CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

KOMMUNIKATIONSPLATTFORM FÖR VÅRDÄRENDEN

En applikation för kommunikation mellan vårdgivare och vårdtagare

Examensarbete inom Högskoleingenjörsprogrammet i Datateknik

Andreas Bäckevik
Martin Claesson

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

Kommunikationsplattform för vårdärenden
En analys och prototyp inom vården

KOMMUNIKATIONSPLATTFORM FÖR VÅRDÄRENDEN
En applikation för kommunikation mellan vårdgivare och vårdtagare
Examensarbete inom Högskoleingenjörsprogrammet i Datateknik

Andreas Bäckevik, Martin Claesson

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

Institutionen för data- och informationsteknik
Göteborg, 2016

FÖRORD

Denna rapport ämnar att ge läsaren en förståelse för hur mjukvaruutveckling mot svensk vård fungerar i praktiken och vilka svårigheter och hinder som generellt stöts på under utveckling. Rapporten understryker även ett behov av IT-lösningar inom sjukvården och hur det styr de satsningar som företag gör tillsammans med olika nationella institut för att förbättra tillgängligheten av att koppla olika mjukvaror med sjukvården.

Denna rapport har skrivits av två studenter som studerar tredje året på högskoleingenjörsprogrammet datateknik på Chalmers Tekniska Högskola.

Ett stort tack till Chorus som bidragit med information om IT-lösningar inom vården. Samt hur deras lösningar kan röra sig inom lagutrymmet för patientuppgifter samt hur dessa lösningar kan kopplas mot andra aktörer. Ett speciellt tack till Petter Wolff från Chorus som under hela projektets gång har stöttat oss och hjälpt oss på de sätt som vi haft behov utav.

Tack till Praktikertjänst som möjliggjort examensarbetet och givit oss information och underlag för att färdigställa en prototyp. Vi vill speciellt tacka Leif Augustsson för stöd och material samt Marcel Aponno som inspirerat oss och givit en insyn i vilka behov läkare och patient har av en kommunikationsplattform.

Och ett tack till Uno Holmer som under projektets gång hjälpt oss skriva denna rapport och bidragit med andra aspekter till problem och tillvägagångssätt.

SAMMANFATTNING

Vården är i behov av att modernisera de kommunikationssätt som används mellan vårdgivare och vårdtagare. Idag sker kommunikation mellan dessa parter via brev, telefon eller i personmöte. Detta medför långa väntetider och onödig tidsuppbokning för att eventuellt endast leverera resultat. Uppdragsgivaren på Läkarhuset i Göteborg önskar att skapa ett kommunikationssätt som förenklar uppgifter och sparar tid för båda parter. Denna rapport ämnar att besvara hur en IT-plattform inom vården kan konstrueras. Även vilka tjänsteplattformar som kan användas för att ta projektet i produktion och göra det användbart för flera olika vårdkoncerner diskuteras. Samt vilka åtgärder och identifieringar som krävs för att ansluta till en sådan tjänst.

Den prototyp som har utvecklats är en Androidapplikation och en webbapplikation. I Android-applikationen kan vårdtagare besvara hälsoformulär, se stegdata och skicka meddelanden till sin vårdgivare. I webbapplikationen kan vårdgivare lägga till sina vårdtagares mobilenheter och få insamlad data från dessa sammanställt grafiskt. Vårdgivaren kan även skicka meddelanden till de vårdtagare som anslutit sig.
I prototypen lagras ingen patientdata då syftet endast är att visa funktioner som är möjliga att använda i en sådan tjänst. Mobilapplikationen är endast utvecklad för Android-enheter med fokus på att kunna demonstrera funktionalitet. Kommunikationen mellan applikationerna sker via JSON och därför kan andra kommunikations- eller tjänsteplattformar integreras.

ABSTRACT

Healthcare is in need to modernize current ways of communication between doctors and patients. This communication is held through letters, phone calls or in person. This prolongs waiting time and forces both parts to unnecessary meetings just to deliver results. The client at Läkarhuset, Gothenburg wishes to simplify this process and ease administrative burdens that follows. This report describes how to create an IT platform that can be used for healthcare purposes, as well as which service platforms that can be used to make this prototype deployable with live patient data. Followed by which measures and identifications that has to be considered when connecting to these types of services. The prototype that has been developed consists of an Android application and a web application. Patients can answer health forms, see their step data and send messages to their doctor in the Android application. Doctors can add devices that belong to patients and see collected data from the same devices in the web application. Doctors can also send messages to connected patients. No personal information has been stored in the prototype due to the prototype´s only purpose of showing functionality. The smartphone application is developed for Android devices only with focus on demonstrating functionality. The communication between applications is independant and can be integrated with other platforms.

INNEHÅLLSFÖRTECKNING

FÖRKORTNINGAR OCH BEGREPP

SDK	Software Development Kit
API	Application Programming Interface
ReST	Representational State Transfer
HIP	Health Innovation Platform
OAuth	OpenID Authorization
MEAN Stack	Plattformslösning som är byggd på **M**ongoDB, **E**xpress.js, **A**ngular.js, **N**ode.js
S Health	Samsung Health
PHR	Personal Health Records
CMS	Content Management System
CRUD	Create, Read, Update and Delete
ACT	astmakontroll Test
HAD	Hospital Anxiety and Depression Scale
NTjp	Nationella tjänsteplattformen
Bucket	Representerar ett tidsintervall för vilket data har aggregerats
Request	Förfrågan om data till API:et
Inflatera	Aktivera/visa objekt
Autentiseringskod	En slumpmässigt genererad kod på tolv tecken som kopplar anonym data med en vårdtagare. Denna kod kan ej ses av vårdgivare eller vårdtagare utan är endast synlig för backend.
Fragment	Ett objekt i Android Java som är en del av en aktivitet och hålls vid liv av aktiviteten. Fungerar som en kontroller för en bestämd mängd data

1. INLEDNING

I detta kapitel introduceras uppdragsgivaren samt bakgrunden till uppdraget. Vilket syfte projektet har samt vilka frågeställningar projektet syftar till att besvara.

1.1 Bakgrund

Företaget Praktikertjänst är Sveriges största privata koncern inom hälsa och sjukvård [1]. Deras koncern består av olika mottagningar med olika specialistinriktningar. Läkarhuset är en vårdcentral i centrala Göteborg som är en del av Praktikertjänst. Där bedrivs specialistvård, vårdmottagning och sjukgymnastik.

Verksamhetsansvarig läkare på Läkarhuset ser behov av att förenkla kommunikationen med vårdtagare. Nuvarande vårdprocesser orsakar onödigt många återbesök för vårdtagare samt föråldrad kommunikation via brev och samtal mellan vårdgivare och vårdtagare. Detta resulterar i längre vårdtider och mer administrativt arbete för alla inblandade parter.

1.2 Frågeställning

- Hur kan en IT-plattform inom vården konstrueras?
- Vilka tjänsteplattformar kan nyttjas för utveckling mot vården?
- Vilka åtgärder krävs för att ansluta en tjänst till vården?

1.3 Syfte

Examensarbetets syfte är att hjälpa Praktikertjänst att ta fram en prototyp för en mobilapplikation och en webbapplikation där vårdgivare och vårdtagare kan utbyta information. Tjänsten skall ersätta det föråldrade sättet att kommunicera via brev, telefonsamtal och informationsutbyte i form av standardiserade pappersformulär som idag används inom vården. Arbetet kommer även att utvärdera möjligheterna och åtgärderna som krävs för att ta prototypen i produktion. För att demonstrera prototypen kommer formulärsvar, stegmätning och en meddelandetjänst att implementeras. Meddelandetjänsten samt formulärsvar är funktioner som uppdragsgivare önskar i prototypen. Stegmätning kommer att testa möjligheten att implementera andra typer av sensorbaserad mätdata, såsom blodtrycksmätning och viktmätning.

1.4 Avgränsningar

Inget annat än en prototyp för Android-baserade mobilenheter kommer att levereras. Den molntjänst och tillhörande databas som används för prototypen hanterar inga personuppgifter och är därför inte anpassad för vård- och patientuppgiftslagar.

2. TEKNISK BAKGRUND

Kapitlet teknisk bakgrund redogör nödvändig förståelse för de tekniska system, tjänster och plattformar som kommer att användas för att fullfölja projektet.

2.1 Android

Android Studio är en utvecklingsmiljö som är utvecklad av Google och har all grafik- och bibliotekstöd för applikationsutveckling med Android SDK. Den senaste versionen av Android SDK är Android 6.0 och är bakåtkompatibel till och med Android 4.0 vilket täcker 95.3% av hela marknaden [24].

Eftersom Android SDK inte fungerar som ett traditionellt programspråk med hur vyer hanteras, används inte någon traditionell mjukvaruarkitektur. En applikation består av layoutfiler med filtypen XML som används för grafisk design. Aktiviteter kontrollerar applikationens grafiska gränssnitt och olika fragment kan användas för att dela upp ansvaret.

2.2 Dokumentbaserad NoSQL

NoSQL står för “Not Only SQL” och är inte relationsbaserad som till exempel relationsdatabasen MySQL. Denna databastyp nyttjas för att hantera stora mängder data som kan förändras över tid och inte har komplexa relationer [32]. I en relationsdatabas krävs en tydlig struktur som är fördefinierad innan data kan hanteras, till skillnad från NoSQL [33]. Det finns många olika typer av NoSQL-databaser och den som används i detta projekt är dokumentbaserad [32]. Detta innebär att all data är sparad som dokument i varierande format som till exempel JSON eller XML. Alla attribut som är relaterade till varandra är sparade i samma dokument och därav undviks komplexa relationer men mer upprepning av data uppstår. Därför är det viktigt att enbart lagra icke-komplex data i en dokumentorienterad databas för att undvika redundans [31].

2.3 Node.js

Node.js är ett nätverksbaserat JavaScript-ramverk som nyttjar asynkron I/O-kommunikation. Ramverket används för att bygga serverbaserade applikationer. Varje request som görs till applikationen sker som asynkrona händelser. Requests hanteras som callbacks vilket innebär att serverapplikationen spärras i väntan på att slutföra varje request från en klient. Applikationen kan fortsätta köra till nästa uppgift tills att tidigare request är klart.

2.4 Express.js

Express.js är ett ramverk till Node.js som ger utvecklare bra verktyg för att hantera URL:er och requests i sina webbapplikationer. Det är ett open source projekt som utvecklats av entusiaster och större aktörer som IBM [21].

2.5 Mongoose

Mongoose är ett ramverk till Node.js som är till föra att modellera databas-tabeller i MongoDB [5]. Ramverket är open source och kräver endast en uppsatt MongoDB-databas som modellerna kan tillämpas på [46].

2.6 Bootstrap

Bootstrap är ett ramverk till HTML, CSS och JavaScript. Detta ramverk används för att kunna sätta teman och lägga till funktionalitet på grafiska komponenter i de olika ramverken. Bootstrap tillhandahåller även bilder och ikoner som kan användas vid skapandet av grafiska komponenter. Alla tillägg från Bootstrap är open source och utvecklare kan själva bidra med teman och funktionalitet till ramverket [45].

2.7 AngularJS

AngularJS ett JavaScript-ramverk som är utvecklat av Google [19]. Ramverket är klientbaserat och förenklar utveckling enligt mjukvaruarkitekturen MVC.

Angular har egen syntax men vid kompilering omvandlas koden till direktiv i ren JavaScript-syntax genom att analysera direktiven i DOM-trädet för applikationen. Olika webbläsare hanterar JavaScript på olika sätt och därför kan det uppstå kompabilitetsproblem mellan plattformar. I Angular omvandlas koden i webbapplikationen till webbläsaranpassad JavaScript-kod. På detta sätt undviks många kompabilitetsproblem mellan plattformar [15].

2.8 MEAN Stack

Ramverkskombinationen MEAN är en förkortning för **M**ongoDB, **E**xpress.js, **A**ngular.js och **N**ode.js [9][29]. Tillsammans skapar dessa ramverk en systemlösning som kallas MEAN. Detta innebär att all backend och frontend utvecklas i ovan nämnda ramverk och bildar en komplett plattformslösning en så kallad stack [38]. Det finns många olika “stacks” för olika problem och utvecklas efter trender på marknaden och olika företags behov. Ett annat exempel på en sådan stack är LAMP som bygger på att använda Apache som webbserver [28].

2.9 Google Fit

Google Fit är en SDK utvecklad av Google för att samla och lagra hälsodata. SDK:n använder sig av Android-enhetens inbyggda sensorer för att på ett energieffektivt sätt hämta och lagra data [23]. Google Fit SDK innehåller fyra olika API:er- History, Recording, Sensors och Sessions. Hämtning av sensordata i realtid görs med Sensors API och i kombination med Sessions API kan information från flera sensorer slås samman till en aktivitet.

Med hjälp av Google Fit Recording API:s prenumerationer kan data från enhetens sensorer samlas in i bakgrunden. Hanteringen av prenumerationerna sköts av operativsystemet vilket gör att applikationer som nyttjar informationen inte behöver köras för att data skall samlas in [23]. History API används för att hämta data som Recording API har samlat in. Informationen från prenumerationer synkroniseras till History API genom användarens Google konto [14]. Detta möjliggör visualisering av information som samlats in på flera olika enheter.

