CHALMERS

Virtuella enheter av hårdvara från Plejd

En studie i möjligheten att skapa ett virtuellt system

Examensarbete inom Data- och Informationsteknik

Johan Ericsson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2020

EXAMENSARBETE

Virtuella enheter av Hårdvara från Plejd

En undersökning om möjligheten att skapa ett virtuellt system som speglar hård och mjukvara i en existerande produkt

Johan Ericsson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2020

Virtuella enheter av Hårdvara från Plejd
En undersökning om möjligheten att skapa ett virtuellt system som speglar hård och mjukvara i en existerande produkt

Johan Ericsson

Examinator: Jonas Duregård

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag:
Pressbild från Plejd

Institutionen för Data- och Informationsteknik
Göteborg 2016

SAMMANFATTNING

Plejd är ett företag inom hemautomation som har sitt största fokus på smart belysning. Deras populäraste produkter är hårdvaru enheter som installeras i väggen som sedan kan använda för att styra lampor och andra laster via bluetooth eller över internet. Målet med detta projekt var att undersöka och implementera ett virtuellt system som syftar att efterlikna befintligt system av mjukvara och hårdvara från Plejd. Ett virtuellt system skulle potentiellt kunna förbättra och snabba på flera områden hos företaget.Denna rapport beskriver metod och tekniker som använts i utvecklingen av systemet.

Resultatet blev ett fungerande virtuellt system som kan göra en förenklad simulering av fysiska enheter med en modifierad version av Plejds app kan kommunicera med en websocket server i samma nätverk.

Nyckelord: Internet of things, Plejd, Virtual, Websocket, server, klient

Abstract

Plejd is a home-automation company with its main focus being intelligent lighting. Their most popular product is hardware devices which can be installed in the wall and then control lamps and other loads via bluetooth or over the internet. The goal of this project was to investigate and implement a virtual system that aims to mimic existing systems of software and hardware from Plejd. This report describes the method and techniques used in the development of the system.

The result was a working virtual system that with a modified version of Plejd's app can communicate with a websocket server on the same network.

Keywords: Internet of things, Plejd, Virtual, Websocket, server, klient

FÖRORD

Detta examensarbete omfattar 15 högskolepoäng och skrevs på Institutionen för datateknik på Chalmers tekniska högskola under våren 2020. Det utfördes av Johan Ericsson som går dataingenjörsprogrammet(senare del) 180hp, på Chalmers tekniska högskola.

Stort tack till alla inblandade, speciellt min handledare Herman Rödström och Magnus Ekberg som varit ovärderliga tillgångar när jag behövt information, tips eller bara bolla idéer genom projektets gång.

Terminologi

C#	Ett objektorienterat programspråk som är en del i .NET plattformen.
Xamarin	Ramverk för utveckling av gemensam kod för iOS och Android
GUI	Graphical User Interface, ett grafiskt användargränsnitt.
CLI	Command Line Interface, ett användargränssnitt där användaren använder en terminal eller konsol för att styra ett program eller system.
ACK	Förkortning för Acknowledgement. ACK används t.ex när en part av en sammankoppling vill meddela den andra parten att denna mottagit ett meddelande.
Full-duplex	Benämning för trafik som kan flöda obehindrat i två riktningar.
Open source	Öppen källkod som kan läsas och användas fritt
JSON	JavaScript Object Notation, ett textbaserat format för att representera dataobjekt.
XML	Extensible Markup Language, liksom JSON ett format för att representera data.

Table of Contents

1 Inledning

I detta kapitel beskrivs bakgrund, syfte och mål med projektet och avgränsningar samt frågeställningar behandlas.

1.1 Bakgrund

Plejd är ett företag inom hemautomation som har sitt största fokus på smart belysning. Deras populäraste produkter är hårdvaru enheter som installeras i väggen som sedan kan användas för att styra lampor och andra laster med en mobil-applikation via bluetooth eller över internet. Testning och utveckling av systemet är beroende av de fysiska enheterna och detta medför vissa begränsningar. T.ex. så måste mycket av utvecklingen ske ihop med test-hårdvaran vilket begränsar utvecklarnas flexibilitet och möjlighet att jobba remote.

1.2 Syfte

Att utforska om det går att skapa virtuella enheter som fungerar ihop med nuvarande mobilapplikation på ett sätt som speglar de fysiska enheterna. Det kan potentiellt finnas många fördelar med virtuella enheter gentemot fysiska ur ett utvecklar- och organisationsperspektiv. Det kan bli mer praktiskt, slippa att ta med och installera något fysiskt för utvecklarna och testarna på företaget, det enda som då skulle behövas är en dator till skillnad mot nuläget som kräver att testhårdvara finns på plats. Det kan även gå snabbare och vara mer ekonomiskt då det tar tid och resurser att bygga den hårdvara som behövs för tester och utveckling idag.

1.3 Mål

Projektets mål är att skapa ett virtuellt system som liknar Plejds fysiska system och fungerar ihop med Plejds mobil applikation på samma sätt som de fysiska enheterna gör.

1.4 Avgränsningar

För att arbetet skall kunna fortskrida så fort som möjligt inriktar jag mig endast på Android i ett första skede och portar koden till iOS i mån om tid.
Xamarin tillåter för enkel kod-delning mellan android och iOS, vilket gör att man kan skriva en gemensam kodbas som fungerar på båda operativsystemen. Dock går detta ej att göra fullt ut då vissa lösningar måste delas upp t.ex. Kan man ej jobba med websockets direkt i Xamarin på ett sätt som fungerar enkelt på både android och iOS.
Att simulera en hårdvara så komplex som Plejds puckar fullt ut är ett arbete som ligger utanför vad som kan förväntas av ett enskilt examensarbete. Det jag ämnar göra är att bygga en bra grund att stå på som fungerar och enkelt kan byggas på med mer funktionalitet om så önskas.

1.5 Frågeställningar

Inför arbetet så fanns det flera frågor som ämnade få svar efter slutfört arbete.

- Kan man skapa en fungerande virtuell miljö som fungerar med appen?
    - Kan man skapa virtuella enheter som appen hitta, installera och kommunicera med?
    - Hur skall de kommunicera?
    - Behövs extra mjukvara? Och isåfall vad?
    - Kan man installera dem så att de känns igen och minns av systemet
    - Kan man få dem ge liknande respons som fysiska enheter som appen kan plocka upp.
    - Kan man i en virtuell miljö skapa funktionalitet som ej skulle vara möjligt eller opraktiskt med fysiska enheter?
    - Kan man skapa en server för att erhålla den virtuella miljön?
    - Kan man få en eller flera klienter att ansluta och erhålla serverns miljö?
    - Vilken typ av kommunikationsprotokoll är lämplig för en sådan server?
- Kan en virtuell miljö vara till nytta för utvecklarna/företaget? Rent praktiskt? Ekonomiskt? Utvecklingsmässigt?
- Kan man få lika mycket funktionalitet från virtuella puckar som de fysiska?

2 Metod

I ett inledande skede så planeras projektet ihop med handledare på Plejd. Vi diskuterar tekniska lösningar, ramverk och sätter upp mål för vad som förväntas ingå i projektet. Därefter sätts en utvecklingsmiljö upp med inklusive git, trello och IDE och projektet kan påbörjas.
Projektet bör brytas ned i mindre delmoment. Ett första kommer vara att implementera en fungerande server och klient koppling och utforska ramverk. Detta kan i ett första steg göras som en egen Android applikation utanför Plejds egen mjukvara för att skala av och ta bort eventuella svårigheter som tillkommer när appen sedan skall modifieras. När server-klient koppling fungerar mellan Android och server så finns kunskapen om att systemet fungerar och nästa steg blir att implementera den funktionaliteten i Plejd-applikationen.

