drawable.ic_pause
_white_36dp));
                }
            }
        });
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
```

```
{
        // Handle action bar item clicks here. The
action bar will
        // automatically handle clicks on the Home/Up
button, so long
        // as you specify a parent activity in
AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        switch (item.getItemId()) {
            case android.R.id.home:
                onBackPressed();
                return true;
        }

        return super.onOptionsItemSelected(item);
    }


}
```

SessionPresenter:

```
public class SessionPresenter {

    //Thread handler, for thread/UI communication
    private TextViewHandler showTextHandler;
    private TextViewHandler showTimeHandler;
    private TextViewHandler showExerciseHandler;
    private ImageViewHandler showImageHandler;
    private FAButtonHandler play_pauseButtonHandler;
    private ProgressHandler progressHandler;

    private SessionPlaybackService
sessionPlaybackService = null;
    private SessionActivity activity;
    private String sessionName;
    //service bound flag, recommended by developers
    private boolean isBound = false;

    //Instance
    private static SessionPresenter instance = null;
```

```
    //singleton
    public SessionPresenter(SessionActivity activity,
String sessionName, TextViewHandler showTextHandler,
ImageViewHandler showImageHandler,
                            FAButtonHandler
play_pauseButtonHandler, TextViewHandler
showExerciseHandler,ProgressHandler progressHandler ) {
        this.showTextHandler = showTextHandler;
        this.showImageHandler = showImageHandler;
        this.play_pauseButtonHandler
=play_pauseButtonHandler;
        this.showExerciseHandler = showExerciseHandler;
        this.progressHandler = progressHandler;
        this.sessionName = sessionName;
        this.activity = activity;
        instance = this;

        bindService(activity);
    }

    public void bindService(SessionActivity activity) {
//Create and bind to the service

        Intent intent = new Intent(activity,
SessionPlaybackService.class);
        activity.bindService(intent, serviceConnection,
Context.BIND_AUTO_CREATE);

    }

    public void setupService() {
        sessionPlaybackService.setupService(activity,
sessionName);//complete the initialization of the
service
    }

    public static SessionPresenter getInstance()
{//eventually throw exception
        return instance;
    }

    public void stopService() {
        if (sessionPlaybackService != null && isBound) {
            isBound = false;
            sessionPlaybackService = null;
            activity.unbindService(serviceConnection);
```

```
        }
        activity.finish();
    }

    public void onPushedToBackground() {
        if (sessionPlaybackService != null && isBound)
            sessionPlaybackService.runInBackground();

    }

    public void onPulledToForeground() {
        if (sessionPlaybackService != null && isBound) {
            sessionPlaybackService.runInForeground();
        }
    }

    //input methods
    public void onAudioMute() {
        sessionPlaybackService.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_MUTE),
0, 0);
    }

    public void onVoiceMute() {
        //sessionService.toggleMuteVoice();
    }

    public void onStop() {
        if (sessionPlaybackService != null && isBound) {
            sessionPlaybackService.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_STOP),
0, 0);
        }
    }

    public void onPause() {
        if (sessionPlaybackService != null && isBound)
            sessionPlaybackService.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_PAUSE),
0, 0);
    }

    public void onResume() {
        if (sessionPlaybackService != null && isBound)
            sessionPlaybackService.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_PLAY),
0, 0);
```

```
    }

    //output methods
    public void updateVisualGuide(String path) {
        Message msg = showImageHandler.obtainMessage();
        Bundle bundle = new Bundle();

        bundle.putString("output", path);
        msg.setData(bundle);
        showImageHandler.sendMessage(msg);
    }

    public void updateTextGuide(String guide) {
        Message msg = showTextHandler.obtainMessage();
        Bundle bundle = new Bundle();

        bundle.putString("output", guide);
        msg.setData(bundle);
        showTextHandler.sendMessage(msg);
    }

    public void updateExerciseName(String exerciseName)
{
        Message msg =
showExerciseHandler.obtainMessage();
        Bundle bundle = new Bundle();

        bundle.putString("output", exerciseName);
        msg.setData(bundle);
        showExerciseHandler.sendMessage(msg);
    }

    public void updateProgress(int timer) {
        Message msg = progressHandler.obtainMessage();
        Bundle bundle = new Bundle();

        bundle.putInt("output", timer);
        msg.setData(bundle);
        progressHandler.sendMessage(msg);
    }

    public void updateButtonState(String state) {
        Message msg =
play_pauseButtonHandler.obtainMessage();
        Bundle bundle = new Bundle();
        if
(state.equals(SessionPlaybackManager.PLAYING))
```