För att information skall kunna hämtas och synkroniseras krävs att användaren tillåter programmet att synkronisera data till användarkontot hos Google. Identifiering samt inloggning hanteras med OAuth 2.0 vilket möjliggör för applikationen att hantera data i användarens namn [16].

2.10 S Health

S Health är en konkurrerande tjänst till Google Fit [36]. Tjänsten består av en SDK som samlar in data om användarens hälsa via sensorer. SDK:n är utvecklad av Samsung men stödjer Android-enheter från andra tillverkare med Android 4.4 KitKat (API level 19) och uppåt [7]. S Health SDK:n kan delas in i två kategorier, Client SDK och Service SDK, som tillsammans bildar en helhetslösning för sensordata. Client SDK är de gränssnitt som möjliggör digitalisering och lagring av information från såväl interna som externa sensorer till en lokal databas. För externa sensorer används standardgränssnittet ANT+ som hanterar trådlösöverföring av data från olika trådlösa enheter. Till detta gränssnitt finns ett gediget utbud av sensorer på marknaden inom såväl hälsoindustri som för inomhusbelysning och bilindustri [8][22].

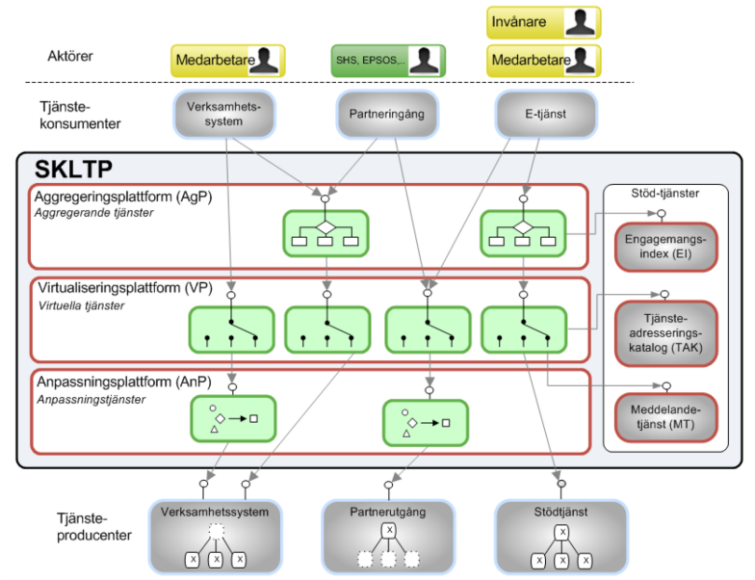
Service SDK tillhandahåller lagring och åtkomst av information som samlats från sensorerna i S Health Personal Health Records (PHR). PHR synkronisera data till användarens S Health konto och ser till att information från tredjepartsapplikationer går att kommat åt på andra enheter än där informationen samlats in.

2.11 Tjänsteplattformarna HIP och NTjp

Tjänsteplattformen HIP centraliserar mjukvaruutveckling för e-hälsotjänster mot NTjp inom vården. NTjp är en nationell plattform dit tjänster som journalsystem kan kopplas för att data skall kunna begäras ut från andra e-tjänster, se figur 2.1. HIP sköter alla loggar och kontrollnivåer som krävs för att begära ut data från ett system i NTjp automatiskt i bakgrunden.

Utveckling mot HIP görs via en SDK som är utvecklad i Java. Denna SDK är helt modulär där utvecklare kan hämta specifikt den information som behövs från den NTjp som är utvecklad av Inera [27]. Beroende på vilka data som hanteras behövs olika autentiseringar, som till exempel samtycke från vårdtagare.

För att få använda HIP SDK behövs ett SITHS-funktionscertifikat [49], vilket är ett certifikat för organisationer som bedriver vård- och omsorgsverksamhet [6].



Figur 2.1 Nationella tjänsteplattformens arkitektur [26]

3. METOD

Metoder och verktyg som används under projektet beskrivs och förklaras i detta kapitel.

3.1 Planering och organisation

För att underlätta projektplaneringen och flexibelt kunna göra nya projektbeslut kommer arbetet att ske agilt. Den agila metod som kommer att användas är baserad på Scrum men på grund av projektgruppens storlek kan inte alla teorier inom Scrum appliceras [2]. Arbetet kommer att utgå från en digital Scrumboard på hemsidan Trello som alla inblandade parter har tillgång till [47]. Detta verktyg visualiserar prioriteringsordningen på en backlogg för de olika funktioner som skall implementeras. Där visas också funktionernas utvecklingsstatus, om de är under utveckling, testning eller om de är färdiga. För att ha möjlighet att utvärdera och förbättra arbetsmetoderna under projektets gång kommer sprintlängd på en vecka att användas. Arbetet under sprintarna fördelas på fyra dagar utveckling samt en dag för dokumentation, sprint review och rapportskrivande [2].

För versionshantering av källkoden kommer ett slutet GitHub-repo att användas [13]. För att kunna hantera och se de olika grenarna i repot kommer Git-verktyget SourceTree att användas [3]. Verktyget visar grenarna grafiskt och vid uppladdning av källkod samt sammanslagning av olika grenar finns stöd för grafisk konflikthantering.

3.2 Utveckling av prototypen

Applikationen kommer att skrivas i utvecklingsmiljön Android Studio 2.0 [17]. Som har stöd för exekvering på Android-emulatorer och fysiska enheter.

Design av applikationen kommer att utgå från Google material design vilket är Googles standard för hur programflödet ska byggas upp. I kombination med material design kommer även nya tillvägagångssätt att testas för att skapa en unik applikation. För att ge applikationen ett unikt utseende kommer flödesdesign att kompletteras med unika bilder som skapas i Photoshop. Alla färgkoder och logotyper ingår i Praktikertjänsts grafiska profil. För att underlätta vidareutveckling och underhåll av applikationen kommer mjukvaruarkitekturen MVC att följas [11].

För att vårdgivare ska kunna sitta vid en dator och få en översikt på data från vårdtagare som använder applikationen skall en enkel webbapplikation utvecklas. I detta projekt är webbapplikationen till för att demonstrera för uppdragsgivaren hur data från användarna av Android-prototypen kan användas av vårdgivaren. Därför kommer endast en hemsida med enkelt grafiskt gränssnitt att implementeras. Hemsidan hämtar data från den molntjänst som används och visualiserar informationen grafiskt. En molntjänst kommer att behövas för att kunna koppla samman den webbapplikation som vårdgivare ska använda för att se sammanställd data från vårdtagarnas mobilapplikation.

3.3 Testning och verifikation

Applikationen kommer att testas i Android Studios emulator på flera versioner av Android. Testning på fysiska enheter kommer att göras i liten skala på en eller ett fåtal enheter. Testningsmetoden som kommer att användas är user based testing [43]. Manuella användartester av prototypens interface har valts då andra alternativ som till exempel testklasser som testar olika moduler är mer tidskrävande att implementera.

4. GENOMFÖRANDE

Vårdtagare skall vid ett läkarbesök kunna be om att få koppla sina mobilenheter till vårdgivaren. Detta möjliggör att vidare kommunikation och datainsamling kan ske från annan ort mellan parterna. Vårdtagare skall kunna lämna formulärsvar och stegmätning så att vårdgivare kan följa utvecklingen av vårdtagarens hälsa utan återupprepade besök. Ifall behovet finns av direkt kommunikation skall båda parter kunna kontakta varandra med direktmeddelanden för att boka besök eller ställa frågor. Vårdgivare skall kunna följa samtliga mobilenheter som kopplats med en vårdtagare och kunna lägga till eller ta bort mobilenheter manuellt. Även vårdtagaren kan avsluta sin uppkoppling manuellt på sin mobilenhet. För detaljerad kravspecifikation se bilaga 5.

En prototyp av den ovan beskrivna tjänsten skall implementeras i syftet av att uppdragsgivaren skall kunna driva införandet av en sådan tjänst inom vården. I denna prototyp kommer ej patientinformation att lagras, endast testdata, därför sker ingen djupgående säkerhetsanalys. Den säkerhet som skall finnas är att all data som skickas är anonym med hjälp av en kod som ges av vårdgivaren. Först när informationen finns hos läkaren kan koden kopplas mot en person.

4.1 Val av molntjänst

För att kunna bygga ett centralt system för all datakommunikation mellan enheter behövs en molntjänst. Den tjänst som används till denna lösning är Digital Ocean [10]. Anledningen till att denna tjänst används är att den är gratis för studenter i ett år och låter utvecklare distribuera egna applikationer i det interna operativsystemet. Molntjänsten kommer med operativsystemet Linux i distributionen Ubuntu [42]. Då ingen patientinformation lagras på molntjänsten läggs ingen fokus på molntjänstens säkerhet utöver kontinuerliga säkerhetsuppdateringar från operativsystemet.

4.2 Databas för vårdtagardata

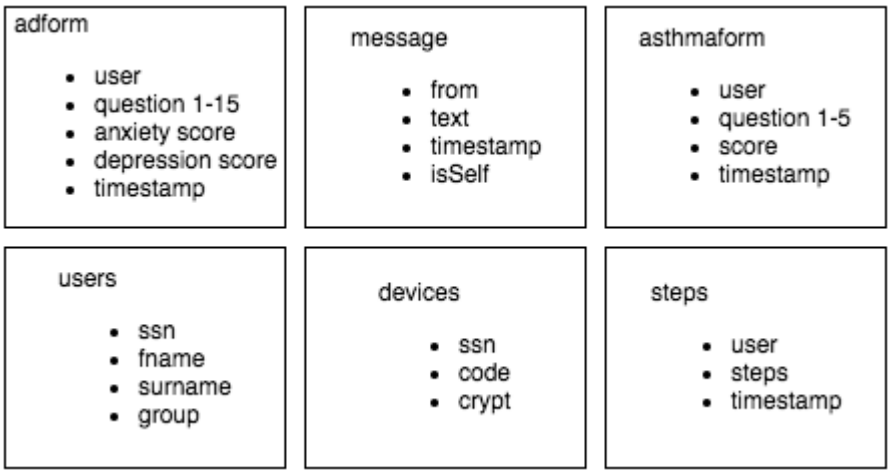
Den databas som används är MongoDB. Denna databas är av typen NoSQL och är dokumentbaserad [37]. Eftersom datan endast kopplas till en nyckel som är unik för enheten finns inget behov av en relationsbaserad databas. Det som krävs av databasen är enbart att den kan lagra stora mängder primitiva datatyper. Tabellerna som finns i databasen ses i figur 4.1.

Data som lagras i formulärtabellerna, ”adform” och ”asthmaform” är totala poängen som vårdtagaren har uppnått på ett formulärsvar. Utöver detta sparas vårdtagarens svar på varje enskild fråga i textform. Detta möjliggör att vårdgivare kan analysera varje enskilt resultat.

I tabellen ”message” finns en sekundärnyckel till vilken vårdtagare som meddelandet är kopplat, d.v.s. autentiseringskoden som genererats. Det finns även en variabel som hanterar vem av vårdtagaren eller vårdgivaren som skrivit meddelandet, samt en tidsstämpel då meddelandet skickades.

Tabellen ”users” håller information om vårdtagare. För att vårdtagare skall kunna koppla upp flera mobilenheter finns tabellen ”devices” som använder vårdtagares personnummer som sekundärnyckel (se figur 4.1). I denna tabell slumpas en sexsiffrig kod som vårdgivaren får se och kan ge till en vårdtagare när denne vill koppla upp sig. Tillsammans med den sexsiffriga koden skapas en autentiseringskod som används vid API-anrop. Varken vårdgivare eller vårdtagare kan se denna sträng som endast är till för API-anslutning. På detta sätt är vårdtagarens data helt anonyma i hanteringen.

Tabellen ”steps” innehåller en sekundärnyckel till den enhet som registrerat stegen, en tidsstämpel när informationen skapades samt datavärdet. All data sorteras efter senaste tillagda tidsstämpeln.



Figur 4.1 Databastabeller

4.3 Utveckling av ReST API

För att kunna kontrollera dataflödet och interna anropa behövs ett API mellan databas och klienter. Då klienternas plattform varierar och endast backend kommer att använda API:et implementerades ett ReST API som enbart hanterar JSON som format [4][40]. Kommunikationen blir då oberoende av plattform, eftersom all information omvandlas till JSON. Detta ReST API är implementerat i CRUD-mönstret kombinerat med standard ReST URL:er [4].

ReST API:et är skrivet i Node.js kombinerat med Express och Mongoose på grund av att dessa ramverk är väl anpassade för varandra. Då detta API endast används i en prototyp och ingen större säkerhet krävs, sparas mycket tid på att använda dessa ramverk. Det finns inga standarder för Node.js till skillnad från ASP.NET eller Java EE och därför går utvecklingen mycket snabbare, men detta medför lägre säkerhet än de mer erkända ramverken till ReST-API:er. Detta på grund av att Node är open source och andra ramverk som till exempel .NET har hårt reglerade standarder [41][30].