Implementationen av virtuella enheter kommer kräva att existerande hårdvara granskas och efterliknas. Detta kommer göras genom att debugga och stega igenom koden för Appen tillsammans med testhårdvara och försöka se hur app och hårdvara interagerar. Debuggingen kan med fördel varvas av egna test av virtuella enheter vilket även gör det lätt att jämföra de båda systemen och felsökning kan göras genom att se hur de fysiska enheterna beter sig där de virtuella kraschar.

Vid tidpunkten då server och klient kan kommunicera och virtuella enheter kan skapas på begäran och upptäckas av mobil-applikationen har målet för projektet nåtts och ytterligare funktionalitet byggas på. Här handlar det om att efterlikna befintligt system i så stor utsträckning som möjligt och se att de virtuella enheterna kan bete sig och reagera som de fysiska vid olika moment. Man kan även bygga ut server med ytterligare funktionalitet som t.ex. möjlighet för flera klienter att ansluta och interagera med samma system. Diskussion om funktionalitet för systemet och prioriteringar kan även diskuteras med Plejds utvecklingsteam här för att se att de påbyggnationer som görs är relevanta.

3 Kravspecifikation

Appen som utvecklas hos Plejd är skriven i C# med plattformen Xamarin, Microsofts Visual Studio och Jetbrains Ryder var de IDEs som utvecklarna använde. Av dessa valdes Visual Studio för projektet.

Krav att appen och servern skulle kunna kommunicera om de var anslutna till samma nätverk.

Krav att miljön skulle efterlikna den verkliga miljön. Vid användande av appen skulle den fungera i stort sett som vanligt, fast med möjlighet att hitta, installera och använda virtuella enheter istället för fysiska.

Önskemål att servern skulle kunna hantera flera klienter.

Eftersom appen är skriven i C# så var ett önskemål att även servern skulle skrivas i detta språk.

Efterhand uppkom det även ett önskemål om en CLI-version av servern, möjlighet att generera många enheter på samma gång och möjlighet att generera enheter med specifika parametrar.

Önskemål att appen skulle förbli vara så oförändrad som möjligt, ytterligare kod kommer förstås att behöva adderas men befintlig kod ska i så liten utsträckning som möjligt editeras.

4 Teknisk Bakgrund

I det här kapitlet beskrivs idéer, tekniker och verktyg som använts för att implementera och efterlikna de olika komponenterna som har varit nödvändiga för att få fram en fungerande produkt.

4.1 Simulering

Simulering är en vanlig teknik för att testa och utveckla system och miljöer som skulle vara dyrt, farligt eller opassande att göra på riktigt. I detta arbete kommer ett system inkluderande mobilapplikation och hårdvaruenheter representeras med hjälp av mobilapplikation och server.

Simulering används idag inom en rad områden och fördelarna med att kunna simulera system är många[1 ny]. Några positiva aspekter av simulering inbegriper:

- Riskfri miljö - En miljö som skulle vara farlig för människor, natur eller utrustning kan riskfritt simuleras och på så vis eliminera risken med den fysiska miljön.
- Sparar tid - Vid t.ex. omorganisation av fabriker så kan driftstopp vara kostsamma och en simulerad miljö kan snabba upp processen då man redan testat kommande uppställning och kunnat planera för att utföra omställningen så snabbt som möjligt.
- Sparar pengar
- Möjlig visualisering
- Bättre nogrannhet

Simulering är vitt tillämpningsbart och kan potentiellt även användas för att förutspå konsekvenserna av klimatförändringar[2 klimatsim], simulera t.ex. molekyler inom bioteknik[3 biotech] och göra tester som skulle vara dyra att göra rent praktiskt, t.ex. kraschtest med bilar[4 carcrashsim].

Med en konstant utveckling av datorer och mjukvara så kommer det finnas det möjlighet att simulera fler och fler områden i framtiden och även på en djupare mer realistisk nivå.

4.2 Emulering

I datasammanhang så är en emulator en hårdvara eller mjukvara som tillåter ett datasystem att bete sig som ett annat system[5 emulering]. Ju bättre emuleringen desto mer verklighetstroget kan det emulerade systemet fungera. Till skillnad från en simulering som syftar att efterlikna ett systems beteende syftar emulering att efterlikna själva hårdvaran av systemet. T.ex. kan en simulation av en bil vara väl fungerande om bilen kan accelerera, bromsa, svänga, tuta mm. men en emulering skulle ta i beaktande

hur hela bilen fungerade och på detaljnivå efterlikna förgasare, bormssystem, startmotor o.s.v. Att emulera ett system kan därför bli mycket mer komplex.

4.1 Simulering kontra Emulering

Skillnaden mellan simulator och emulator är som skrivet ovan att en simulator endast simulerar eller efterliknar ett önskat beteende från en specifik enhet och en emulator kopierar varje aspekt av hårdvaran den syftar att efterlikna[6]. Efter egen reflektion och diskussion med handledare gjordes bedömningen att en simulerad miljö skulle vara tillräckligt för att tillgodose kraven för projektet och att en implementation av en emulerad hårdvara skulle bli onödigt komplicerad i ett första steg. En emulering av systemet var dock inte uteslutet och något som skulle kunna bli aktuellt i framtiden. I ett emulerat system skulle t.ex. enheternas Firmware kunna installeras, testas och debuggas.

4.1.1 Emuleringens för och nackdelar

En av Emuleringens fördelar när det gäller produktutveckling är att ju mer verklighetstroget man kan avbilda systemet desto bättre bild kan man skapa sig av den produkt man utvecklar. Tester, problem och begränsningar blir mer verklighetstrogna och enklare att utföra/upptäcka.
Om information för det system som skall emuleras finns tillgänglig så finns det även en mall på vad som måste utvecklas.

En nackdel med emulering är att det är tidskrävande att utveckla eftersom varje del av systemet måste imiteras inklusive cpu, ramminne och kommunikationsenheter.

Ytterligare en nackdel för större system är att systemet som kör emuleringen måste vara väsentligt snabbare än systemet som det ämnar emulera för att klara av att hantera både sitt eget system och det emulerade systemet samtidigt.

4.1.2 Simuleringens för och nackdelar

En fördel med simulering är att det kan göras godtyckligt enkel till en början och sedan byggas på och förbättras. Detta gör att det möjligt att få fram resultat snabbt och iterativt göra förbättringar och förändringar utifrån nuvarande version av systemet.

En annan fördel är att det inte kräver lika mycket detaljkunskap om enheterna som skall simuleras då endast beteendet behöver efterliknas.

En nackdel är att simuleringen ofta förblir en abstraherad version av verkligheten. När man bara efterliknar betendet hos hårdvara så kan det hända att systemet ej tar hänsyn till viktiga testfall. T.ex. kan hårdvaran ha en låg minneskapacitet eller beräkningsförmåga vilket ger den begränsningar som inte klarar av vissa instruktioner. Vid testning mot en simulerad miljö finns det risk att sådana begränsningar missats och tester som skulle ha fått negativt utfall går igenom. Detta kan i sin tur leda till att man utvecklar mjukvara som ej kommer att fungera på de fysiska enheterna.

4.2 Websockets

Websockets är ett protokoll som möjliggör full duplex över en TCP-socket. Websockets har full duplex, kryptering och efter handskakningprocessen finns en “stream” som man kan kommunicera via med låg overhead.

4.3 Xamarin

Xamarin är ett ramverk som förlänger utvecklingsplattformen .NET. Ramverket gör det möjligt för android och iOS att dela kodbas och att utveckla i ett enda språk(C#) i Windows eller MacOs.[7]
Xamarin gör det även möjligt att utveckla macOS applikationer(Cocoa) i C# istället för att använda Objective-C eller Swift.

4.4 Cocoa

Cocoa är en mängd ramverk framtagna för utveckling för OSX och iOS [8]. Visualstudio erbjuder en utvecklingsmiljö för Cocoa där koden kan skrivas i VS och det grafiska gränssnittet kan utvecklas i Xcodes Interface Builder.