```
            bundle.putString(ToggleButtonHandler.KEY,
"checked");
        else
            bundle.putString(ToggleButtonHandler.KEY,
"unchecked");
        msg.setData(bundle);
        play_pauseButtonHandler.sendMessage(msg);
    }

    //connection to the service, retrieves the service
object
    private ServiceConnection serviceConnection = new
ServiceConnection() {

        public void onServiceConnected(ComponentName
className,
                                       IBinder binder) {

SessionPlaybackService.SessionNotificationServiceBinder
b =
(SessionPlaybackService.SessionNotificationServiceBinder
) binder;
            sessionPlaybackService = b.getService();
            isBound = true;
            setupService();
        }

        public void onServiceDisconnected(ComponentName
className) {

            sessionPlaybackService = null;
        }
    };
}
```

SessionPlaybackService:

```
public class SessionPlaybackService extends Service
implements AudioManager.OnAudioFocusChangeListener {
    //Actions, refactor to int
    public static final String ACTION_PLAY =
"action_play";
    public static final String ACTION_PAUSE =
```

```
"action_pause";
    public static final String ACTION_STOP =
"action_stop";
    public static final String ACTION_MUTE =
"action_mute";
    public static final String ACTION_REFRESH_TIMER =
"action_refresh_timer";
    public static final String ACTION_REFRESH_TEXT =
"action_refresh_text";
    public static final String ACTION_REFRESH_VISUAL =
"action_refresh_visual";
    public static final String ACTION_REFRESH_EXERCISE =
"action_refresh_exercise";
    public static final String
ACTION_REFRESH_BUTTON_STATE =
"action_refresh_button_state";
    //Other
    private SessionNotificationServiceBinder binder;
    private SessionPlaybackManager
sessionPlaybackManager;
    private SessionNotificationManager
sessionNotificationManager;

    //Flags
    private boolean isInBackground = false; //true when
pushed to background, switches to notification
    private SessionPresenter sessionPresenter;
    private boolean audioFocusLost = false; //playback
interrupted by device

    //default constructor for auto-create
    public SessionPlaybackService() {

    }

    public void setupService(SessionActivity activity,
String sessionName) {
        sessionPresenter =
SessionPresenter.getInstance();
        sessionPlaybackManager = new
SessionPlaybackManager(this, sessionName,
activity.getIntent().getBooleanExtra("music",false));
        isInBackground = false;
        sessionNotificationManager = new
SessionNotificationManager(this, sessionName);

        AudioManager audioManager = (AudioManager)
```

```
getSystemService(AUDIO_SERVICE);
        audioManager.requestAudioFocus(this,
AudioManager.STREAM_MUSIC,
                AudioManager.AUDIOFOCUS_GAIN);
    }

    private void handleActionIntent(Intent intent) {

        if (intent == null || sessionPresenter == null)
            return;

        String action = intent.getAction();
        if (!isInBackground && sessionPlaybackManager !=
null) {
            switch (action) {
                //output
                case ACTION_REFRESH_TIMER:

sessionPresenter.updateProgress(intent.getIntExtra(Sessi
onPlaybackManager.TIMER, 0));
                    break;
                case ACTION_REFRESH_TEXT:

sessionPresenter.updateTextGuide(intent.getStringExtra(S
essionPlaybackManager.TEXT));
                    break;
                case ACTION_REFRESH_EXERCISE:

sessionPresenter.updateExerciseName(intent.getStringExtr
a(SessionPlaybackManager.EXERCISE_NAME));
                    break;
                case ACTION_REFRESH_VISUAL:

sessionPresenter.updateVisualGuide(intent.getStringExtra
(SessionPlaybackManager.VISUAL));
                    break;
                case ACTION_REFRESH_BUTTON_STATE:

sessionPresenter.updateButtonState(intent.getStringExtra
(SessionPlaybackManager.PLAYBACK));
                    break;
                //input
                case ACTION_PLAY://change button, start
playback
                    sessionPlaybackManager.resume();
                    break;
                case ACTION_PAUSE:
```