Mongoose används på grund av den välutvecklade synergin med MongoDB. Det går enkelt att skapa databasmodeller via Mongoose vilket minimerar tiden för att konfigurera en databas.

Express hanterar alla CRUD-requests och omdirigerar varje request beroende på hur URL:en ser ut. Express agerar i detta fall router för hela API:et, i figur 4.2 finns exempel på hur Express hanterar requests till just astmaformulär enligt CRUD till rätt kontroller.

```
router.route('/forms/asthma')
	.get( asthmaFormController.getForms)
	.post( asthmaFormController.postForms);

router.route('/forms/asthma/:user_id')
	.get(asthmaFormController.getForm);
```

4.2 Hantering av anrop gällande astmaformulär

Hela databasmodellen byggs ifrån API:et med hjälp av Mongoose. Det går därför att skapa väl kontrollerade anslutningar och requests enligt modellen. I figur 4.3 visas ett exempel på hur tabellen för vårdtagare skapas och exporteras till kontrollern av API:et.

```
var UserSchema = new mongoose.Schema({
	ssn: {
		type: String,
		required: true
	},
	fname: {
		type: String,
		required: true
	},
	surname: {
		type: String,
		required: true
	},
	group: { //group can be admin or user
		type: String,
		required: true
	}
});
module.exports = mongoose.model('User', UserSchema);
```

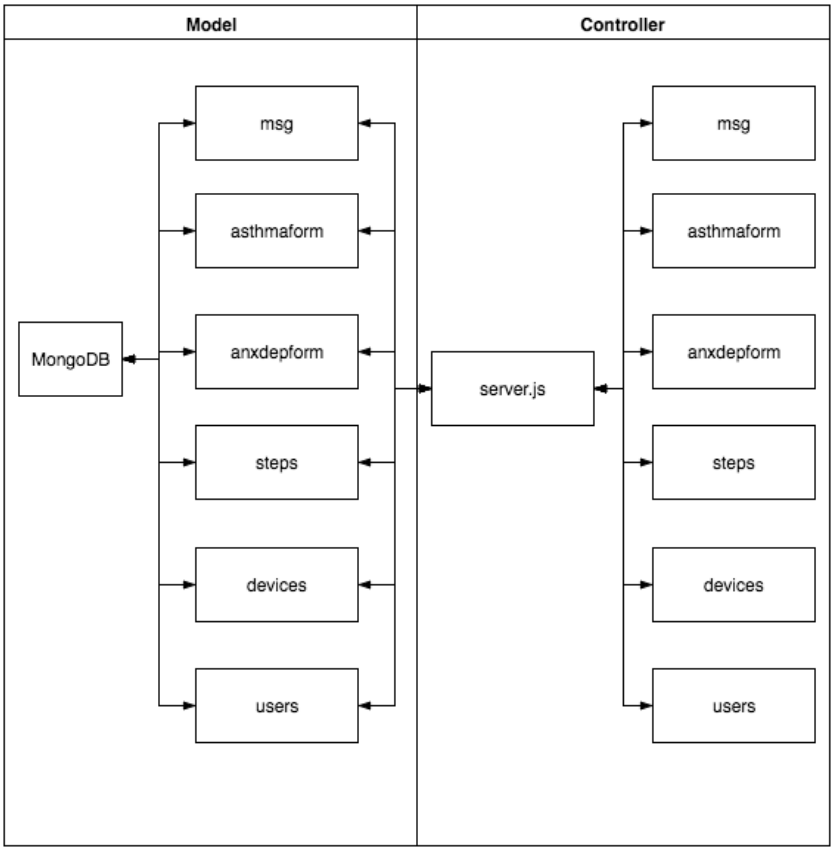
4.3 Databasmodellen av användare som skapas i API:et

I figur 4.4 följer ett exempel på hur data läggs in i databasen. På detta vis skyddas databasen mot SQL-injektioner automatiskt på grund av att hämtning av parametrar sker i JSON från HTTP-paketen som sedan läggs som attribut i en passande objekttyp.

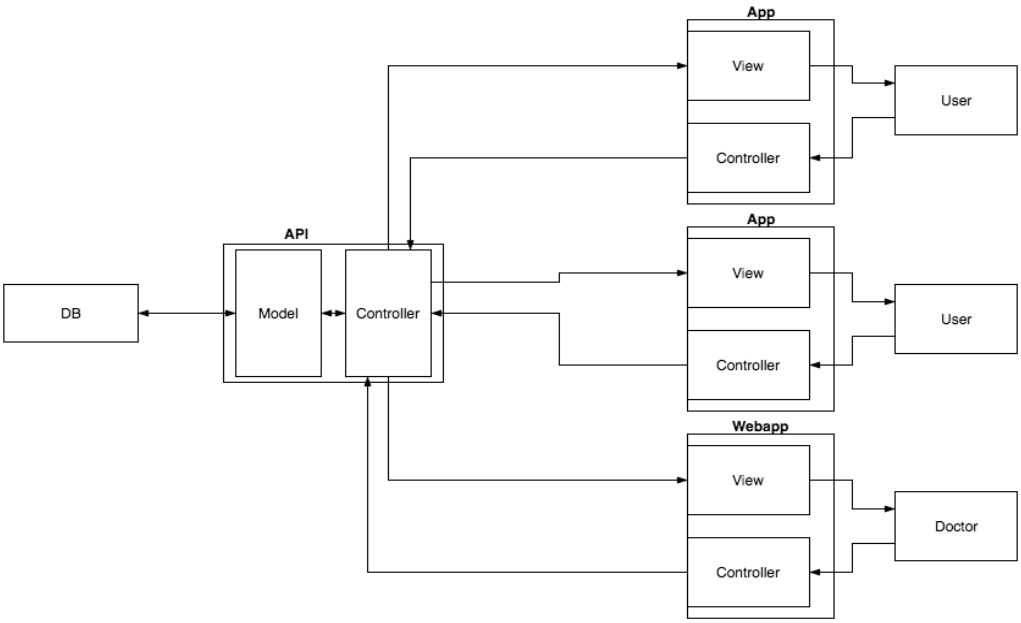
```
exports.postUsers = (function(req,res){
        var user  = new User({
                ssn: req.body.ssn,
                fname: req.body.fname,
                surname: req.body.surname,
                group: req.body.group
        });
        user.save(function(err){
                if(err){
                        res.send(err);
                }else{
                        res.json({ message: 'User created!' });
                }
        });
});
```

4.4 Kontroller för skapandet av ny användare i API:et

API:et bygger upp en modell av hur databasen skall se ut och exporterar sedan modellen till en central kontroller/router. Där dirigeras varje request till en lämplig kontroller beroende på vilken URL som ges, se figur 4.5. Sedan hanteras detta request enligt CRUD och hämtar eller lägger in data i databasen och skickar en passande statuskod enligt HTTP-protokollet tillbaks till anropets ursprung (se figur 4.6) [39].



Figur 4.5 Arkitekturen på ReST-API.



Figur 4.6 Övergripande bild på hela systemet

4.4 Utveckling av Android applikationen

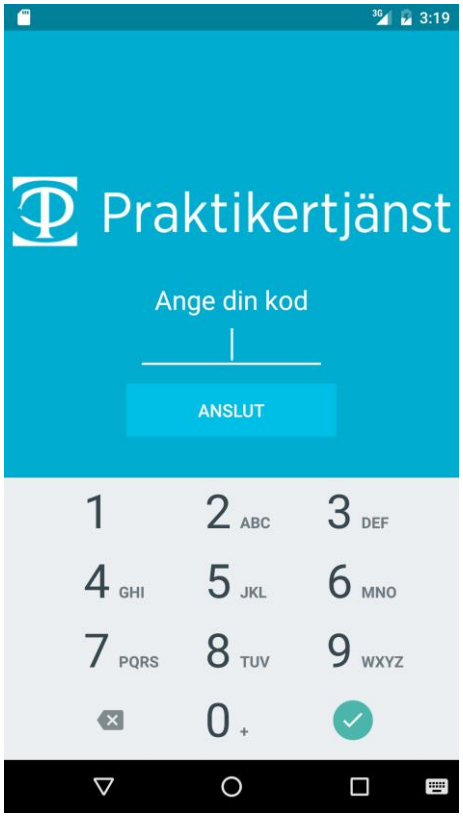
Den Android-applikation som utvecklats är vårdtagarens grafiska gränssnitt för kommunikationen med vårdgivaren. Applikationen samlar och skickar automatiskt användarens stegdata till vårdgivaren. Vårdtagaren kan skicka meddelanden samt standardiserade hälsoformulär till ansluten vårdgivare. Vårdgivaren kan svara på meddelanden samt ta del av hälsodata i ett webbgränssnitt.

4.4.1 Grafisk Vyer

Applikationen har i så stor utsträckning som möjligt utformats enligt Google material design vilket är Googles riktlinjer för applikationsdesign. Google material design innefattar färgsättning, animationer, elevationer på komponenter, layouter med mera. Det som vanligtvis utmärker en applikation som använder sig av Google material design är komponenternas ljussättning och skuggor. Genom att samla alla komponenter för ett objekt i grafiska kort som upphöjs i djupled ger vyerna ett mer levande intryck [18]. Ett exempel finns i figur 4.8 där översiktvyns objekt visualiseras i ett kort. Förutom de riktlinjer som finns för Material Design följer applikationen Praktikertjänsts grafiska profil. Profilen innefattar riktlinjer för användande av logotyp, färgsättning och typsnitt.

Samtliga grafiska komponenter i Android-applikationens vyer är uppbyggda i XML. Därmed är de frånskilda från kontroller och modelldelarna av applikationen. Grafiska komponenter som behöver användas i kontroll-objekt eller modell-objekt i applikationen hämtas från XML-filerna och initieras i respektive fragmentvy. På så vis kan till exempel lyssnare läggas till på knappar. Således blandas vy och kontroller minimalt, vilket är önskvärt då arkitekturmönstret MVC (se bilaga 4) bör implementeras i så stor utsträckning som möjligt. Applikationen har tre huvudsakliga funktioner: meddelanden, stegdata och formulär. Funktionerna har var sin vy där informationen visualiseras och där användaren kan interagera med vårdgivaren genom att skicka information.

För att användare skall kunna nyttja applikationen och tjänsten, måste först mobil-applikationen kopplas med hjälp av vårdgivaren i samband med besök. Vårdtagaren ges en sexsiffrig kod som knappas in vid mötet hos läkaren (se figur 4.7). Koden kontrolleras genom ett anrop till API:t. Om rätt kod angivits returneras en autentiseringskod som sedan sparas i enhetens lokala databas. Denna anonyma kod används sedan vid anrop till API:t vilket gör att ingen information som kopplar identiteten på vårdtagaren finns i applikationen. Om vårdtagaren tidigare kopplat sig till en läkare behövs ingen kod anges vid uppstart av applikationen. Då hämtas istället autentiseringskoden från den lokala databasen på telefonen. Användaren kan ta bort koppling till läkare under inställningar och då kan en ny parkoppling göras.



Figur 4.7 Parkoppling

Figur 4.8 Översiktsvy

När vårdtagaren anslutit sin enhet möts denna av översiktsvyn (se figur 4.8). Härifrån kan användaren gå till respektive aktivitet. Aktiviteternas fragmentvyer inflateras under menybanderollen och gör menyn tillgänglig i alla vyer, så vårdtagaren kan navigera till en annan aktivitet. När innehållet ”glider” in/ut används egenutvecklade animationer enligt Google material designs riktlinjer. Kodexempel på hur fragment inflateras kan ses i figur 4.9. I bilaga 2 sida 1 visas samtliga aktivitetsvyer som nås från översiktsvyn. Från navigationsmenyn i menybanderollen kan användaren nå applikationens meddelandefunktion, logg, inställningar och översiktsvy. Navigationsmeny och vyerna som nås där ifrån kan ses i bilaga 2, sidan 2.