4.5 .NET Core

.NET Core är en open source och general-purpose utvecklingsplattform där man kan utveckla .NET Core applikationer för windows, macOS och Linux.[9]

4.6 Programvaror

Överblick av de programvaror som använts under arbetets gång. Utöver nedanstående programvaror så har slack och google använts för kommunikation, trello för överblick av uppgifter och github för versionshantering.

4.6.1 Visual studio

Visual Studio är en integrerad utvecklingsmiljö (IDE) från Microsoft. Det kan användas bland annat för att utveckla datorprogram och mobilappar.[10]

4.6.2 XCode

XCode är en integrerad utvecklingsmiljö från apple som kan användas för att utveckla applikationer för apples produkter. [11]

4.6.3 Interface Builder

Interface Builder är en editor som kommer förinstallerad med XCode som förenklar processen med att designa och skapa användargränssnitt genom drag and drop funktioner för objekt.[12]

5 Genomförande

Arbetet med att ta fram virtuella enheter har haft olika faser, i en första fas så samlades kunskap om utvecklingsverktyg, språk, nuvarande kod och hur fysiska enheter fungerar. I en andra fas så utformades och testades en plan för att efterlikna det befintliga systemet fast med virtuella enheter och websockets som komponenter och kommunikation iställer för fysiska enheter och bluetooth.
I en tredje fas så fungerar all grundläggande kommunikation och programvara för att enkelt efterlikna ett riktigt system och funktionalitet kan byggas på.

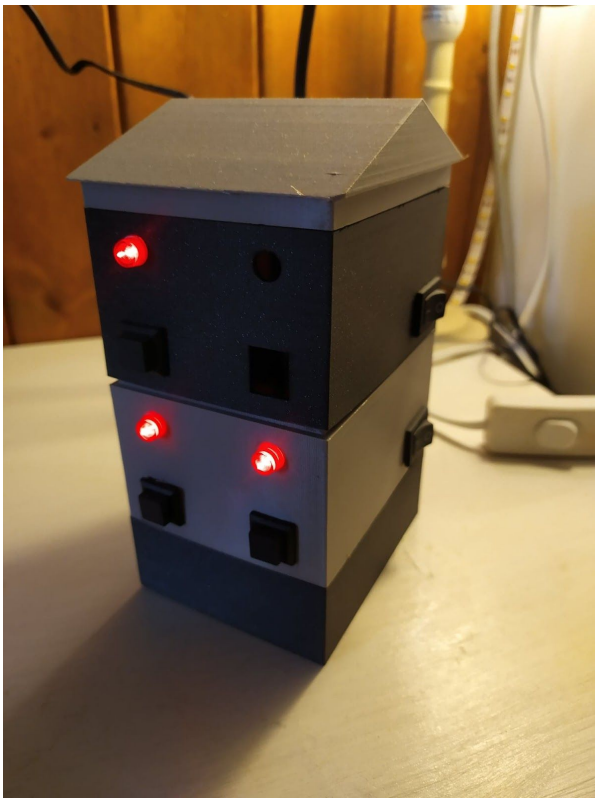
5.1 Uppstart

Vid uppstartsmöte diskuterades projektet och ideér ventilerades. Vi diskuterade möjligheter och svårigheter med simulering och emulering och kom fram till att det det mest realistiska som skulle kunna ge resultat snabbt var att simulera hårdvaran och bygga den på ett sådant sätt att mer funktionalitet lätt kan implementeras och i framtiden kanske ersättas av ett emulerat system.

Beslut att bygga en server som kan kommunicera med appen över nätverket via websockets gjordes. Vi diskuterade även om det var en bra idé att kommunicera med servern via bluetooth, då det är så de fysiska puckarna kommunicerar, men kom fram till att det skulle antagligen skapa onödigt många problem.

Appteamet hos Plejd utvecklar mestadels i C# och ett önskemål var att servern även den skulle skrivas i detta språk. Jag hade jobbat litegrann med C# innan och tyckte det var en bra idé. Speciellt eftersom appen i övrigt är skriven i C# så blir det enkelt att rikta in sig på endast ett språk.

Efter uppstartsmötet så försåg Plejd mig med ett sk torn, se figur 1. Det är ett litet 3d-printat hus som innehåller fysiska enheter, framtagen för att kunna användas bredvid datorn när man utvecklar.



Figur 1. Test-torn med 2 olika plejd enheter inuti.

5.2 Skapa en server

Servern skapades i två steg, först valdes teknik sedan gjordes test och implementationer av tekniken.

5.1.1 Val av teknik

Under uppstartsmöte så diksuterades tekniker för anslutning mellan mobil-applikation och server. Valet blev websockets då protokollet ansågs lämpligt för projektet.
Tekniker som togs i beaktande var även Bluetooth och TCP. Websockets använder TCP kommunikation i implementationen, men med websockets protokoll. Detta sågs som en fördel framför att skriva endast i TCP då det kan portas om och köras med andra tekniker om så önskas. Websockets är även en välkänd teknik som är relativt enkelt att hitta information om, detta är fördelaktigt vid vidareutveckling av annan part.
Bluetooth var aktuell då det är så appen kommunicerar med de fysiska enheterna vanligtvis. Förhoppningen med att välja bluetooth var att kommunikationen från appen skulle behöva minimal modifikation. Efter diskussion med handledare och delar från app teamet kom vi dock fram till att möjligheterna blir större och implementationen enklare ifall vi använder websockets. Möjligheten att koppla upp sig om man är ansluten till ett nätverk ger bättre anslutningsmöjligheter än bluetooth, lättare att behandla många anslutningar, inget krav på att servern använder bluetooth, mer tillgänglig information sågs som argument att välja websockets över bluetooth.
Så fördelarna med websockets är sammanfattningsvis[13]

- Full duplex

- Väl använd teknik
- Öppen ström
- Minimal overhead - hastighet

5.1.2 Implementation av server

Efter uppstarten så var det naturligt att utforska och implementera en websocket server. Det finns mycket information och kod att tillgå på internet och inom kort så hade jag en server som fungerade ihop med en webklient. Eftersom jag utvecklar i C# på mac så skapade jag ett Cocoa projekt och gjorde ett enkelt GUI till servern som i första steget var en chatt-app där klienter kunde ansluta och man kunde skicka meddelanden till varann.
Nästa steg blev att skapa en klient i C#, detta följde naturligt då Plejds app senare kommer behöva en klient för att kommunicera med servern, men jag ville först isolera och förstå hur man skriver en websocket klient separat i C#. Detta visade sig vara förhållandevis enkelt tack vare mycket tillgänglig information och Microsofts System.Net ramverk.

5.3 Förstå Plejd Appen

När grundläggande funktionalitet fanns på plats för servern och den information som krävdes för att skapa en server-klient koppling hade samlats in blev nästa steg att förstå hur Plejds App fungerar, hur de fysiska enheterna fungerar och deras symbios.

5.4 Virtuell Miljö inuti Appen

Efter jag testat att stega mig igenom installations- och kommunikationsprocesser så började jag testa att modifiera appen. Inuti appen finns en adapter som fungerar som ett nav i kommunikationen. Jag började att byta ut den mot min egen och började experimentera med att skapa egna enheter. Arbetsflödet blev

- skriva kod
- testa
- Debugga
- se vart det blev fel
- byta till de fysiska enheterna
- granska hur de beter sig
- efterlikna det beteendet

Detta var en långdragen process. Dels så tog projektet mellan 1-2 minuter att bygga om (något som appteamet sedan hjälpte mig få ner till 20 sek), dels så fanns det vissa svårigheter med att debugga de fysiska enheterna. Den första svårigheten var att det ej gick att kolla på alla variabler vid breakpoints. Detta kommer sig av att många klasser har “getters” som utför en metod när de blir kallade på, när man tittar på sina lokala variabler och objekt vid breakpoints så står exekveringen still och ingen metod kallas, detta i sin tur gör att visual studios debugger kraschar för att den ej kan få tag i

variablerna. Lösningen fick bli att printa ut intressant information. En annan svårighet är att det finns flera timeouts i koden. Detta gör att om man sätter en breakpoint i ett tids-kritiskt kodblock och debuggar långsammare än time-outen (som ofta är under en sekund) så kommer det förväntade utfallet ej att ske.
I vissa fall kunde printouts lösa problemen, i andra fall var det smidigare att editera om koden och tillfälligt ta bort timers.