```
                    sessionPlaybackManager.pause();
                    break;
                case ACTION_STOP:
                    sessionPlaybackManager.stop();
                    sessionPresenter.stopService();
                    break;
                case ACTION_MUTE:

sessionPlaybackManager.toggleAudioMute();
                    break;
            }
        } else if (sessionNotificationManager != null) {
            switch (action) {
                case ACTION_REFRESH_TIMER:

sessionNotificationManager.updateNotification(sessionPla
ybackManager.retrieveExerciseName(),
sessionPlaybackManager.retrievePlaybackState(),

TimeHelper.secondsToTimeString(intent.getIntExtra(Sessio
nPlaybackManager.TIMER, 0)));
                    break;
                case ACTION_PLAY://change button, start
playback
                    sessionPlaybackManager.resume();

sessionNotificationManager.updateNotification(sessionPla
ybackManager.retrieveExerciseName(),
sessionPlaybackManager.retrievePlaybackState(),

TimeHelper.secondsToTimeString(sessionPlaybackManager.re
trieveTimerState()));
                    break;
                case ACTION_PAUSE:
                    sessionPlaybackManager.pause();

sessionNotificationManager.updateNotification(sessionPla
ybackManager.retrieveExerciseName(),
sessionPlaybackManager.retrievePlaybackState(),

TimeHelper.secondsToTimeString(sessionPlaybackManager.re
trieveTimerState()));
                    break;
                case ACTION_STOP:

sessionNotificationManager.updateNotification(sessionPla
ybackManager.retrieveExerciseName(),
```

```
sessionPlaybackManager.retrievePlaybackState(),

TimeHelper.secondsToTimeString(sessionPlaybackManager.re
trieveTimerState()));
                    break;
            }
        }
    }

    public void runInBackground() {
        isInBackground = true;
        if (sessionPlaybackManager == null ||
sessionPlaybackManager.retrievePlaybackState().equals(Se
ssionPlaybackManager.STOPPED) ||
sessionNotificationManager == null)
            return;

sessionNotificationManager.updateNotification(sessionPla
ybackManager.retrieveExerciseName(),
sessionPlaybackManager.retrievePlaybackState(),

TimeHelper.secondsToTimeString(sessionPlaybackManager.re
trieveTimerState()));
    }

    public void runInForeground() {
        isInBackground = false;
        sessionNotificationManager.stopNotification();
        sessionPlaybackManager.refreshState();
    }

    //Back communication application, method keeps
service running while pushed to background
    @Override
    public int onStartCommand(Intent intent, int flags,
int startId) {
        handleActionIntent(intent);
        return START_NOT_STICKY;
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {//used for
communication with the service
        isInBackground = false;
        return new SessionNotificationServiceBinder();
    }
```

```
    @Override
    public void onRebind(Intent intent) {
        isInBackground = false;
        sessionPresenter =
SessionPresenter.getInstance();
        sessionPlaybackManager.refreshState();
    }

    @Override
    public boolean onUnbind(Intent intent) {
        isInBackground = false;

        return false;//super.onUnbind(intent);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
    }