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
fragmentTransaction.setCustomAnimations(R.animator.fade_in, R.animator.fade_out);
GraphViewFragment graphViewFragment = new GraphViewFragment();
fragmentTransaction.replace(
     R.id.app_bar_main_coordLayout, graphViewFragment, "Graph Active");
fragmentTransaction.commit();
toolbar.setTitle("Steg");
```

Figur 4.9 Hur fragmentvy inflateras

4.4.2 ReST API

Kommunikationen med ReST API:et hanteras av en hjälpklass som skapar anslutningen till API:et och bygger med hjälp av angivna parametrar URL:en för ett givet request. Första parametern som anges specificerar vilken typ av request som skall göras. Övriga parametrar innehåller data och varierar därför med request-typen. Klassen hanterar post- samt get-requests, och statuskoden från get-requests innan svaret returneras. I figur 4.10 visas ett exempel på hur autentiseringskoden hämtas vid parkopplingen.

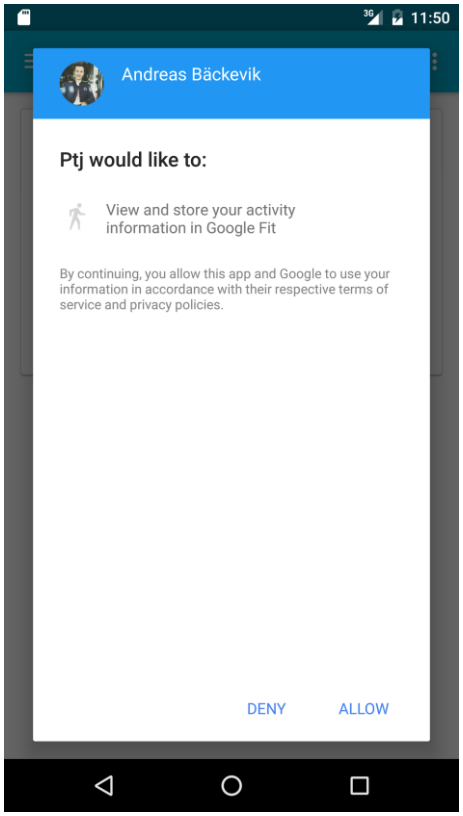
```
if(param[0].equals("pair")){
        url = new URL("http://46.101.96.201:8080/api/devices/connect/"+param[1]);

        urlConnection = (HttpURLConnection) url.openConnection();
        urlConnection.setRequestProperty("Content-Type","x-www-form-urlencoded");
        urlConnection.setUseCaches(false);
        urlConnection.setAllowUserInteraction(false);
        urlConnection.setRequestMethod("GET");
        urlConnection.connect();
        int status = urlConnection.getResponseCode();
        Log.d(getClass().getName().toString(), "Status code "+status);
        switch (status) {
            case 200:
            BufferedReader br = new BufferedReader(new
                      InputStreamReader(urlConnection.getInputStream()));
            sb = new StringBuilder();
            String line;
            while ((line = br.readLine()) != null) {
                      sb.append(line + "\n");
            }
        br.close();
        return sb.toString();
        }
}
```

Figur 4.10 Get-request för parkoppling

4.4.3 Google Fit API

För att möjliggöra datainsamling av användarens steg på ett batterieffektivt sätt och samtidigt stödja insamling från flera enheter används Google Fit API. Av de fyra API:er som Google Fit består av används enbart Recording API och History API. Genom prenumerationer på sensordata från utvalda sensorer i Recording API kan data samlas in utan att applikationen körs. Data som samlats in sparas i History API är i likhet med Recording API kopplat till användarens Google-konto. När applikationen begär ut data i ett visst tidsintervall kommer alla steg som samlas in från samtliga enheter att hämtas. Google Fit API kräver inte något extra konto för informationslagring utan användarens Google-konto nyttjas för att spara information. Då användare med största sannolikhet redan har ett Google-konto för att ladda ner applikationer behövs inget konto skapas för att hantera hälsodata.

Nyttjandet av Google-kontot måste godkännas av användaren. Applikationen använder sig av AOuth 2.0 för att logga in och begära ut data i användarens namn [16]. Första gången applikationen startas och koden som krävs för koppling till läkaren angivits, kommer användaren att bli uppmanad att logga in med sitt Google-konto. Användaren får läsa och godkänna de rättigheter som applikationen önskar ha. Användaren kan i efterhand ta bort rättigheterna att samla och läsa steg i applikationens inställningar. I figur 4.11 visas hur inloggningen ser ut för användaren och figur 4.12 visar hur konversationen startas i programkoden.



Figur 4.11 Acceptera rättigheter Google-konto

```
public void requestPermissions() {
    boolean shouldProvideRationale =
        ActivityCompat.shouldShowRequestPermissionRationale(MainActivity.this,
                Manifest.permission.BODY_SENSORS);
    if (shouldProvideRationale) {
        Snackbar.make(findViewById(R.id.app_bar_main_coordLayout),"text for snackbar",
                Snackbar.LENGTH_INDEFINITE)
                .setAction("OK", new View.OnClickListener() {
                    @Override
                    public void onClick(View view) {
                          ActivityCompat.requestPermissions(MainActivity.this,
                                    new String[]{Manifest.permission.BODY_SENSORS},
                                  REQUEST_PERMISSIONS_REQUEST_CODE);
                }
                }).show();
        } else {
               ActivityCompat.requestPermissions(MainActivity.this,
                      new String[]{Manifest.permission.BODY_SENSORS},
                       REQUEST_PERMISSIONS_REQUEST_CODE);
        }
}
```

Figur 4.12 Kodexempel rättighetsförfrågan

För kunna begära ut data eller lägga till en prenumeration på en sensor måste objetet GoogleApiClient skapas. Tjänsten hanterar vilka API som kan användas samt vilka rättigheter som kan nyttjas inom dessa. I figur 4.13 skapas en GoogleApiClient som använder sig av History och Recording API med rättigheter att läsa och skriva hälsodata. För att applikationen inte skall låsas när ett request på information utförts använder tjänsten sig av callbacks. I figur 4.13 läggs ett ConnectionCallback till för att hantera status för kopplingen till tjänsten. Det är inte förrän detta callback tagits emot som tjänsten är redo för att utföra requests.

```
public void buildFitnessClient() {
        googleApiClient = new GoogleApiClient.Builder(this)
        .addApi(Fitness.HISTORY_API)
        .addApi(Fitness.RECORDING_API)
        .addScope(new Scope(Scopes.FITNESS_ACTIVITY_READ_WRITE))
        .addConnectionCallbacks(
                new GoogleApiClient.ConnectionCallbacks() {
                        @Override
                        public void onConnected(Bundle bundle) {
                        if(googleApiCaller.equals("settings")){
                                FragmentManager fm = getFragmentManager();
                                SettingsFragment settingsFragment =
                                    (SettingsFragment)
                                    fm.findFragmentByTag("SettingsFragment Active");

                                settingsFragment.setRecordSteps(checkStepSubscription());
                         }else if (googleApiCaller.equals("steps")){
                                    getStepsFromWeeks(1);
                        }else if(googleApiCaller.equals("main")){
                                 if(checkStepSubscription()){
                                    getStepsFrom(stepsDBHandler.getData());
                                 }
                        }
}
```

Figur 4.13 Skapande av Google API tjänsten

När tjänsten är skapad och kopplingen är klar kan requests göras till de olika API:er som finns anslutna till tjänsten. I applikationens grafvy hämtas och visualiseras de steg som samlats in under en viss period. Hämtningen för hela den angivna perioden görs i ett request och stegen kommer i buckets där varje bucket innehåller steg för en dag. I figur 4.14 visas hur detta anrop byggs och skickas till API:t samt hur svaret hanteras med ResultCallbacks.

```
final DataReadRequest dataReadRequest = new DataReadRequest.Builder()
        .aggregate(DataType.TYPE_STEP_COUNT_DELTA, DataType.AGGREGATE_STEP_COUNT_DELTA)
        .bucketByTime(1, TimeUnit.DAYS.DAYS)
        .setTimeRange(startTime, endTime, TimeUnit.MILLISECONDS)
        .build();

PendingResult<DataReadResult> pendingResult = Fitness.HistoryApi.readData(googleApiClient,
dataReadRequest);
pendingResult.setResultCallback(
        new ResultCallback<DataReadResult>() {
            @Override
            public void onResult(@NonNull DataReadResult dataReadResult) {
                processData();

                FragmentManager fm = getFragmentManager();
                GraphViewFragment graphView = (GraphViewFragment)
                             fm.findFragmentByTag("Graph Active");

                graphView.setGraphData(steps);
             }
        }
);
```

Figur 4.14 Steg request till Google API tjänsten

4.4.4 Visualisering av stegdata

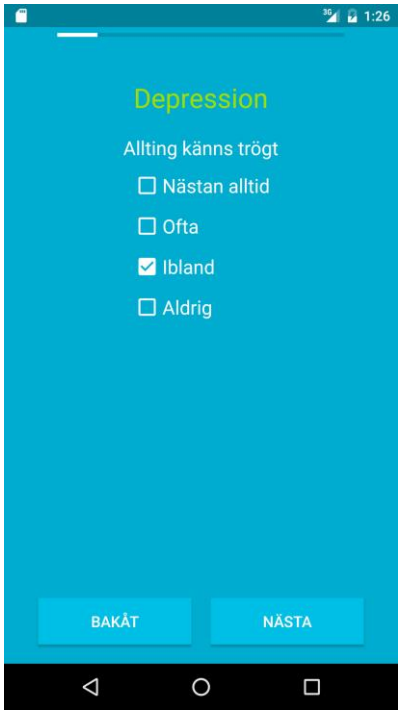
I applikationens stegvy visualiseras användarens samlade stegdata som visas i figur 4.15. När vyn öppnas hämtas automatiskt steg för den senaste veckan och ritas ut i en graf. Data hämtats genom ett request till Google Fit History API. Varje mätpunkt representerar en dag och antalet steg kan avläsas på y-axeln eller vid utritad mätpunkt. Tidsintervallet för vilket stegdata som visas kan ändras genom att markera den kryssrutan med önskat intervall. För att förtydliga grafen för användaren visar x-axeln mellan vilka datum stegen är hämtade.

Figur 4.15 Stegvy

4.4.5 Självskattningsformulär

Formulärfunktionen i applikationen utgår från de standardiserade självskattningsformulären HAD och ACT [44] [25]. De används inom vården för patienter som lider av ångest och depression respektive astma. Formulären har digitaliserats och anpassats till Google material design samt Praktikertjänsts grafiska profil. Allt innehåll från HAD och ACT såsom texter, ordningen på frågor och instruktioner har bibehållits. Formulären är uppbyggda på ett sådant vis att varje fråga poängssätts och avslutningsvis summeras poängen så att en bedömning kan göras. Till skillnad från de fysiska formulären där vårdtagaren själv måste summera sina poäng sköts detta automatiskt av applikationen. I figur 4.16 visas som exempel hur startvyn för ångest- och depressionstestet ser ut. Knappen *INFO* visar instruktionerna för det aktuella testet. Figur 4.17 är ett exempel på hur vyn för frågorna i testen ser ut. Överst i figur 4.17 syns en förloppsindikator som indikerar hur mycket av formuläret som återstår. När samtliga frågor är besvarade skickas svaren på varje enskild fråga samt den totala poängen till ansluten vårdgivare.

Figur 4.16 Startvy formulär



Figur 4.17 Frågeexempel formulär

Varje fråga är uppbyggd av en XML-fil innehållande alla grafiska komponenter och en fragmentklass som agerar kontroller för de grafiska komponenterna. Fragmentklasserna hanterar svaren med hjälp av en bundle som innehåller eventuella tidigare svar. Bundlen skickas med till nästkommande frågeframent och på så viss kan samtliga svar skickas till API:et när frågorna i formuläret besvarats. Kodexempel på en fragmentklass kan ses i bilaga 3.

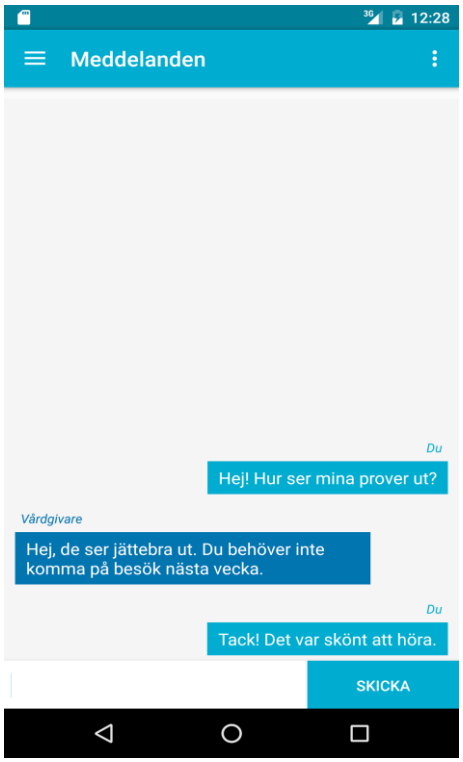
4.4.6 Meddelandefunktion

En av uppdragsgivarens högst prioriterade funktioner är att kunna kommunicera med vårdtagarna i textform. Detta har åstadkommits genom att meddelande skickas och tas emot från ReST API:et. Hjälpklassen som beskrivs i kapitel 4.4.2 hanterar denna kommunikation. I figur 4.18 visas ett kodexempel hur ett meddelande hanteras och skickas. När vyn inflateras hämtas samtliga meddelanden som skickats mellan kopplad vårdgivare och vårdtagare. Detta sker på ett liknande sätt som när nyckeln hämtas vid parkoppling i figur 4.10.