I appen finns många olika sorters interface för att hantera enheter, och det framkom 3 stycken av dessa behövde implementeras. Test gjordes med att göra olika virtuella enheter för var och en men den smidigaste lösningen blev att slå samman alla nödvändiga interface till en klass “VirtualDevice” som kunde hantera alla nödvändiga moment. Detta fungerade och gjorde koden mer överskådlig.

Därefter skapades virtuella enheter och en sökningsprocess simulerades från adaptern. Då koden fungerade tillräckligt bra för att skapa enheter och visa dem vid en sökning för nya enheter så var nästa steg att låta dem genomgå en installationsprocess. Här framkom problem och brister som fann i de virtuella enheterna och det blev en lång process att identifiera och lösa problem som förhindrade installation.

5.5 Virtuell Miljö ihop med server

Då appen accepterade de virtuella enheterna var nästa steg att ej generera dem inuti appen utan via en server så att appen kunde fungera mer som vanligt, med skillnaden att sökning och kommunikation skedde via websockets och virtuella enheter istället för bluetooth och fysiska enheter.

Då kommunikation redan upprättats var frågeställningen hur enheterna skulle skickas. Lösningen blev att serialisera en lista med virtuella enheter som kunde skickas via websockets och deserialiseras av appen. Det är i teorin en enkel process men vissa svårigheter fanns då serialiseringen ej gick att genomföra med inbyggda funktioner i C#.

Test av olika paket och serialiseringsmetoder genomfördes och Newtonsoft.Json var det ramverk som fungerade enklast och det som gav önskade resultat först. Det fungerade också bra ihop med Xamarin. Test gjordes även av av microsofts egna Json och XML serialisering bibliotek, dessa fungerar på liknande sätt som Newtonsofts ramverk och valet av Newtonsoft var subjektivt.

Serialiseringen fungerade inte rakt av utav de virtuella enheterna och vidare undersökning av serialiseringen blev nödvändig. Det framkom att serialiseringen sker på sådant sätt att en json-array skapas innehållande json-objekt som avbildar de virtuella enheterna. Detta var som förväntat, problemen som uppkom skedde vid deserialiseringen utav de virtuella enheterna. Det visar sig att deserialiseringen som paket gör vill ha en tom konstruktor för att återskapa objekten och därefter kräver den att alla variabler är publika för att kunna sätta variablerna till sina rätta värden. Detta gjorde att “VirtualDevice” klassen behövde editeras för att passa deserialiseringen.

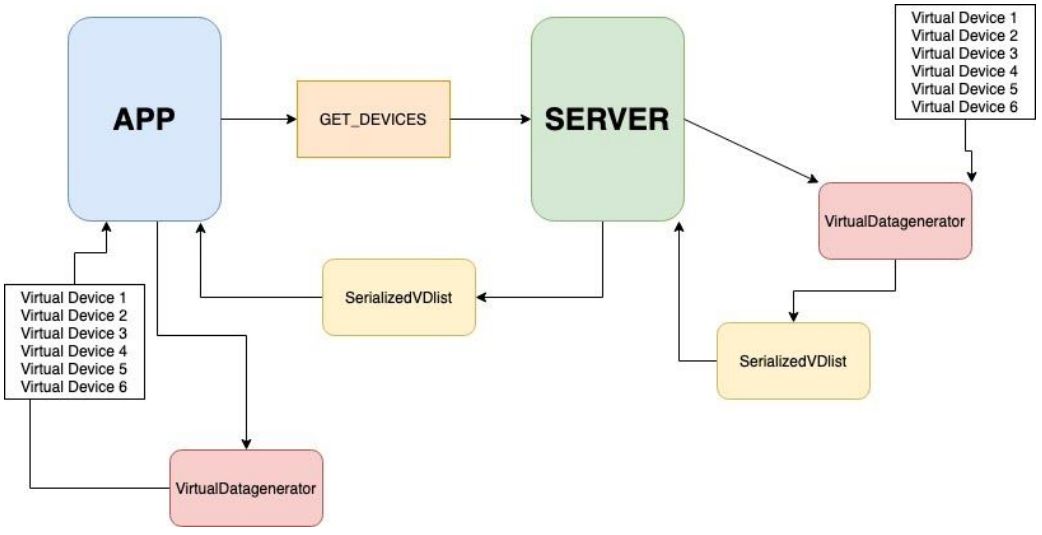
Efter att serialisering och deserialisering fungerade på servern så var listan av enheter redo att serialiseras och skickas.

När kommunikationen skulle upprättas visade det sig att Xamarin ej stödjer System.Net vilket förvanskligade websocket-kommunikationen. Efter att sökt information och testat ramverket ASP.NET SignalR och WebsocketClientLite.PCL utan fungerande resultat drogs slutsatsen att det kommer vara mer tidseffektivt att utveckla specifika lösningar till Android och iOS. Lösningen blev således att använda System.Net och skriva koden på ett sådant sätt att websocket klienten endast genereras av android versionen av appen, där stöd för System.Net finns.

När valet gjorts att använda microsofts egna ramverk så gick kommunikationen enkelt, detta då jag i tidigare test-applikationer gjort liknande implementationer med samma ramverk.

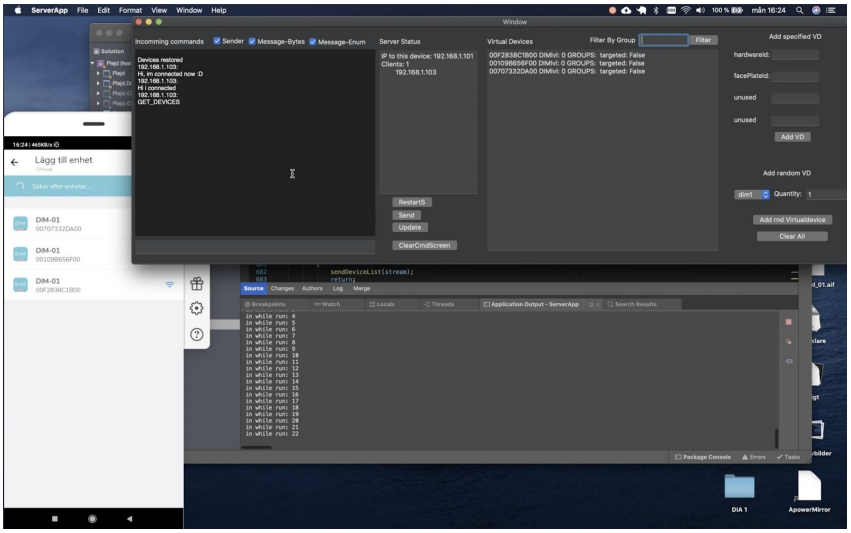
Koden som tar hand om socketen behövde skrivas generiskt så att lämplig tcp-klient kunde sättas beroende på om android eller iOS versionen startas. Här hittade jag att plejd redan gjort en liknande lösning för sin kommunikation med gateway och jag kunde återanvända deras "IClientWebSocket" interface som var framtagen för att lösa liknande problem. Problemet med socketen är att det ej finns en tcp-klient som går att använda i Xamarin då iOS och Android behöver använda sina egna ramverk för tcp-kommunikation. Lösningen blev då att skriva en ny klient som hade en "IClientWebsocket" som den satte till Appens websocket vilket i sin tur initieras när man skapar applikationen och således kan man skapa önskad socket för android och iOS.