    @Override
    public void onAudioFocusChange(int focusChange) {
        if (sessionPlaybackManager == null) return;
        switch (focusChange) {//pause playback on every
loss
            case AudioManager.AUDIOFOCUS_GAIN:
                // resume playback
                if (audioFocusLost)
                    sessionPlaybackManager.resume();
                audioFocusLost = false;
                break;

            case AudioManager.AUDIOFOCUS_LOSS:
                // Lost focus for an unbounded amount of
time: stop playback and release media player
                if
(sessionPlaybackManager.retrievePlaybackState() ==
SessionPlaybackManager.PLAYING) {
                    audioFocusLost = true;
                    sessionPlaybackManager.pause();
                }
                break;

            case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
                // Lost focus for a short time, but we
have to stop
```

```
                // playback. We don't release the media
player because playback
                // is likely to resume
                //if (mMediaPlayer.isPlaying())
mMediaPlayer.pause();
                if
(sessionPlaybackManager.retrievePlaybackState() ==
SessionPlaybackManager.PLAYING) {
                    audioFocusLost = true;
                    sessionPlaybackManager.pause();
                }
                audioFocusLost = false;
                break;

            case
AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
                // Lost focus for a short time, but it's
ok to keep playing
                break;
        }

    }

    //Binder class
    public class SessionNotificationServiceBinder
extends Binder {
        public SessionPlaybackService getService() {
            return SessionPlaybackService.this;
        }
    }
}
```

SessionPlaybackManager:

```
public class SessionPlaybackManager extends Observable {

    //var
    public SessionMonitor monitor;
    private Session session;
    private AssetRetriever assetRetriever;
    //private Context context;
    SessionPlaybackService service;
    //state keys
    public static final String EXERCISE_NAME =
"exercise_name";
```

```
    public static final String TIMER = "timer";
    public static final String PLAYBACK = "playback";
    public static final String TEXT = "text";
    public static final String VISUAL = "visual";
    //playback values
    public static final String PLAYING = "playing";
    public static final String PAUSED = "paused";
    public static final String STOPPED = "stopped";

    //Threads
    private VisualThread visualThread;//image guide
    private TextThread textThread;//text guide
    private AudioThread audioThread; //background music
    private TimeThread timeThread;

    //Instance
    private static SessionPlaybackManager instance =
null;

    public SessionPlaybackManager(SessionPlaybackService
service, String sessionName, boolean isMusic) {

        session =
SessionListModel.getInstance().retrieveSessionByName(ses
sionName);
        this.service = service;
        monitor = new SessionMonitor(session);//last
index
        //prepareAudioSession(session);
        setupThreads(service.getApplicationContext(),
isMusic);

        instance = this;

        startThreads();
        //refreshState();
    }

    //getters
    public static SessionPlaybackManager getInstance() {
        return instance;
    }

    public String retrieveExerciseName() {
        return
session.get(monitor.getCurrentExerciseIndex()).getName()
;
```

```
    }

    public String retrievePlaybackState() {
        if (monitor.isStopped())
            return STOPPED;
        if (monitor.isPaused())
            return PAUSED;
        return PLAYING;
    }

    private String retrieveTextGuide() {
        if (textThread != null)
            return textThread.getCurrentEvent();
        return "";
    }

    public int retrieveTimerState() {
        if (timeThread != null)
            return timeThread.getTimer();
        return 0;//return empty in hope that no one
notice ;)
    }

    public void refreshState() {
        service.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_REFRESH
_TIMER).putExtra(TIMER, retrieveTimerState()), 0,
0);//timer
        service.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_REFRESH
_EXERCISE).putExtra(EXERCISE_NAME,
retrieveExerciseName()), 0, 0);
        service.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_REFRESH
_TEXT).putExtra(TEXT, retrieveTextGuide()), 0, 0);
        service.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_REFRESH
_BUTTON_STATE).putExtra(PLAYBACK,
retrievePlaybackState()), 0, 0);
    }

    //Interface methods
    public void stop() {
        monitor.stop();
    }

    public void pause() {
```

```
        monitor.pause(true);
    }

    public void resume() {
        monitor.pause(false);
    }

    public void toggleAudioMute() {
        monitor.changeAudioState();
    }

    private void setupThreads(Context context, boolean
isMusic) {
        visualThread = new VisualThread(session,
monitor, service);
        textThread = new TextThread(monitor, session,
service);
        timeThread = new TimeThread(monitor, session,
service);

        if (isMusic)
            audioThread = new MusicThread(context,
monitor, session);
        else
            audioThread = new VoiceThread(context,
monitor, session);
    }

    private void startThreads() {
        visualThread.start();
        textThread.start();
        timeThread.start();
        audioThread.start();
    }
}
```

SessionNotificationManager:

```
public class SessionNotificationManager {

    private SessionPlaybackService service;
    private Context context;
    private final int NOTIFICATION_ID = 101;
```

```
    private NotificationManager notificationManager;
    private NotificationCompat.Builder
notificationBuilder;
    private String sessionDuration;

    private boolean isBusy = false; //not the best
solution... for synchronization

    //Actions
    private NotificationCompat.Action action;
    //Style
    NotificationCompat.MediaStyle style;
    //Constants
    private static final int PLAY_ICON =
R.drawable.ic_play_circle_outline_white_48dp;
    private static final int PAUSE_ICON =
R.drawable.ic_pause_circle_outline_white_48dp;

    public
SessionNotificationManager(SessionPlaybackService
service, String sessionName) {//context =
SessionActivity
        this.service = service;
        context = service.getApplicationContext();
        createSessionNotification(sessionName);
    }

    private void createSessionNotification(String
sessionName) {//take some input like timer...

        notificationBuilder = new
NotificationCompat.Builder(service);
        Intent intent = new
Intent(service.getApplicationContext(),
SessionPlaybackService.class);

//intent.setAction(SessionPlaybackService.ACTION STOP);
        //PendingIntent deletePendingIntent =
PendingIntent.getService(service.getApplicationContext()
, NOTIFICATION_ID, intent,
PendingIntent.FLAG UPDATE CURRENT);


        intent = new
Intent(service.getApplicationContext(),
SessionActivity.class);
        PendingIntent returnPendingIntent =
```

```
PendingIntent.getActivity(service.getApplicationContext(
), NOTIFICATION_ID, intent,
PendingIntent.FLAG_UPDATE_CURRENT);


        notificationBuilder

.setSmallIcon(R.drawable.ic_headset_white_24dp)
                .setContentTitle(sessionName)
                        //.setContentText(exerciseName)

//.setDeleteIntent(deletePendingIntent)//on swipe
                .setShowWhen(false)

//.setColor(service.getResources().getColor(R.color.cust
om_accent))//change color
                        //.setSubText(timer)
                .setContentIntent(returnPendingIntent)

.setPriority(NotificationCompat.PRIORITY_MAX);

        style = new NotificationCompat.MediaStyle();
        notificationBuilder.setStyle(style);


notificationBuilder.addAction(generateAction(PLAY_ICON,
"action_play",
SessionPlaybackService.ACTION_PLAY));//placeholder
action
        style.setShowActionsInCompactView(0);
        notificationManager = (NotificationManager)
service.getSystemService(service.NOTIFICATION_SERVICE);
    }

    //Update methods
    public void updateNotification(String exerciseName,
String playbackState, String timer) {

        if
(playbackState.equals(SessionPlaybackManager.PAUSED)) {
            notificationBuilder.mActions.clear();

notificationBuilder.addAction(generateAction(PLAY_ICON,
"action_play", SessionPlaybackService.ACTION_PLAY));
        }
        if
(playbackState.equals(SessionPlaybackManager.PLAYING)) {
```

```
            notificationBuilder.mActions.clear();

notificationBuilder.addAction(generateAction(PAUSE_ICON,
"action_pause", SessionPlaybackService.ACTION_PAUSE));
        }
        if
(playbackState.equals(SessionPlaybackManager.STOPPED)) {
            notificationBuilder.setSubText("completed");
        } else
            notificationBuilder.setSubText(timer);


notificationBuilder.setContentText(exerciseName);

        notificationManager.notify(NOTIFICATION_ID,
notificationBuilder.build());
    }

    public void stopNotification() {
        if (notificationManager != null)
            notificationManager.cancel(NOTIFICATION_ID);