```
url = new URL("http://46.101.96.201:8080/api/message");
urlConnection = (HttpURLConnection) url.openConnection();
urlConnection.setRequestProperty("Content-Type","application/json");
urlConnection.setRequestMethod("POST");
urlConnection.setRequestProperty("key",param[1]);
urlConnection.setUseCaches(false);
urlConnection.setDoInput(true);
urlConnection.setDoOutput(true);

JSONObject jsonParam = new JSONObject();
jsonParam.put("from",param[1]);
jsonParam.put("text",param[2]);
jsonParam.put("timestamp",param[3]);
jsonParam.put("self",param[4]);
DataOutputStream printout = newDataOutputStream(urlConnection.getOutputStream());
urlConnection.connect();
```

Figur 4.18 Försändelse av meddelande till vårdgivare

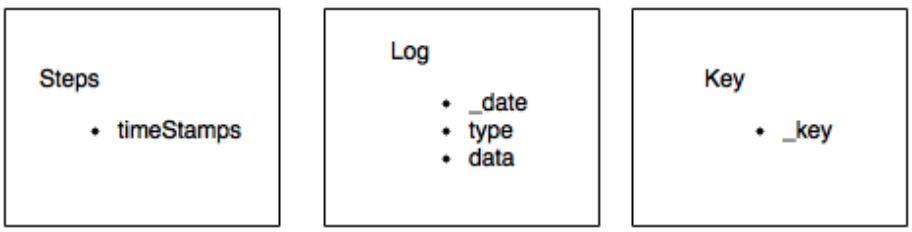
Meddelandevyn byggs upp efter att samtliga meddelanden hämtats genom att en ListView fylls med meddelandeobjekt. Klassen för meddelanden innehåller textmeddelandet samt en boolean som indikerar om det är vårdgivare eller vårdtagare som skrivit meddelandet. Beroende på om vårdgivaren eller vårdtagaren skickat ett meddelande hämtas respektive grafisk layout. I figur 4.19 visas ett exempel på en konversation. Meddelanden från vårdgivare är vänsterjusterad och mörkblå. Vårdtagarens meddelanden är ljusblå och högerjusterade. En text över meddelandena indikerar vem avsändaren är.



Figur 4.19 Meddelandevyn

4.4.7 Lokal databas i Android

Applikationen använder sig av en lokal databas för att lagra information. Databasen innehåller tre tabeller Steps, Log och Key (se figur 4.20). Steps-tabellen har ett heltalsattribut som representerar Unix-tiden för när användarens stegdata senast synkroniserades. I log-tabellen sparas de data som behövs för att loggen skall kunna visualiseras i loggvyn. Tabellen innehåller tre attribut, den typ av data som synkroniseras, datavärdet och Unix-tiden då värdet skickades. Key-tabellen innehåller autentiseringskoden som används vid anrop till API:et för att vårdgivaren skall kunna identifiera avsändaren.



Figur 4.20 Databastabeller i Android

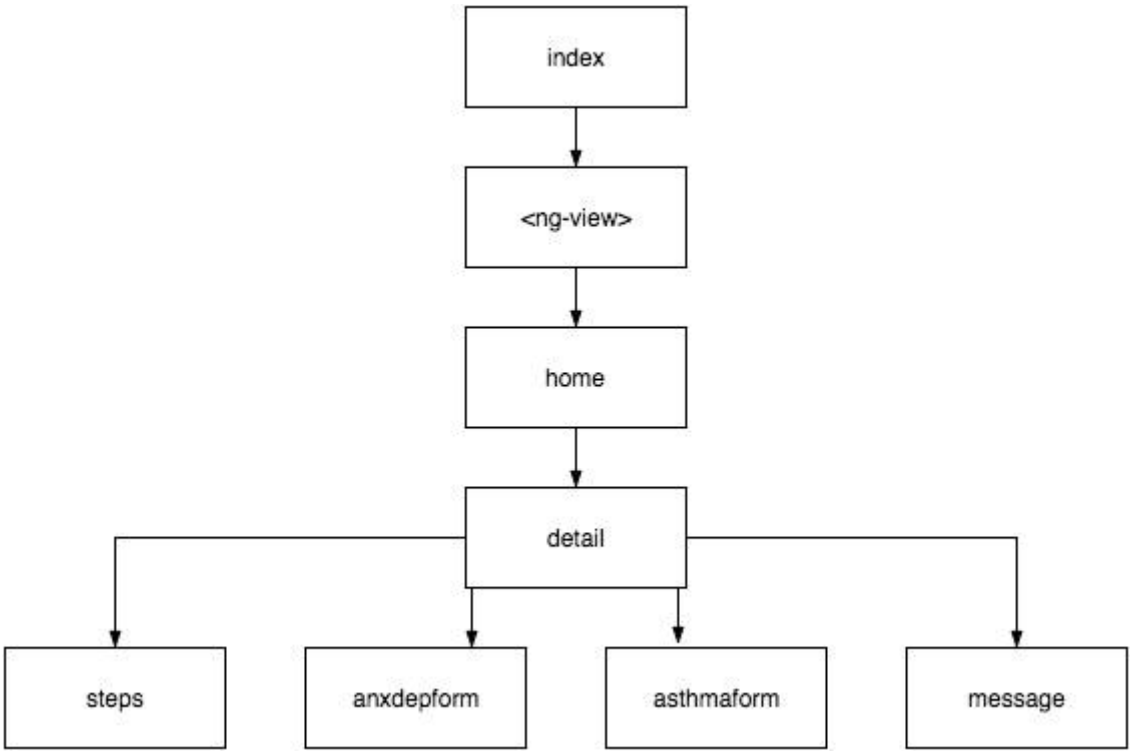
4.5 Utveckling av webbapplikation för vårdgivare

Webbapplikationen fungerar som en CMS-klient för det API som är byggt. Denna klient är endast till för vårdgivaren och ger en överblick över anslutna vårdtagare och all tillhörande information. Vårdgivare kan lägga till nya vårdtagare, koppla nya enheter till enskilda vårdtagare och ta del av den information varje enhet laddat upp till API:et.

Webbapplikationen är byggd med hjälp av HTML 5.0, CSS och JavaScript-ramverket AngularJS [12]. Anledning att ett ramverk används är att i vanlig JavaScript finns ingen systemarkitektur implementerad. Ett ramverk ovanpå JavaScript underlättar därför vid webbutveckling. AngularJS valdes för att språket funnits på marknaden sedan mitten av 2012 och har en väletablerad community [20]. Det finns andra liknande ramverk, till exempel ReactJS, men de är fortfarande relativt nytt på marknaden och därför finns inga ordentliga standarder [34][12].

4.5.1 Design av grafiskt flöde

Den grafiska funktionaliteten en användare ser är skriven i HTML och CSS. Komponenter är uppbyggda med CSS-ramverket Bootstrap, som används till att förfina komponenterna utan att behöva skriva all CSS från grunden [45]. Med hjälp av dessa språk och ramverk skapas ett sidhuvud med en banderoll med uppdragsgivarens logotyp på. Detta sidhuvud ligger kvar genom hela flödet på webbapplikationen (se Bilaga 1). När användaren klickar runt på sidan uppdateras enbart sidans kropp med nytt material. Navigationsflödet anges i figur 4.21. Varje ruta är en egen HTML-fil utom Angular-taggen <ng-view> som är en tagg i indexfilen.



Figur 4.21 Navigationsflöde för webbapplikationen

För att användaren ska kunna navigera genom hemsidan och att varje vy ska fyllas med korrekt information används Angular. Genom att koppla en JavaScript-fil till varje HTML-sida och skapa en kontroller, som sköter kommunikationen mellan JavaScript och HTML, kan Angular ha kontroll över varje HTML-sida.

För att binda en kontroller till varje sida behövs en huvudkontroller. Denna kallas för app.js och sköter rutthanteringar och tilldelningar. Denna kontroller tar reda på den URL som anropats och fyller sedan <ng-view>-taggen i indexfilen med rätt HTML-layout. Rätt kontroller kopplas sedan till rätt layout som sköter visning av data.

I figur 4.22 visas kod från den centrala kontrollern som definierar alla delmoduler i API:et

```
angular.module('myApp', [
  'ngRoute',
  'myApp.detail',
  'myApp.steps',
  'myApp.restfulFactory',
  'myApp.message',
  'myApp.asthma',
  'myApp.anxdep',
  'myApp.home'
])
```

Figur 4.22 Delmoduler som kopplas till den centrala kontrollern myApp

Kodexemplet i figur 4.23 visar hur rutthantering och kontroller bestäms i den centrala kontrollern för visning av astmagrafen.

```
  $routeProvider.when('/asthma', {
    templateUrl: 'forms/asthma.html',
    controller: 'AsthmaCtrl'
  });
```

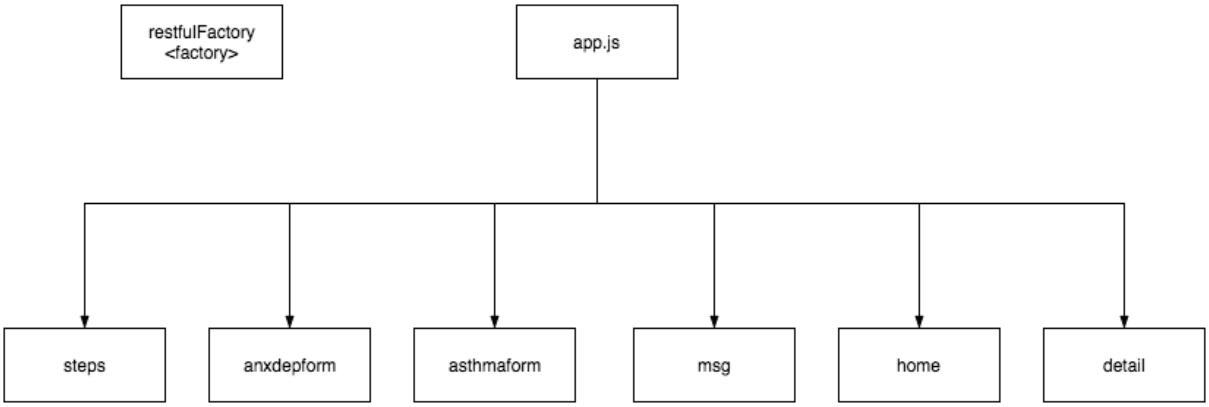
Figur 4.23 MVC struktur på rutthantering i webbapplikationen

Hämtning av data från API:et sköts med hjälp av en factory, som är en singleton. Skillnaden från till exempel Java, är att factory:n är ett instansierat objekt (se figur 4.25). I denna factory ligger alla API-anrop som returnerar svar till webbapplikationen. På detta vis kan rätt data hämtas från en kontroller som grafiskt presenterar detta i en HTML-layout.

I figur 4.24 är ett exempel på hur en factory skickar ett postmeddelande till API:et med ett meddelande och returnerar ett svar från API:et.