I servern så implementerades en funktion för att skapa virtuella enheter och stöd för att serialisera dessa och möjlighet att skicka dem som ett textmeddelande till ansluten klient, se figur 2.



Figur 2. Servers respons till kommandot GET_DEVICES, en serialiserad lista av virtuella enheter skickas till appen där de kan deserialiseras och upptäckas.

När kommunikationen väl var upprättad gick det att byta ut tidigare simulerade sökning som skett inifrån appen med att skicka en förfrågan till servern om att få enheter, varpå servern serialiserade sin genererade lista, skickade över denna till appen, där den deserialiserades och hanterades i enlighet med redan implementerad funktionalitet för att upptäcka enheter, se figur 3.



Figur 3. Server som genererat 3 enheter, accepterat en connection från app och svarat på ett skanningsmeddelande.

5.6 Installera enheter

Plejds app fungerar på så sätt att enheter kan upptäckas, därefter installeras och slutligen användas eller raderas. Efter server och app fått kontakt och kunde utbyta information i forma av virtuell-enhetsinfo (se Figur 3) var nästa naturliga steg att installera enheterna. Detta var på många sätt mer krävande än själva dataöverföringen då många steg måste fungera i installationsprocessen. Först så laddas information upp till målet, därefter sätts kryptering på enheten, därefter skickas inställningar om t.ex. in och utgångar för enheten från appen. Appen i sin tur vet ej om att den kommunicerar med en server istället för en fysisk enhet och förväntar sig därför svar som i vanlig ordning bekräftar de kommandon som den skickat ut. Vid ett första test så verkade det som att appen förväntade sig identiska meddelanden tillbaka, men det fungerade ej för alla kommandon. Efter undersökning och diskussion med handledare så framkom att kommandon skickas i byte-arrayer, där de första fem byten innehåller overhead och eventuellt ytterligare bytes innehåller payload. I overheaden finns en flagga som visar huruvida appen förväntar sig ett “acknowledgement” eller ej. Flaggan kallas för “do not respond”.

Ett första angreppssätt var att titta på alla kommandon som skickas och se huruvida flaggan “don´t respond” var satt eller ej och sedan lägga in de kommandona som ville ha svar (där flaggan ej var satt) i en lista över kommandon som förväntade sig “acknowledgement”. Detta var en tidsödande metod som krävde att alla kommandon som krävde ett ACK las till listan och specialbehandlades och för varje nytt kommando som krävde svar så krashade programmet, varpå kommandot kunde avläsas i konsolen,

läggas till i listan och sedan fick programmet byggas om och köras till nästa krasch. För att slippa dessa steg så testades att svara med den simpla regeln att om “do not respond” flaggan var satt, svara med identiskt meddelande, annars svara med ett ack. Detta fungerade hela installationsprocessen och är den logik som finns för svar i slutgiltiga versionen. Det kom som en överraskning för både mig och handledare att även när “do not respond” flaggan var satt så behövdes ett svar.

5.7 Kommunikation med specifika enheter

När enheter kunnat installeras på appen och servern kunde svara som förväntat på appens kommandon så blev det tydligt att de enkla kommunikationslösningarna hade kommit på bekostnad av livlösa enheter. Oavsett vilka kommandon som appen skickade ut så svarade servern med “okej”. Vilket gjorde att appen trodde att alla inställningar gjorts på enheten fast ingen kontakt med enheten gjorts. T.ex. så gick det att ändra dimningsnivå, sätta på och av enheter inuti appen, men de virtuella enheterna ändrade inget tillstånd. Nästa steg var att låta de virtuella enheterna reagera på kommandon. Vid efterliknelse av det fysiska systemet så skulle enheterna stå för all kommunikation, detta skulle betyda att processen måste göras om och alla kommandon som skickas i installationsprocessen behöver noggrant implementeras. Detta skulle vara ett arbetssätt som skulle ta mycket tid och ej leda fram till en fungerande produkt inom tidsramen för arbetet. Lösningen blev istället att låta serverns respons-logik ligga kvar, men att ge de virtuella enheterna en lista med kommandon som de stödde. När servern sedan mottar ett kommando så tittar den i listan över vilka kommandon som de virtuella enheterna stödjer och som kommandot stöds så vidarebefordras kommandot.
Stöd gavs för tre kommandon att sätta id och grupper, att sätta dimningsnivå och att stänga av / sätta på enheten.
Första byten i ett kommando är enhetens id inom systemet, detta id är ej samma som själva enhetens id utan sätts vid installationsprocessen. För att kunna skicka meddelanden till specifika enheter var det första kommandot som måste implementeras att kunna sätta detta id:et under installationen.
För att göra detta så skickas kommando till ansluten enhet i det fysiska systemet. Slutsatsen drogs att en bra väg att gå är att efterlikna detta i det virtuella systemet, därmed implementerades stöd för att koppla upp sig mot virtuella enheter.
När detta var löst så kunde man enkelt lägga till grupper för de virtuella enheterna och sedan filtrera inkommande kommandon efter dessa.
Ett kommando vidarebefordras då till enheten om två villkor uppfylls:

- kommandot stöds av de virtuella enheterna
- kommandot är adresserat till enhetens id eller till en grupp där enheten ingår.

5.8 Multiconnection

Ett önskemål från appteamet var möjligheten att koppla upp flera enheter till servern. I första implementationen så kördes servern i en tråd och kunde endast hantera en klient.

Jag hade två ideér på hur man skulle kunna skriva om servern för att ge stöd åt multipla klienter. Den första iden var att köra 2 trådar där den ena väntar på nya klienter och hanterar dem genom att efter handskakning lägga dem i en lista medans den andra tråden cirkulerar bland de anslutna enheterna och hanterar dem en efter en ifall de har data de vill skriva på socketen. Den andra idéen var att en tråd välkomnar klienter och sedan startas en ny tråd för varje klient som lyssnar på inkommande kommandon.

Den första idéen som implementerades var idé ett med 2 trådar. Valet gjordes då risk för att hanterandet av gemensamma resurser, som t.ex. manipulation av virtuella enheter, skulle komma att krocka ifall varje klient tilläts att köra sin egen tråd. Utvecklingen kom till ett steg där viss funktionalitet fanns men där kommunikationen ej fungerade i alla fall utan den fungerade som tänkt ibland och ibland inte. Vid diskussion med handledare så fastslogs att problemet med hanterandet av gemensamma resurser ej var något som behövdes undvikas, varpå idé nummer två med flera trådar implementerades. Detta fungerade utan större problem och blev den lösningen som finns i servern i slutprodukten.

Eftersom servern ej talar till appen direkt utan endast svarar på kommandon så finns ej möjlighet att uppdatera dimningsnivåer eller annan info mellan anslutna enheter.

5.9 CLI

Mot slutet av projektet gjordes även en version som kunde köras via ett CLI(command line interface). Det allra mesta av den befintliga serverkoden gick att återanvända. Dock fick observer-mönstret bytas ut mot en EventHandler istället för att skicka uppdateringar. Detta skulle även kunna fungera i GUIt vilket skulle möjliggöra delad kodbas för CLI och GUI versioner.

5.10 Debugging

När arbetet började gå mot en produkt som fungerar, om än med enkel funktionalitet, så började tid läggas på att lösa kända buggar, den största buggen handlade om att när en enhet installerats så blev resten oinstallerbara då mobilapplikationen tolkade dem som redan installerade fast i behov av återställning. Denna bugg tog 3 dagar att lösa då problemet var svårt att hitta. Felet låg i att en statisk byte array användes för att identifiera om enheten var ledig eller ej på byte 3. Vid genererandet av data så skickades antingen en byte array med byte 3 satt som 1 för upptagen eller 0 för ledig, men i installationsprocessen av första enheten så tilldelade Applikationen byte 3 värde 1 i den statiska byten för lediga enheter. Vid ny sökning så sattes därför alla enheter till upptagna, men systemet kunde ej hitta dem som installerade vilket gjorde att de tolkades som korrupta och i behov av återställning.