    }

    private NotificationCompat.Action generateAction(int
icon, String title, String intentAction) {
        Intent intent = new
Intent(service.getApplicationContext(),
SessionPlaybackService.class);
        intent.setAction(intentAction);
        PendingIntent pendingIntent =
PendingIntent.getService(service.getApplicationContext()
, 1, intent, 0);
        return new
NotificationCompat.Action.Builder(icon, title,
pendingIntent).build();
    }
}
```

SessionMonitor:

```
public class SessionMonitor {
    private boolean isStopped;
```

```
    private boolean isPaused;
    private final int TOTAL_AMOUNT_OF_THREADS = 4;

    private int timerAdjustment = 0; //relative timer
adjustment
    private Session session;

    //text events
    private boolean showText = false;
    private int timeToShowTextEvent = 0;

    //image events
    private boolean showImage = false;
    private int TimeToNextImageEvent = 0;

    //Audio events
    private boolean audioMuteChanged = false;
    private boolean isAudioMute = false;

    //Exercise index
    private int currentExerciseIndex = 0;
    private int lastExerciseIndex = 0;
    private boolean exerciseIndexChanged = false;

    //synchronization
    private int done = 0;
    private boolean ready = false;

    public SessionMonitor(Session session) {
        isStopped = false;
        isPaused = true;
        lastExerciseIndex = session.size() - 1;
        this.session = session;
    }

    //Time methods

    /**
     * Counts seconds (time duration of the exercise),
triggers events in parallel threads, returns remaining
delay,
     * no need to synchronize threads on ready (250ms
delay by android on button pressed)
     * method used by TimeThread
     */
    synchronized public void waitOneSecond(long delay) {
        boolean done = false;
```

```
        try {
            while (!isPaused && !done &&
!exerciseIndexChanged) {
                wait(delay);
                done = true;
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    synchronized public boolean isTimeToShowText() {
        return showText;
    }

    synchronized public int getTimeToShowTextEvent() {
        return timeToShowTextEvent;
    }

    synchronized public void setTimeToShowTextEvent(int
timer) {
        timeToShowTextEvent = timer + timerAdjustment;
    }

    synchronized public void setShowText(boolean set) {
        showText = set;
        if (showText) notifyAll();
    }

    synchronized public boolean getShowImage() {
        return showImage;
    }

    synchronized public void setTimeToNextImageEvent(int
timer) {
        TimeToNextImageEvent = timer + timerAdjustment;
    }

    synchronized public int getTimeToNextImageEvent() {
        return TimeToNextImageEvent;
    }

    synchronized public void setShowImage(boolean set) {
        showImage = set;
        if (showImage) notifyAll();
    }
```

```
    //used by event threads
    synchronized public void waitForEvent() {
        try {
            while (!isStopped && !showImage && !showText
&& !exerciseIndexChanged) {//use "or" for faster
evaluation??
                wait();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    //used by audio threads
    synchronized public void waitForCommand() {
        try {
            while (!isStopped && !audioMuteChanged &&
!isPaused && !exerciseIndexChanged) {
                wait();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }

    synchronized public void changeAudioState() {//from
view
        audioMuteChanged = true;
        if (isAudioMute)
            isAudioMute = false;
        else
            isAudioMute = true;
        notifyAll();
    }

    synchronized public boolean isAudioMute() {
        return isAudioMute;
    }

    synchronized public boolean isAudioStateChanged(){
        return audioMuteChanged;
    }
    synchronized public void resetAudioStateChanged() {
        audioMuteChanged = false;
    }
```

```
    synchronized public void waitResume() {
        try {
            while (isPaused && !isStopped &&
!audioMuteChanged && !exerciseIndexChanged) {
                if (isStopped)//not needed??
                    stop();
                else
                    wait();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    //Pause methods
    synchronized public void pause(boolean set) {
        isPaused = set;
        notifyAll();
    }

    synchronized public boolean isPaused() {
        return isPaused;
    }

    //Stop methods, reset the player on stop
    synchronized public void stop() {
        isPaused = true;
        isStopped = true;
        //reset all

        notifyAll();
    }

    synchronized public boolean isStopped() {
        return isStopped;
    }

    //Next
    synchronized public void nextExercise() {
        if (currentExerciseIndex < lastExerciseIndex) {
            timerAdjustment +=
session.getExerciseDuration(currentExerciseIndex);
            currentExerciseIndex++;
            exerciseIndexChanged = true;
        } else
            stop();
```