```
restfulFactory.postMessage = function(message){
 return $http.post(urlBase+'message/',message, config);
};
```

Figur 4.24 Post request till API från webbapplikationen



Figur 4.25 Hur rutthantering sker samt datahantering i webbapplikationen

5. RESULTAT

I detta kapitel presenteras de resultat som uppnåddes med Android-applikationen, webbapplikationen, API:et och databasen samt deras interaktion som system. Därefter redovisas en tjänsteplattform som kan möjliggöra kommersialisering av prototypen.

5.1 Prototyp för systemlösning

Under projektet har en applikation för Android, en webbapplikation, API och databas utvecklats. Projektets mål var att utveckla en prototyp för en systemlösning där vårdgivare och vårdtagare ska kunna ha en mer effektiv och förenklad kommunikation. Kommunikationen innefattar standardiserade hälsoformulär, meddelande och stegdata. Genom att uppnå dessa mål möjliggörs syftet för uppdragsgivare på Läkarhuset att demonstrera systemet och på så vis kunna driva på införandet av en liknande systemlösning på koncernnivå.

5.2 Android-applikation

Fokus under utveckling har varit att implementera de funktioner som uppdragsgivaren upplevt vara viktigast och som berör flest vårdtagare. Applikationens design är utvecklad enligt Google material design för att vara lätthanterad och tydlig att använda för vårdtagare i alla åldrar. Applikationen låter vårdtagare koppla sin Android-telefon till att passivt skicka data till vårdgivare med en sexsiffrig kod som enbart vårdgivare kan ge ut.
Därefter tillåts vårdtagare kontakta sin personliga vårdgivare vid behov och även skicka in enklare information till vårdgivaren, som till exempel formulärsvar och stegdata. Genom dessa möjligheter kan både vårdgivare och vårdtagare spara tid.

5.3 Databas

Databasen är utvecklad till att hålla information om vårdtagare och dennes kopplade enheter tillsammans med den data som varje kopplad enhet samlar in.

5.4 API

API:et är utvecklat till att kontrollera och skicka data anonymt och endast koppla vårdtagares data med en autentiseringskod som ingen användare kan se. Det är inte utvecklat för de lagar som gäller för lagring av patientdata, utan enbart i syfte att demonstrera hur koppling mellan vårdgivare och vårdtagare kan tänkas se ut.

5.5 Webbapplikation

Webbapplikationen är utvecklad för att användas av vårdgivare och fokus har legat på att skapa ett gränssnitt som är informativt och i enlighet med den grafiska profil som uppdragsgivare använder.

Vårdgivaren kan genom webbapplikationen lägga till nya vårdtagare och även nya enheter till befintliga vårdtagare. Genom tillagda enheter kan vårdgivare se detaljerad information om den data som samlas in på enheten.

6. DISKUSSION

I detta kapitel presenteras tankar om de tekniker som använts för att uppnå resultatet i from av en prototyp. Diskussion angående potentiell vidareutveckling och samhällspåverkan av införandet av prototypen i vården kommer även att föras.

6.1 Vidareutveckling av prototyp

För att kunna ta protoypen till produktion krävs att den backend som prototypens systemlösning innehåller ersätts av en backend som följer alla de lagkrav som finns i Sverige och i EU. Den måste även vara kopplad till den Nationella tjänsteplattformen (NTjp) för att kunna ta del av patientdata från sjukvården.

Skall systemet kopplas direkt till den NTjp krävs ett samarbete med företaget Inera som utvecklat plattformen för att ansluta till tjänsteplattformen. Stegen som krävs för att realisera prototypen består av följande:

1. Kund fyller i och skickar in förstudiemall till Inera för godkännande av anslutning mot testmiljöerna.
2. Kund skickar in beställning av anslutning till testmiljöerna.
3. Inera förbereder tjänsteplattformen för tester.
4. Kund testar och verifierar anslutningen tillsammans med Inera.
5. Kund skickar in förstudiemall för godkännande av anslutning mot produktionsmiljöerna.
6. Kund skickar in beställning av anslutning till produktionsmiljöerna.
7. Inera förbereder tjänsteplattformen för produktion.
8. Kund testar och verifierar anslutningen tillsammans med Inera.
9. För att sedan kunna etablera samverkan med en producent behöver dessa steg göras om tillsammans med varje samverkande producent.

Denna process är tidskrävande och kräver samverkan mellan många olika parter. Detta höjer utvecklingskostnader och utvecklingstider [48].

För att undvika dessa komplexa steg kan till exempel tjänsten HIP användas. Denna tjänst minskar komplexiteten genom att ta hand om anslutningen och intern loggning mot NTjp. Här kan kund utveckla tjänster mot det begränsade antal resurser som HIP tillhandahåller. HIP förväntas innehålla ett större antal resurser i framtiden som tas fram tillsammans med Karolinska Institutet.

En fas i projektet bestod av att testa HIP 2.0 som är under betatestning. Detta för att undersöka de möjligheter som finns för att ta prototypen i produktion. För implementering av SDK:n behövs ett projekt i Maven som är en webbaserad applikation i Java-ramverket Spring.

6.2 Värderingar kring MEAN

Om en utvecklare väljer att bryta ny mark i utvecklingsprocessen genom användning av nya programmeringsspråk, betyder det både för- och nackdelar. En fördel är att utvecklaren är väldigt fri i processen och kan utforma programkoden efter de behov som finns i projektet. Detta är på grund av att nya programspråk tenderar att inte ha en satt industristandard utan endast spekulativa standarder som tas fram på diverse forum. Detta är också en av de större nackdelarna, programkoden är därför svår att ta över för en annan utvecklare då det sannolikt ej finns en standard att följa i programstrukturen.

I en MEAN Stack är alla programspråken som används baserade på JavaScript, vilket betyder att det går fort att utveckla ett system om utvecklaren har kunskaper inom JavaScript. Tekniken i sig börjar bli mer mogen då detta arbetssätt har funnits sedan 2013. Flera företag som till exempel Netflix, Uber, PayPal och Yahoo använder Node.js för sin backend. Detta tyder på att flera stora aktörer tror på tekniken och att den går att använda i större projekt.

En mycket stor fördel med MEAN är att all data hanteras automatiskt mellan ramverken och behöver därför enbart hanteras i databasen. På så vis undviks problem med formateringar och plattformar då denna systemlösning kan köras på både Windows, Linux och OSX.

Troligtvis kunde andra programspråk ha används för den utveckling som bedrivits under projektet. Med MEAN går utvecklingen fort och många hinder undviks då de tas hand om av ramverken. Tillsammans med att detta tillvägagångssätt blir mer vanligt i industrin och accepterat bland större företag kommer tekniken som används att vara modern i flera år framöver.

6.3 Val av plattform för hälsodata

Under projektet har två plattformsalternativ för hälsodata granskats, Google Fit och S Health. Tjänsterna konkurrerar om samma marknad och har liknande funktionalitet. Men styrkor och svagheter finns självklart i respektive tjänst. Google Fit API används i prototypen då tjänsten stödjer testning i Android Studios emulator, vilket S Health inte gör. Möjligheten att kunna testa applikationen direkt i utvecklingsverktyget har underlättat utvecklingsprocessen. En annan fördel med Google Fit är att den till skillnad från S Health använder sig av användarens Google-konto för datalagring och synkronisering, således behövs inget extra konto.

6.4 Samhällsaspekt

En kommersialisering och införande av en utvecklad prototyp i sjukvården kan medföra många positiva samhällsaspekter. Genom att minska antalet möten mellan vårdtagare och vårdgivare minskar sjukvårdskostnader för staten. Resultatet av detta är kortare kötider och minskat antal resor till och från vårdmottagningen. Detta leder förhoppningsvis till färre resor med färdtjänst och andra färdmedel som bekostas av staten. Som i sin tur har en positiv inverkan på samhällets infrastruktur. Kontinuerlig kontakt med läkare i form av till exempel hälsoformulär samt möjlighet till direktkontakt med läkare skapar trygghet i hemmet för vårdtagaren. Tillgången till fler mätpunkter och typer av hälsodata som plattformen tillhandahåller, möjliggör för vårdgivaren att införa förebyggande vårdåtgärder.

7. SLUTSATS

Slutkapitlet ämnar att redogöra och värdera de slutsatser som dragit kring applikationsplattformen som utvecklats. Samt redovisa möjliga val av tjänsteplattform för vidareutveckling.

7.1 Applikationsplattform

En applikationsplattform mellan vårdtagare och vårdgivare kan konstrueras på ett flertal olika sätt. Fördelen med att bygga en tjänst med en mobiltjänst för vårdtagare och en webbtjänst för vårdgivare är många. Vårdgivare har höga säkerhetskrav på inloggning och använder sig av SITHS-kort kopplade till en PC för att identifiera sig. Vanligtvis skriver vårdgivare journal i PC-miljö. Därför lämpar sig en lösning med stöd för arbete på en och samma enhet [49].

För vårdtagare är tillgängligheten av hög prioritet och därför lämpar sig en mobilapplikation. Möjligheterna till notifikationer i mobilapplikationen säkerställer en kontinuerlig kontakt med vårdgivaren. En Android-baserad mobilapplikation möjliggör även datainsamling från såväl externa som interna sensorer. Expansionsmöjligheterna för nya funktioner och datainsamlingsmetoder via till exempel Bluetooth är därmed stora på denna plattform.