6 Tekniskt genomförande

I detta kapitel redogörs mer ingående om olika tekniker och ramverk använts i projektets genomförande och de tekniska lösningarna bakom de olika komponenterna.

6.1 Server

Servern byggdes som en Cocoa applikation och slutprodukten är en websocket-server som kan hantera flera klienter och fungerar som en brygga mellan klient och virtuell enhet.

6.1.1 Skriva en websocket server

Beslut fattades att skriva en WSS (websocket server), efter undersökningar av tillgängliga bibliotek och hur en websocket-server fungerar verkade det vara det snabbaste och mest lärorika sättet att skapa server-sidan.

En WSS lyssnar på inkommande TCP-sockets connections på en bestämd port. När en klient ansluter så förväntar sig servern en handskaknings-förfrågan, denna kommer i form av en HTTP-request, efter handskakningen är klar så skall en “stream” skapats där det är möjligt att skriva och läsa data, samt vissa kommandon som ping-pong och avsluta connection.

6.1.2 Handskaknings-processen

Klienten initierar kontakten genom att skicka en förfrågan med en HTTP header.

```
GET /chat HTTP/1.1
Host: example.com:8000
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

Figur 4. Exempel på klients http header med förfrågan till server om att initiera en websocket connection. [14]

Headern innehåller bland annat version och nyckel som skall användas för att skapa kopplingen, se Figur 4.
Servern som lyssnar efter inkommande connections kan se ifall meddelandet innehåller “GET” och ifall servern endast förväntas hantera websockets så kan handskaknings-processen från serverns sida påbörjas.

Servern svarar sedan med en HTTP header som bekräftar att protokollet ändrats till websocket-protokoll.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

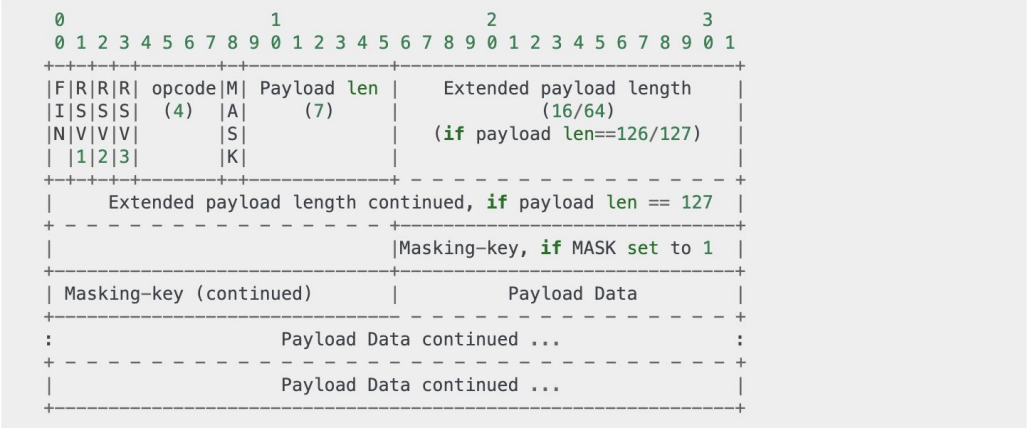
Figur 5. Exempel på Websocketservers svar till klients handskakningsförfrågan [14]

Det viktigaste fältet i serverns response är “Sec-Websocket-Accept:”, se Figur 5, som baserat på klientens nyckel[15]. För att generera detta fält kopieras klientens nyckel utan “whitespaces” och binds ihop med GUID:et “258EAFA5-E914-47DA-95CA-C5AB0DC85B11”.
Därefter så genereras en SHA-1 (Secure Hash Algorithm 1 [16]) hash utav den kombinerade nyckeln och GUID:ets bytes, vilket sedan konverteras till en sträng med base64 (metod för att omvandla binär data till t.ex ASCII tecken[17]).

När servern genererat sin “Sec-Websocket-Accept:” så skickas en http header tillbaka med fälten som i bild 4 och därmed är handskakningen slutförd, server och klient har bytt till att använda websocket protokollet för kommunikation.

6.1.3 Generera frames

När data skall skickas över websocketen så görs detta i form av en frame, se Figur 6. Framen innehåller overhead i form av olika bitar, en eventuell mask nyckel och payload. Eftersom websocketservern implementerades från start behövdes en metod för att generera flera frames dynamiskt ifall meddelandet ej fick plats i en enda frame.



Figur 6. En bild på en websocket frame som illustrerar vad bitarna representerar [14] .

bit 0
Är fin biten, denna är satt till ett ifall framen är den sista av frames i meddelandet, annars 0[15].

bit 1-3
Är rsv bitar som hanterar extensions, dessa extensions måste ha initierats innan dessa kan användas. Dessa bitar har ej behövts för detta projekt[15].

bit 4-7
Opcode, denna berättar hur meddelandet skall tolkas[15]:

- Hexvärde 0 tolkas som en fortsättnings-frame
- Hexvärde 1 tolkas som en text-frame
- Hexvärde 2 tolkas som en binär-frame
- Hexvärde 3-7 är reserverade för framtida icke control frames
- Hexvärde 8 är en control frame för close
- Hexvärde 9 är en control frame för ping
- Hexvärde A är en control frame för pong
- Hexvärde B-F är reserverade för framtida control frames

bit 8
Mask bit, är 1 ifall meddelandet är maskat och 0 annars, vid maskat meddelande måste Masknyckeln användas för att avläsa original meddelandet.

bit 9-15
Payloadens längd i bytes (0-125). Payloaden kan göras större, i dessa fall så behöver även overheaden för payloadens längs ökas på. Om värdet på bit 9-15 är 126 så tolkas nästkommande 2 bytes som en 16 bitars osignerad integer. Om värdet på bit 9-15 är 127 så tolkas nästkommande 8 bytes som en 64 bitars osignerat integer, den mest signifikanta biten måste i detta fall vara satt till noll.

bit 16-47
Masknyckel på 0 eller 4 bytes beroende om meddelandet är encodat eller ej. Denna används för att decodea meddelandet.

resterande bitar
Payloaden.

För control-frames och små meddelandet så räcker det med en frame, men när långa listor av virtuella enheter har serialiserats till json-format och skall skickas över så behövs det mutlipla frames. Jag valde att generera flera frames när längden på meddelandet blev längre än 125. Man hade även kunnat uttöka payload längden, men det skulle då finnas risk att den nya längden inte heller räckte till och flera frames skulle behöva genereras ändå. För att slippa ta hänsyn till för många fall valdes att endast skicka frames med max 125 bytes i längd.

Servern genererar frames dynamiskt och sätter fin biten till 1 när sista framen genererats, samt payload length till 125 eller längd av sista biten av payloaden på sista framen.

6.1.4 Avmaska meddelanden

När meddelande kommer från appen så är de maskade, detta betyder att man måste avmaska eller “decodea” dem. Detta gör man genom att köra meddelandet genom en decoding process där man för varje byte i payloaden[i] kör en XOR operation (ger 1 om två olika bitar jämförs, annars 0) med maskbyte[i%4] se figur 7.