```
        notifyAll();
    }

    synchronized public boolean isExerciseIndexChanged()
{
        return exerciseIndexChanged;
    }

    synchronized public int getCurrentExerciseIndex() {
        return currentExerciseIndex;
    }

    synchronized public void synchronizeAll() {
        done++;

        if (done == TOTAL_AMOUNT_OF_THREADS) {
            done = 0;
            exerciseIndexChanged = false;//reset flags
on sync (switch exercise)
            notifyAll();
        }
        try {
            while (done != 0 && !isStopped) {
                wait();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

TimeThread:

```
public class TimeThread extends Thread {//keeps track of
total time
    private SessionMonitor monitor;

    private final long DEFAULT_DELAY = 1000; //one
second
    private int timer = 0;
    private int currentDuration = 0; //in seconds
    private int sessionDuration = 0; //in seconds
    private Session session;
    SessionPlaybackManager playbackManager;
    SessionPlaybackService sessionPlaybackService;
```

```
    //time measuring
    long time = 0;
    long maxTime = 0;
    long minTime = 2000;
    long totalTime = 0;
    long timeAverage = 0;

    public TimeThread(SessionMonitor mon, Session
session, SessionPlaybackService service) {
        this.monitor = mon;
        this.session = session;
        sessionPlaybackService = service;
        currentDuration =
session.getExerciseDuration(0);//start session
        sessionDuration =
session.getSessionDurationInSeconds();
    }

    private void refreshTimer() {
        if (sessionPlaybackService != null) {
            sessionPlaybackService.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_REFRESH
_TIMER).putExtra(SessionPlaybackManager.TIMER,
getTimer()), 0, 0);
/*
            if (!monitor.isPaused()) {//measure time
taken (intent 1001ms, direct 1002ms)
                if (time == 0)
                    time = System.currentTimeMillis();
                else {
                    time = System.currentTimeMillis() -
time;
                    if (time > maxTime) maxTime = time;
                    if(time < minTime)minTime=time;
                    totalTime += time;
                    time = 0;
                }
            }*/
        }
    }

    public int getTimer() {
        return timer;
    }

    private void switchExercise() {
```

```
        monitor.nextExercise();
        currentDuration +=
session.getExerciseDuration(monitor.getCurrentExerciseIn
dex());//adjust duration

        monitor.synchronizeAll();//wait for all threads

        if (playbackManager != null)
playbackManager.refreshState();
    }

    @Override
    public void run() {
        playbackManager =
SessionPlaybackManager.getInstance();
        long remainingDelay = 0;
        long startTime;
        refreshTimer();//shows 0 at start of exercise
        monitor.synchronizeAll();

        if (playbackManager != null)
playbackManager.refreshState();

        while (!monitor.isStopped()) {

            if (monitor.isPaused())
monitor.waitResume();

            //event triggers
            if (timer == currentDuration) {
                switchExercise();
            }

            if (timer ==
monitor.getTimeToShowTextEvent()) {
                monitor.setShowText(true);
            }

            if (timer ==
monitor.getTimeToNextImageEvent()) {
                monitor.setShowImage(true);
            }


            if (!monitor.isPaused() ||
!monitor.isStopped()) {
                startTime = System.currentTimeMillis();
```

```
                if (remainingDelay != 0)//adjust the
delay on resume from pause

monitor.waitOneSecond(remainingDelay);
                else {

monitor.waitOneSecond(DEFAULT_DELAY);
                }

                if (monitor.isPaused())//during
waitOneSecond
                    remainingDelay = DEFAULT_DELAY -
(System.currentTimeMillis() - startTime);//returns the
remaining delay on pause

                if (!monitor.isPaused() &&
!monitor.isExerciseIndexChanged()) {
                    timer++;
                    remainingDelay = 0;
                }
                refreshTimer();
            }
        }
        sessionPlaybackService.onStartCommand(new
Intent().setAction(SessionPlaybackService.ACTION_STOP),
0, 0);
        try {//stops the thread
            // timeAverage = totalTime / (timer/2);
            join();//joins the UI thread
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

AudioThread:

```
public abstract class AudioThread extends Thread
{//synchronized on pause, stop, resume

    //Declarations
    protected MediaPlayer audio;
```

```
    protected SessionMonitor monitor;
    protected AssetRetriever assetRetriever;
    protected boolean mute = false; //not currently
implemented
    protected boolean looping;

    protected ArrayList<String> audioList;

    //possible updates: volume regulation (relative),
playlist
    public AudioThread(Context context, SessionMonitor
monitor, Session session) {
        this.monitor = monitor;
        assetRetriever = new
AssetRetriever(context.getAssets());
        initAudioList(session);
    }

    @Override
    public void run() {
        initializeAudio();
        monitor.synchronizeAll();
        audio.start();
        audio.pause();

        while (!monitor.isStopped()) {
            if (monitor.isPaused()) {
                audio.pause();
                monitor.waitResume();
                audio.start();
            }

            if (monitor.isExerciseIndexChanged()) {
                switchExercise();
                monitor.synchronizeAll();
                audio.start();
            }
            if (monitor.isAudioStateChanged()) {
                if (monitor.isAudioMute())
                    audio.setVolume(0, 0);
                else
                    audio.setVolume(1, 1);
                monitor.resetAudioStateChanged();
            }

            monitor.waitForCommand();
        }
```

```
            audio.stop();
            audio.release();//release MediaPlayer

            try {//stops the thread
                join();//joins the UI thread
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        private void initializeAudio() {
            audio =
assetRetriever.loadMediaFromAsset(audioList.get(0));
            audio.setLooping(looping);
            try {
                audio.prepare();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }

        private void switchExercise() {
            audio.stop();
            audio =
assetRetriever.loadMediaFromAsset(audioList.get(monitor.
getCurrentExerciseIndex()));
            try {
                audio.prepare();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        abstract void initAudioList(Session session);
}
```

GuideThread:

```
public abstract class GuideThread extends Thread {
    protected Session session;
    protected SessionMonitor monitor;
    protected int currentEventIndex;
    protected int lastEventIndex;
```

```
    protected EventList currentEventList;
    protected SessionPlaybackManager
sessionPlaybackManager;
    protected String currentEvent;
    protected SessionPlaybackService
sessionPlaybackService;

    public GuideThread(Session session, SessionMonitor
mon, SessionPlaybackService service) {
        this.session = session;
        sessionPlaybackService = service;
        currentEventIndex = 0;

        currentEvent = "";
        monitor = mon;
    }

    @Override
    public void run() {
        sessionPlaybackManager =
SessionPlaybackManager.getInstance();
        monitor.synchronizeAll();

        while (!monitor.isStopped()) {

            monitor.waitForEvent();

            if (monitor.isExerciseIndexChanged()) {
                switchExercise();
            }

            if (isTimeToShow()) {
                resetMonitorFlag();
                currentEvent = (String)
currentEventList.get(currentEventIndex).getEvent();//typ
e cast!!!
                updateState();
                if (currentEventIndex < lastEventIndex)
{
                    currentEventIndex++;
                    setNextEventTimer();
                }
            }
        }
        //killThread
        try {
            join();//joins the UI thread
```

```
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        public String getCurrentEvent() {
            return currentEvent;
        }

        //abstract methods
        abstract void switchExercise();

        abstract void resetMonitorFlag();//inside monitor

        abstract boolean isTimeToShow();

        abstract void setNextEventTimer();

        abstract void updateState();
    }
```

LUND
UNIVERSITY

Series of Bachelor's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2016-538
http://www.eit.lth.se