7.2 Möjliga val av tjänsteplattform

Vid val av tjänsteplattform för implementering av en kommunikationsplattform är valmöjligheterna inte lika stora. Efter att ytligt ha analyserat marknaden har två alternativ identifierats, utöver alternativet att driva all utveckling själv utan någon tjänsteplattform. För att utveckla applikationer utan tjänsteplattform krävs utveckling av lösningar mot specifika system. För projekt av större skala där stöd för lagring av nya resurser krävs lämpar sig egenutveckling och anslutning till NTjp. Detta medför långa utvecklingstider och höga utvecklingskostnader då riktlinjer från Inera måste följas. Fördelen är att detta medför full kontroll över systemet och nya resurser kan läggas till. Dock måste utförliga tester göras för varje ändring som görs för att systemet skall behålla sin CE-märkning [35]. Det andra alternativet är att använda sig av HIP SDK. HIP hanterar autentisering och intern kommunikation som loggning av data som begärs från NTjp. Då de resurser som finns tillgängliga i SDK:ens olika API:er motsvarar de juridiska krav som finns för persondata underlättas processen för programvarans kvalitetsmärkning avsevärt [6]. Därmed lämpar sig HIP för projekt med kort utvecklingstid eller där vidareutveckling av nya tjänster i systemet förväntas. HIP SDK är fortfarande under utveckling och alla resurser som krävs för vår prototyp stöds ej i dagsläget. Trots att HIP inte har alla funktioner som finns i prototypen är HIP mest sannolikt den lösning som inom en snar framtid kan användas för att realisera en kommersialisering av prototypen.

KÄLLFÖRTECKNING

[1] P. AB, “Om Praktikertjänst.” [Online]. Available: http://www.praktikertjanst.se. [Accessed: 15-Apr-2016].

[2] S. ALLIANCE, “Scrum.” [Online]. Available: https://www.scrumalliance.org. [Accessed: 22-Apr-2016].

[3] I. Atlassian, “SourceTree.” [Online]. Available: https://www.sourcetreeapp.com/. [Accessed: 22-Apr-2016].

[4] R. Battle and E. Benson, “Bridging the semantic Web and Web 2.0 with representational state transfer (ReST),” Web Semant. Sci. Serv. Agents, 2008.

[5] K. Chodorow, MongoDB: the definitive guide. 2013.

[6] Chorus, “HIP.” [Online]. Available: http://hip.se/hip-sdk/. [Accessed: 13-May-2016].

[7] S. E. Co, “Programming Guide : Samsung Digital Health - Health Data,” pp. 1–37.

[8] S. E. Co, “S Health SDK,” 2014-02-26. [Online]. Available: https://news.samsung.com/global/s-health-sdk. [Accessed: 06-May-2016].

[9] S. Davis, “Mastering MEAN: Introducing the MEAN stack,” IBM.com, 2014. [Online]. Available: http://www.ibm.com/developerworks/library/wa-mean1/index.html.

[10] Digital Ocean, “Cloud computing.” .

[11] Robert Eckstein, “MVC,” *March*, 2007. [Online]. Available: http://www.oracle.com/technetwork/articles/javase/index-142890.html. [Accessed: 12-May-2016].

[12] Facebook Inc, “ReactJS.” [Online]. Available: https://facebook.github.io/react/. [Accessed: 12-May-2016].

[13] I. GitHub, “GitHub.” [Online]. Available: https://github.com. [Accessed: 22-Apr-2016].

[14] Google Inc, “Google konto.” [Online]. Available: https://support.google.com/accounts/topic/2373943?hl=en&ref_topic=3382296. [Accessed: 29-Apr-2016].

[15] Google Inc, “AngularJS.” [Online]. Available: https://angularjs.org/. [Accessed: 06-May-2016].

[16] Google Inc, “OAuth 2.0,” 2016-01-25. [Online]. Available: https://developers.google.com/fit/android/get-api-key#release-cert. [Accessed: 29-Apr-2016].

[17] Google Inc, “Android Studio.” [Online]. Available: http://developer.android.com/tools/studio/index.html. [Accessed: 22-Apr-2016].

[18] Google Inc, “Material Design.” [Online]. Available: https://www.google.com/design/spec/material-design/introduction.html. [Accessed: 11-May-2016].

[19] B. Green and S. Seshadri, “AngularJS,” 2013.

[20] M. Hevery, “AngularJS Release,” developers.googleblog.com, 2012. [Online]. Available: https://developers.googleblog.com/2012/06/better-web-templating-with-angularjs-10.html. [Accessed: 12-May-2016].

[21] O. IBM, “Express.” [Online]. Available: http://expressjs.com/. [Accessed: 29-Apr-2016].

[22] D. I. Inc, “Ant+.” [Online]. Available: https://www.thisisant.com/consumer/ant-101/what-is-ant/. [Accessed: 09-May-2016].

[23] G. Inc, “Google Fit.” [Online]. Available: https://developers.google.com/fit/. [Accessed: 29-Apr-2016].

[24] G. Inc, “Android version distributions,” 2016. [Online]. Available: http://developer.android.com/about/dashboards/index.html. [Accessed: 22-Apr-2016].

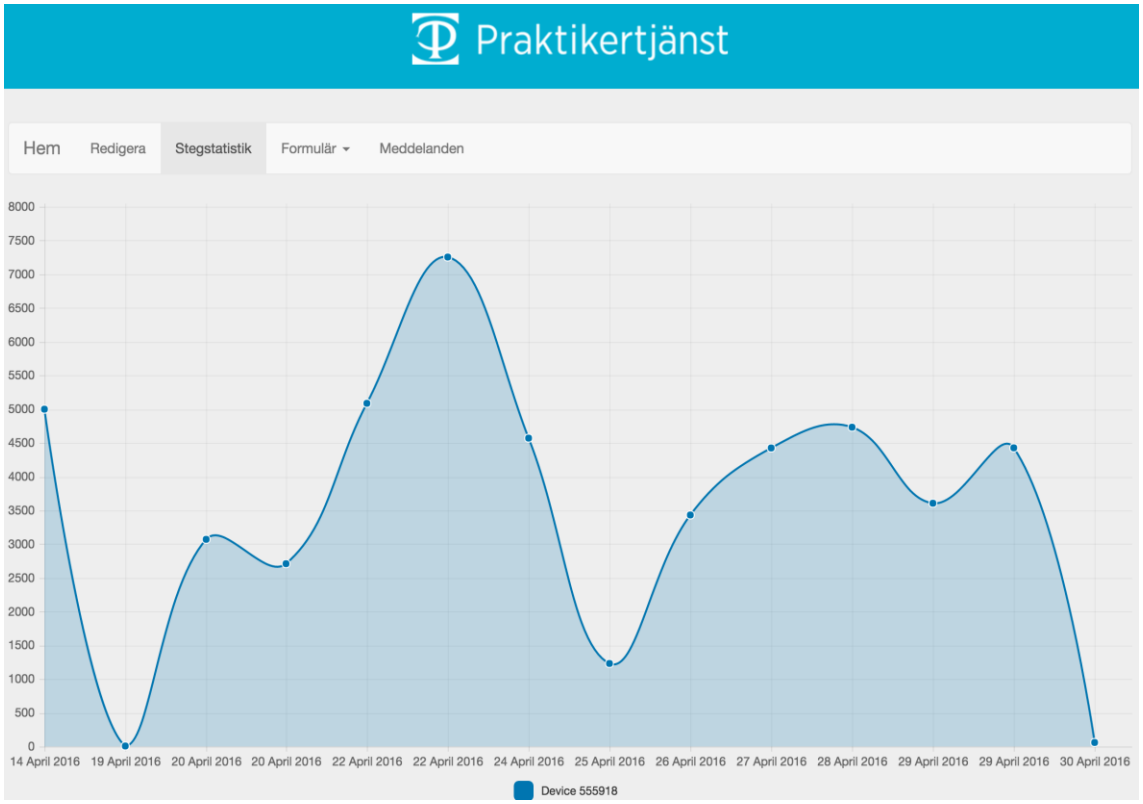
[25] Q. Incorporated, A. C. Test, and Q. Incorporated, “A C T,” pp. 2–3.

[26] Inera, “Teknisk översikt NTjp.” [Online]. Available: https://skl-tp.atlassian.net/wiki/display/SKLTP/skltp+Home. [Accessed: 13-May-2016].
[27] Inera, “Nationella Tjänsteplattformen.” [Online]. Available: http://skltp.se/. [Accessed: 13-May-2016].
[28] G. Lawton, “LAMP lights enterprise development efforts,” Computer (Long. Beach. Calif)., 2005.
[29] Linnovate, “MEAN documentation.” [Online]. Available: http://learn.mean.io/. [Accessed: 29-Apr-2016].
[30] Microsoft Inc, “ASP.NET.” [Online]. Available: http://www.asp.net/web-api. [Accessed: 09-May-2016].
[31] MongoDB, “Document-oriented Databases.” [Online]. Available: https://docs.mongodb.org/manual/introduction/#document-database. [Accessed: 29-Apr-2016].
[32] MongoDB, “NoSQL Explained.” [Online]. Available: https://www.mongodb.com/nosql-explained. [Accessed: 29-Apr-2016].
[33] Oracle Inc, “Relational Database Overview.” [Online]. Available: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html. [Accessed: 29-Apr-2016].
[34] P. O’Shannessy, “ReactJS Release,” facebook.github.io, 2014. [Online]. Available: https://facebook.github.io/react/blog/2014/03/28/the-road-to-1.0.html. [Accessed: 12-May-2016].
[35] H. I. Platform, “CE-märkning.” [Online]. Available: http://hip.se/ce-guiden/. [Accessed: 17-May-2016].
[36] I. Samsung, “Health Data | Programming Guide,” 2016-02-25. [Online]. Available: http://developer.samsung.com/technical-doc/view.do?v=T000000197. [Accessed: 29-Apr-2016].
[37] C. Strauch, U. Sites, and W. Kriha, “NoSQL databases,” Lect. Notes, Stuttgart Media, 2011.
[38] Techopedia, “Solution Stack.” [Online]. Available: https://www.techopedia.com/definition/28154/solution-stack. [Accessed: 29-Apr-2016].
[39] The Internet Assigned Numbers Authority, “Hypertext Transfer Protocol (HTTP) Status Code Registry,” http://www.iana.org/, 2016. [Online]. Available: http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml. [Accessed: 10-May-2016].
[40] R. Thomas, “Architectural Styles and the Design of Network-based Software Architectures,” Doctoral dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Accessed: 09-May-2016].
[41] S. Tilkov and S. Vinoski, “Node. js: Using javascript to build high-performance network programs,” IEEE Internet Comput., 2010.
[42] Ubuntu, “Ubuntu.” [Online]. Available: http://www.ubuntu.com/. [Accessed: 20-Apr-2016].
[43] L. White, H. Almezen, and N. Alzeidi, “User-based testing of GUI sequences and their interactions,” Reliab. Eng. 2001. …, 2001.
[44] Zigmond and Snaith, “HAD,” vol. 93, no. 12, pp. 884–892, 1996.
[45] “Bootstrap.” [Online]. Available: http://getbootstrap.com/. [Accessed: 12-May-2016].
[46] “Mongoose.” [Online]. Available: http://mongoosejs.com/. [Accessed: 12-May-2016].
[47] “Trello.” [Online]. Available: www.trello.com. [Accessed: 22-Apr-2016].

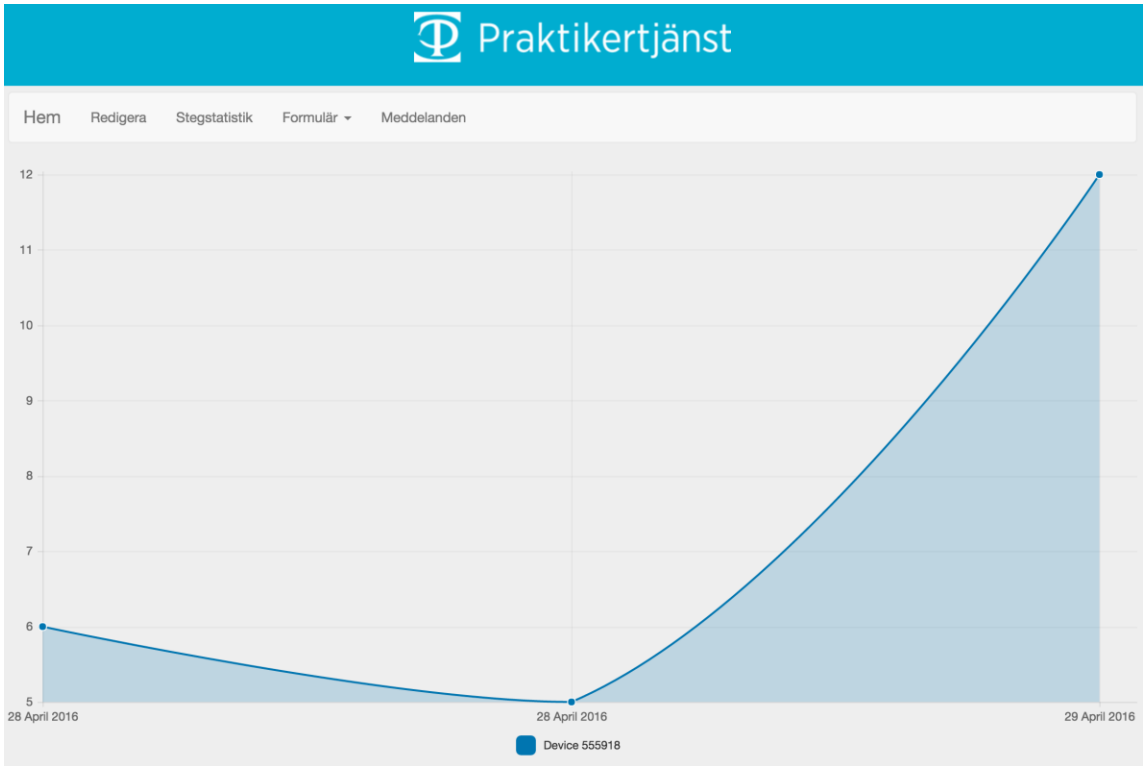
[48] Inera, “Anslutning till Tjänsteplattform,” 2015. [Online]. Available: http://www.inera.se/TJANSTER--PROJEKT/ICC-Integration-Competence-Center/Anslutning-till-Tjansteplattformen/. [Accessed: 16-May-2016].
[49] Inera, “SITHS Certifikat,” 2015. [Online]. Available: http://www.inera.se/siths. [Accessed: 16-May-2016].

Listan som visar alla vårdtagare i webbapplikationen

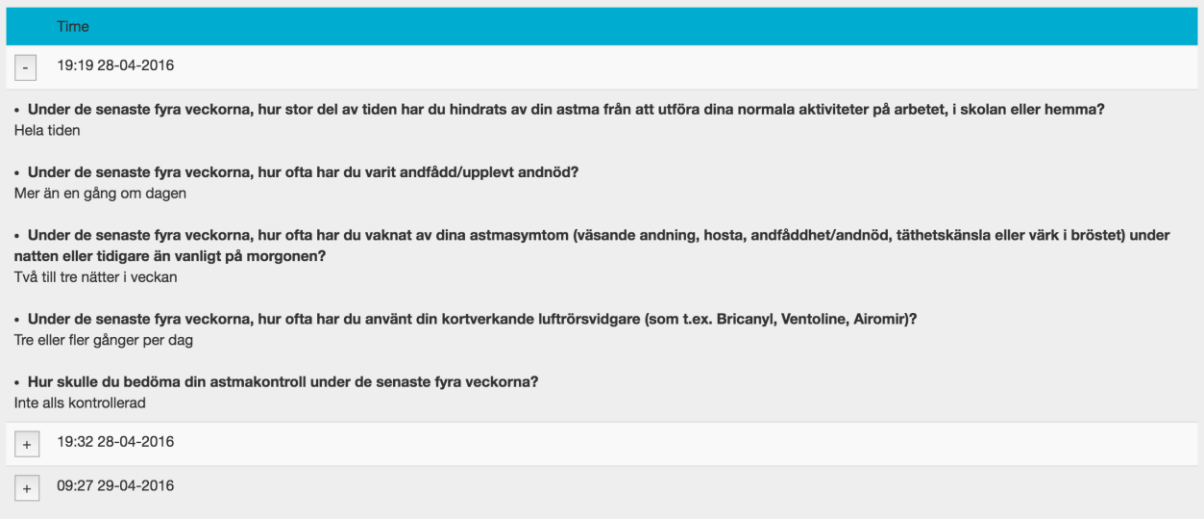
Detaljerad vy över en vårdgivare i webbapplikationen



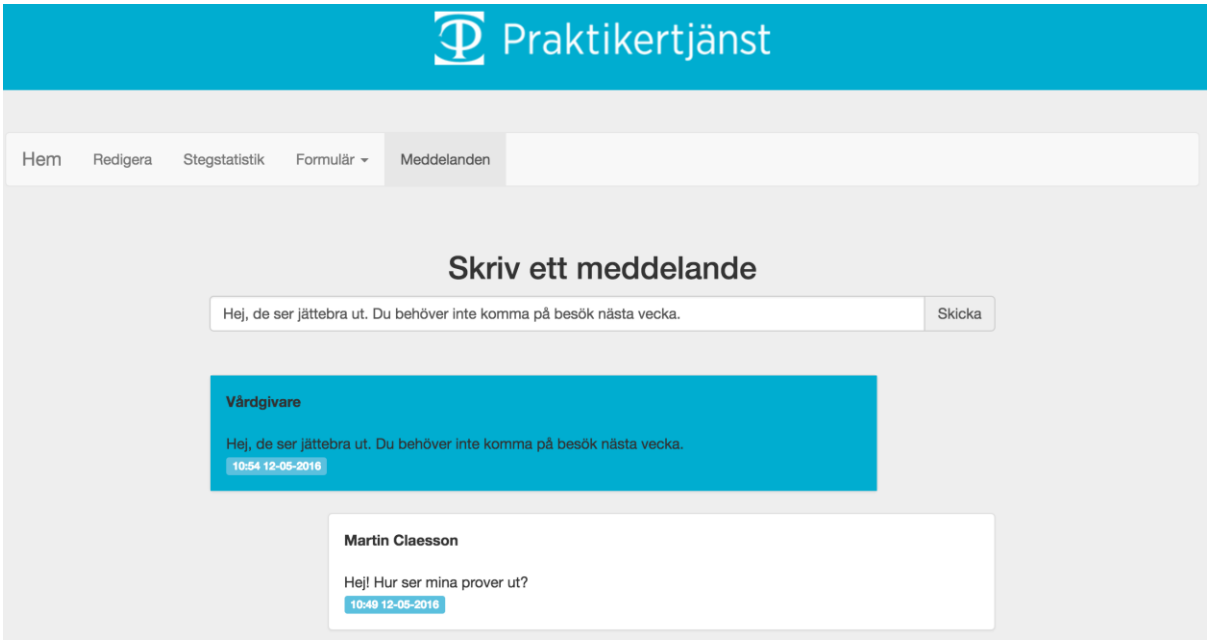
Graf i webbapplikationen som representerar en vårdtagares steg som samlats via en enhet



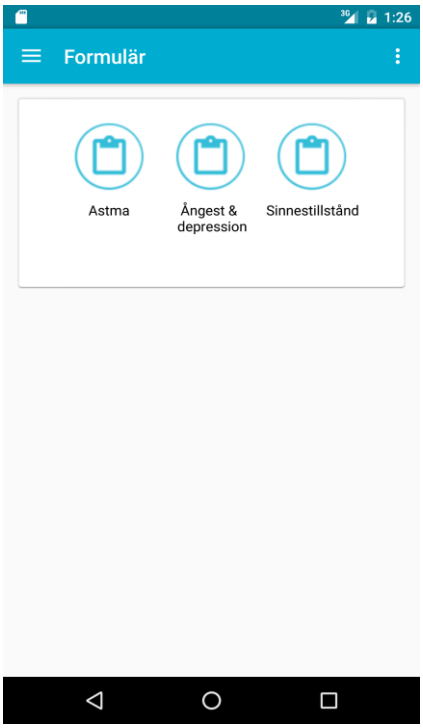
Graf i webbapplikationen som representerar en vårdtagares formulärsvar som samlats in



Frågor och svar från en vårdtagares formulärsvar som visas i webbapplikationen



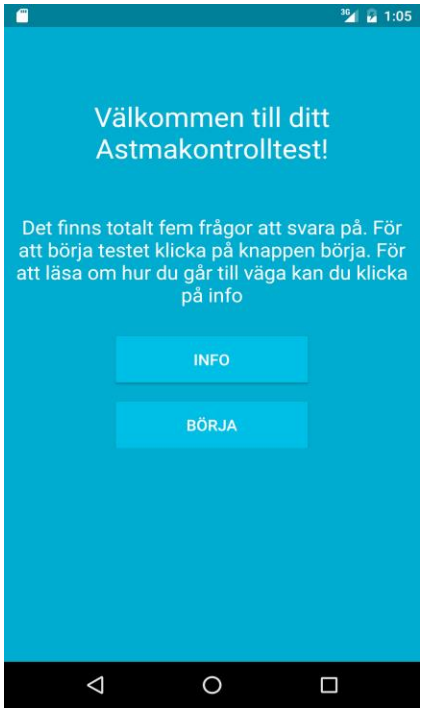
Meddelandefunktionen mellan vårdgivare och vårdtagare i webbapplikationen



Vyn för val av formulär i mobil-applikationen

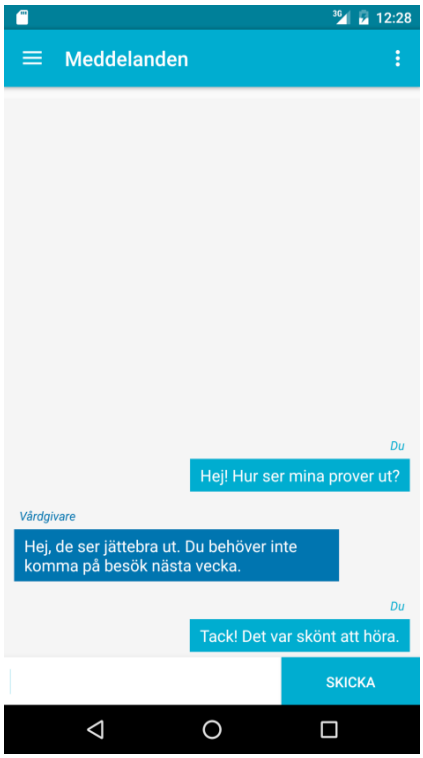
Vyn för att visa samlade steg i mobil-enheten

Startvyn för att svara på ångest- och Depressionstestet i mobil-enheten

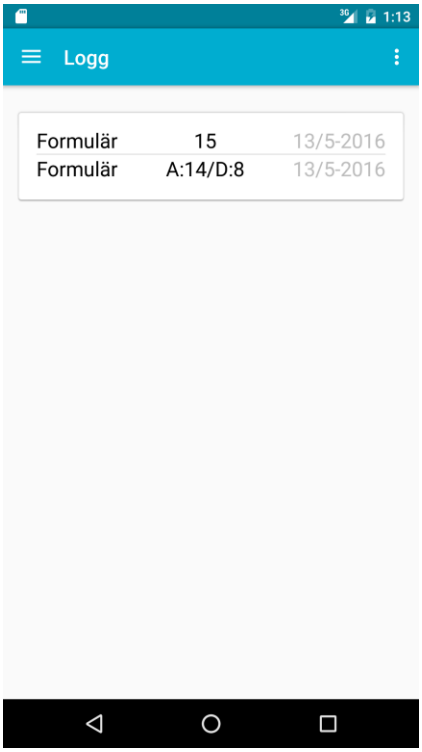


Startvyn för att svara på astmatestet i mobil-enheten

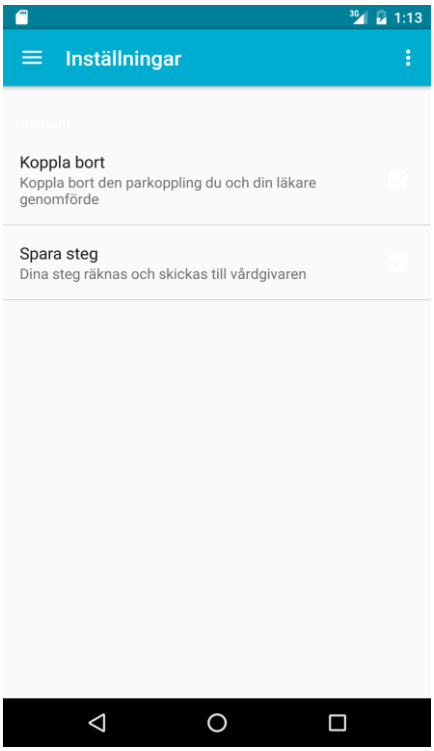
Översiktsvyn i mobil-applikationen



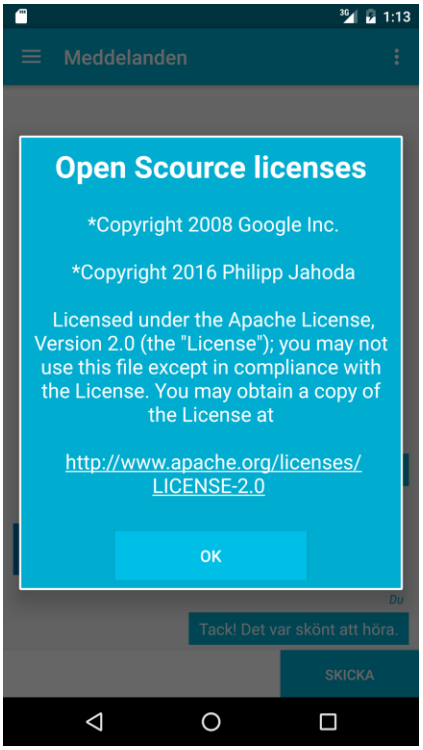
Vyn för meddelandefunktionen i mobil-applikationen



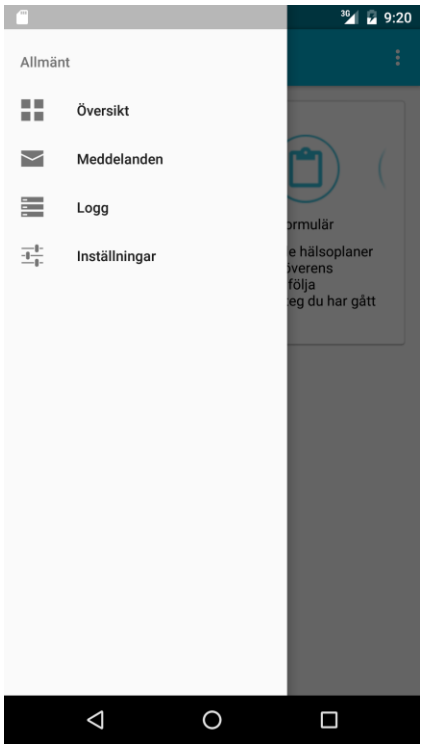
Vyn för logg som visar data som samlats in på mobil-applikationen



Vyn för inställningar i mobil-applikationen



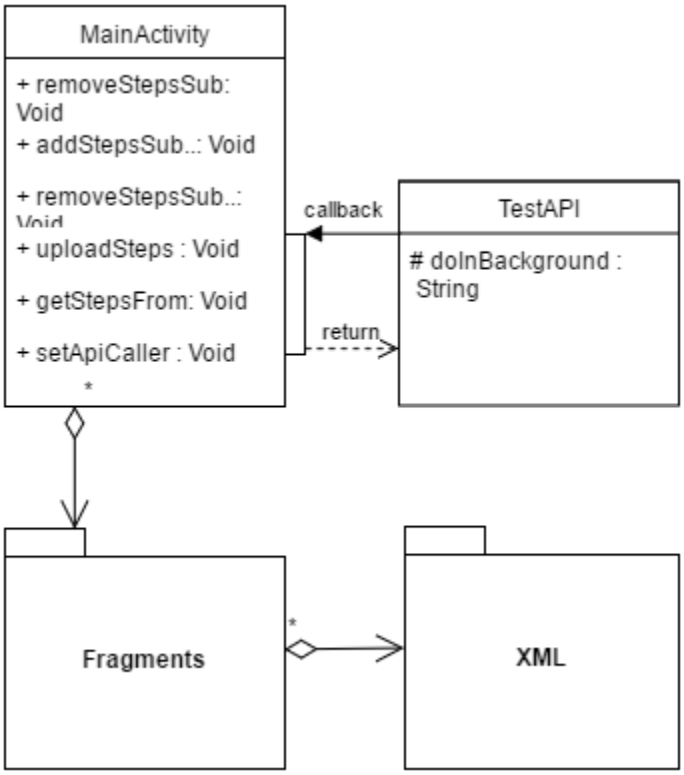
Dialogruta som visar använda licenser vid utveckling av mobil-applikationen



Navigationsmenyn i mobil-applikationen