```
byte[] decoded = new byte[msglen];
byte[] masks = new byte[4] { bytes[offset], bytes[offset + 1], bytes[offset + 2], bytes[offset + 3] };
offset += 4;

for (int i = 0; i < msglen; ++i)
    decoded[i] = (byte)(bytes[offset + i] ^ masks[i % 4]); // ^ for XOR
```

Figur 7. Avmaskningsproceduren i kod.

Exempel
Låt säga att vi har en payload på 8 bytes, då kommer payloadens byte 1 avmaskas genom att göra en XOR operation med maskens byte 1. Samma gäller för byte 2,3,4. Sedan kommer payloaden fortsätta räkna upp sina bytes men masken återgår till sitt första värde. Nästkommande operation blir då mellan payloadens 5e byte och maskens första.

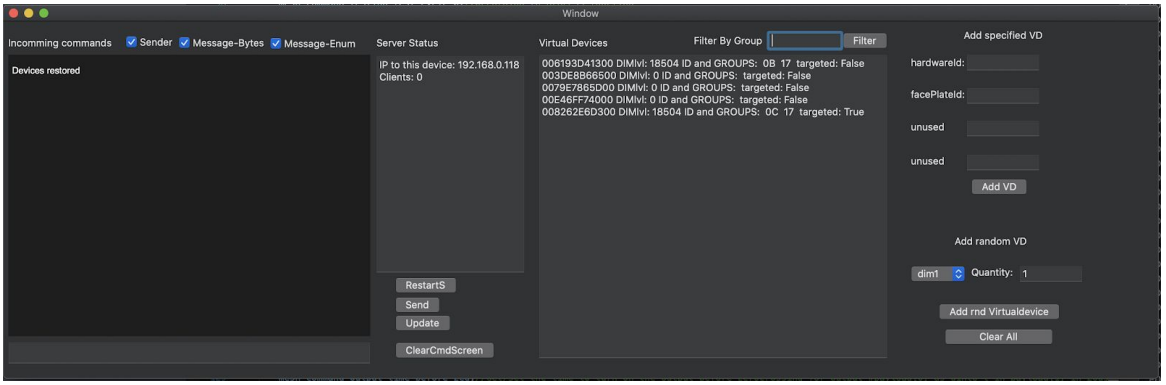
Avmaskningsprocessen görs för meddelanden som ej är controll frames.
När meddelandet har avmaskats så kan det skickas vidare för att hanteras.

6.1.5 Hantera meddelanden

Textmeddelanden skickas efter avmaskning vidare till en metod som hanterar kommandon. Här finns det vissa specialkommandon som har implementerats av mig, t.ex. så finns det specialkommando för connect, get_devices och read. Om det inte är ett specialkommando så är det ett av appens vanliga kommandon och då görs en koll om kommandot stöds av de virtuella enheterna och skickas isåfall vidare, annars så hanterar servern kommandot genom att svara med förväntat svar. Förväntat svar genereras med den enkla regeln: svara med identiskt om dont respond flaggan är satt, annars svara med identiskt meddelande men sätt ack-flaggan.

6.2 Gui

Servern är byggd med Visual Studio som en Cocoa applikation. GUIt byggdes med hjälp av Xcodes interface builder, vilket är ett enkelt verktyg som tillåter drag and drop av komponenter.



Figur 8. Servern i uppstartat läge med 5 virtuella enheter, varav 2 st installerats.

Guit är framtaget dels för att vara ett hjälpmedel vid utveckling av servern men även för att fungera som en en applikation med bra översikt för debugging, se Figur 8.

Mer ingående information om knappar och fält finns i readme-filen (bilaga 1).

6.3 CLI

Serverns funktionalitet var enkel att flytta över till ett .NET CORE projekt vilket gjorde att ett command line interface snabbt kunde utvecklas och testas, se Figur 9. Eftersom det så enkelt gick att flytta över det som utvecklats till CLI så testades bara enkel funktionalitet från CLI och när bekräftelse gjorts att detta fungerade fortsattes utvecklingen i GUIT med tanken om att det i ett senare skede är lätt att konvertera till eller dela kodbas med CLI.

Figur 9. CLI version av server.

6.4 Virtuella enheter

I Plejds mobilapplikation så finns flera klasser och interface för att representera de fysiska enheterna. Några exempel är IUnclaimedPlejdMeshDevice, IDevice, IDiscoveredPlejdMeshDevice. Vid användning av appen så används de olika komponenterna för att representera enheter som upptäckts, som ej installerats och som redan finns installerade på systemet.
Då undersökningar gjorts att dessa interface var nyckelkomponenter för att hantera enheter så kunde en ny typ av device skapas, nämligen VirtualDevice, som implementerade ovan nämnda interface.
Första idén var att utveckla en virtuell representation för varje interface, men då detta skulle bli mindre överskådligt testades att förena alla under en klass och då detta fungerade så behölls den lösningen.

Vidare så behövdes de virtuella enheterna byggas ut för att kunna hålla värden som outputgroup(som används för att sätta belysningen), dimlevel(som håller värdet av ljusstyrka) och groups(håller värdet av de grupper enheten tillhör). Detta är värden som annars kan sättas och läsas i de fysiska enheterna.

Då de fysiska enheterna kan hantera kommandon så skapades även funktionalitet för detta i de virtuella enheterna. För att göra detta skalbart skapades en lista med kommandon de virtuella enheterna stödjer och för varje kommando en tillhörande metod som kan hantera kommandot inklusive eventuell payload.
Denna innehåller vid projektets slut 3 kommandon och är enkel att bygga ut.
Stöd skrevs för:

- mesh_command_output_groups (lägger till enheten till valda grupper i installationsprocessen)
- mesh_command_group_output_state_and_level (sätter nivå och på/av för enheter)
- mesh_command_group_output_state (sätter enheter på/av)

Vilka ändrar de virtuella enheternas “state” och uppdaterar GUIt.

6.5 Virtuell Adapter

Själva navet i kommunikationen är adaptern. Det är hit som appen gör förfrågan och skickar kommandon samt skannar efter nya enheter. I normalläget så används en bluetooth-adapter, men appen är konstruerad så att alla klasser som implementerar interfacet “IAdapter” kan användas. För att få igång kommunikationen med servern så gjordes en Virtuell adapter som använde en websocket klient för att skicka och ta emot meddelanden.

6.6 Virtuell Websocket klient

En wrapper till en websocket skapades som kallades VirtualWebsocketClient. Anledningen till att en wrapper skapades var för att kunna kontrollera avbrutna sessioner, reconnecta och ha bättre kontroll över att skicka och ta emot meddelanden.

6.7 Virtuell Datagenererare

Data måste genereras och hanteras för att skapa virtuella enheter, skapa sk “advertisement data” som används vid installation och för att serialisera och deserialisera listor av virtuella enheter. En klass för att hantera alla dessa sorters datamanipulation skrevs och döptes till VirtualDataGenerator. Denna används av både mobil-applikationen och servern.

6.8 App kommunikation

Om overhead tillhörande websockets eller i vanliga fall bluetooth skalas bort så skickar appen meddelande direkt till enheterna bestående av 5 eller fler bytes.
Om meddelandet består av fler än 5 bytes så är alla övriga bytes en payload tillhörande kommandot. T.ex. kan kommandot vara “sätt dimnivå” och payload kan då innehålla värdet till vilket dimningsnivån skall sättas.

Byte 1 är det Meshid som meddelandet är riktat till. Det kan vara ett id till en specifik enhet eller till en grupp.

Byte 2 är versionen för meddelandet, i skrivande stund är denna alltid satt till 1.