```
final CheckBox qa11 = (CheckBox) v.findViewById(R.id.qa91);
   final CheckBox qa12 = (CheckBox) v.findViewById(R.id.qa92);

   if(bundle != null){
       if(bundle.getInt("q9") == 4){qa11.toggle();}
       else if(bundle.getInt("q9") == 3){qa12.toggle();}
     }
   }
   qa11.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {
           if(qa11.isChecked()){
               if(qa12.isChecked()){qa12.toggle();}
           }
       }
   });
   qa12.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {
           if(qa12.isChecked()){
               if(qa11.isChecked()){qa11.toggle();}
           }
       }
   });
   backBtn.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {
           ADFormQ8Fragment adFormQ8Fragment = new ADFormQ8Fragment();
           adFormQ8Fragment.setArguments(bundle);
           FragmentManager fragmentManager = getFragmentManager();
           FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
           fragmentTransaction.setCustomAnimations(R.animator.fade_in_from_left,
                   R.animator.fade_out_to_right);
           fragmentTransaction.add(R.id.form_rel_layout, adFormQ8Fragment, "Question 8 fragment
                   Active");
           fragmentTransaction.addToBackStack(null);
           fragmentTransaction.commit();
       }
   });
   nextBtn.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {

           if(!qa11.isChecked() && !qa12.isChecked()){
               showErrorOnNextDialog();
           }else{
               if(qa11.isChecked()){bundle.putInt("q9",4); bundle.putString("qt9","Väldigt ofta");}
               if(qa12.isChecked()){bundle.putInt("q9",3); bundle.putString("qt9","Ganska ofta");}
               ADFormQ10Fragment adFormQ10Fragment = new ADFormQ10Fragment();
               adFormQ10Fragment.setArguments(bundle);
               FragmentManager fragmentManager = getFragmentManager();
               FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
               fragmentTransaction.setCustomAnimations(R.animator.fade_in_forms,
                       R.animator.fade_out_forms);
               fragmentTransaction.add(R.id.form_rel_layout, adFormQ10Fragment, "Question 10
                       fragment Active");
               fragmentTransaction.addToBackStack(null);
               fragmentTransaction.commit();
           }
       }
   });
```

Fragmentklass för en formulärfråga, knappar hämtas från en XML-layout och lyssnar läggs till. Lyssnaren för ”nextBtn” visas hur bundle:s används för att skicka info mellan olika fragment.



Paketdiagram av mobilapplikationen där MainActivity representerar kontroller i MVC samt där Fragments och XML representerar modell respektive vy. TestAPI är den hjälpklass som används för att skicka och ta emot data från API:et.

Androidapplikation som följer Material Design samt Praktikertjänsts grafiska profil

- Användare skall kunna koppla enheten till vårdgivaren
- Användaren skall kunna fylla i astmaformulär och få ett resultat givet
- Användaren skall kunna fylla i ångest & depressions formulär och få ett resultat givet
- Användaren skall kunna godkänna att enheten samlar in steg och kunna se dessa grafiskt
- Användaren skall kunna kontakta sin vårdgivare genom att skicka direktmeddelanden i applikationen
- Användaren skall kunna sluta sin koppling samt inskickning av data

Webbapplikation som följer Praktikertjänsts grafiska profil

- Användaren skall kunna se alla kopplade enheter som en vårdtagare har
- Användaren skall kunna lägga till nya enheter och få en given kod som kan ges till vårdtagare
- Användare skall kunna ta bort en enhet ur systemet
- Användare skall kunna se en vårdtagares sammanlagda stegdata från alla enheter grafiskt
- Användare skall kunna se en vårdtagares sammanlagda formulärpoäng samt svar från alla enheter grafiskt
- Användare skall kunna kontakta vårdtagare genom direktmeddelanden