Byte 3 är en flagga som kan användas för att läsa, skriva eller skicka med information om appen förväntar sig ett svar eller ej.

Byte 4 och 5 är ett kommando till enheten, t.ex. sätt dimningsnivå, stäng av, gör inställning.

Eventuella övriga bytes är payload.

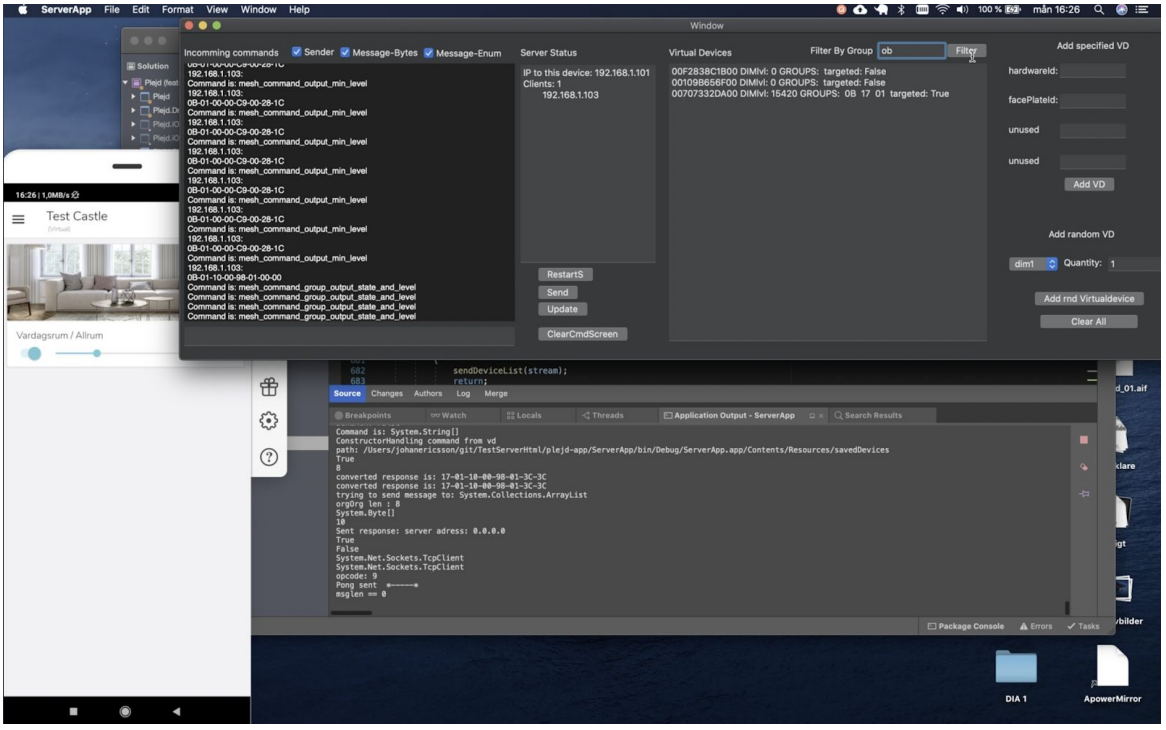
Exempel
Låt säga att vi får meddelandet 1D-01-10-00-98-01-23-23

1D	id:et eller gruppen till vilket meddelandet riktar sig.
01	versionen, alltid 01 i skrivande stund.
10	flaggan satt till “don't respond”
00-98	kommando “mesh_command_group_output_state_and_level”
01-23-23	payloaden

Detta bryts då ned som att skicka kommando 152 (98 i Hexadecimalt format) enumen för detta är “mesh_command_group_output_state_and_level” till grupp eller id 1D med version 1 och flagga 16(10 i Hexadecimalt format) enum: “DontRespond”.
Payoloaden är 01-23-23 och det betyder sätt utgång 1 till värde 2323(hex) vilket är 8995 i decimal form. Dimningsnivåerna går mellan 0-65535 (0-FFFF i hex) så denna dimningsnivå är förhållandevis låg.

7 Resultat

Projektet resulterade i en server som kan skapa och tillhandahålla information om virtuella enheter som avspeglar fysiska enheter från Plejds produkter, se Figur 10. Servern kan även kommunicera med en modifierad version av Plejds Android applikation och svara på kommandon, samt vidarebefordra vissa kommandon till de virtuella enheterna.



Figur 10. Server och avspeglad Android telefon körandes applikation.

8 Diskussion

Arbetet med att sätta sig in i och försöka efterlikna Plejds system har varit både kul, utmanande och lärorikt. Att återskapa fysiska objekt eller system virtuellt görs redan inom en rad discipliner och uppskattningsvis så kommer det bara bli vanligare i framtiden att ta fram digitala versioner av produkter och system.

Genom att prata med de anställda så har jag även fått uppfattningen att ett fullt utvecklat system skulle kunna snabba upp och förenkla många utvecklings och test arbeten på företaget. Generera stora siter skulle kunna gå på ett ögonblick, system med nya enheter skulle kunna testas innan ens den nya produkten är färdigutvecklad och de anställda skulle kunna bli mer mobila.

8.1 Utvecklingspotential

När grunden nu är lagd så finns det mycket utvecklingspotential för den virtuella miljön. Jag delar in framtida visioner för projektet i 3 delar, där den första är sådant som man skulle vilja se av en helt färdig produkt. Det är på sätt och vis det som inte har hunnits med och som var utanför avgränsningarna för arbetet men som är ett naturligt nästa steg i en utvecklingsprocess.

8.1.1 Nivå 1

Utveckling som skulle kunna vara en naturlig fortsättning vid eventuell vidareutveckling.

Filhantering

Hit hör hantering med sparfiler, just nu finns endast en autosave som sparar vid förändring av virtuella enheter. Möjlighet att spara undan och ladda in flera uppsättningar av virtuella enheter skulle komma som en naturlig del, det borde även gå relativt snabbt att implementera då tekniken är densamma som i autosave.
Vid utveckling har det hänt att sparfilen blir korrupt, sannolikt därför att programmet avslutas oväntat mitt i en skrivning till fil. Även om detta är relativt ovanligt och skulle ha mycket lägre sannolikhet i en stabil release som ej debuggas så kan det vara värt att ha en automatisk backup feature och kanske även en feature som kan laga korrupta filer. När det kommer till att laga dem så handlar det om att “lappa ihop” json dokumentet. Man kan förstås ej ersätta data som förlorats i sin helhet, men om hälften av datan finns kvar så kan man tänka sig att den räddar det som finns. En enkel lösning skulle vara att matcha förväntade tecken ([] {} “” : ,), lista ut vad som fattas och därefter generera en ny fil med data man lyckats rädda. Om systemet redan installerats på appen så finns information om det tillgängligt via appen eller via en cloud-server och det finns då möjlighet att utveckla funktionalitet för att återställa eller spegla detta.

Server initiativ till kommunikation

I sin nuvarande form då detta skrivs så inleds kommunikationsprocessen alltid med att klienten tar initiativet och servern svarar. Detta är förväntat i en inledande sammankoppling av de båda, men efter att handskakningen slutförts så har servern

möjlighet att ta initiativ till kommunikation med klienten. Detta är önskvärt för att spegla de fysiska puckarna som ibland hör av sig med information (om bland annat deras dimnivåer och ifall en fysisk lampknapp har trycks på och på så sätt ändrat enhetens värde). Det kan även ha framtida potential att ladda in information till appen som t.ex. gör att man kan kringgå installationsprocessen.
Om servern skall kommunicera på detta sätt behövs websocket klienten på appen modifieras. Varje gång klienten hör av sig så förväntar den en respons av ett eller annat slag från servern och i nuvarande version så lyssnar den endast när den förväntar sig svar. En klient skulle behöva köras i bakgrunden och lyssna kontinuerligt och fall där ett oväntat servermeddelande råkar hamna mellan klient meddelande och respons, så att klienten erhåller serverns oförväntade meddelande när det förväntar sig en respons, måste hanteras. Detta skulle kunna göras relativt enkelt genom att ge server-initierade meddelande en speciell overhead varpå de kan filtreras ut och behandlas separat från den normala responderingen.

Synliga svar
Just nu skrivs svaren som servern skickar endast ut i consolen, ett önskemål från app-teamet var att även visa detta i guit. Detta kan implementeras fort, svårigheten ligger främst i att presentera det på ett användarvänligt sätt.

Ta bort enheter
Efter att ha testat och installera enheter måste man i dagsläget ta bort dem en efter en, detta kan vara en tidsödande och tråkig process och det vore bra ifall det gick att snabba upp denna process.

Stöd för fler/alla enheter
Plejd tillverkar flera olika sorters enheter, de två som jag har haft möjlighet att testa och debugga under projektet är dim-01 och dim-02 enheterna och främst dim-01 fungerar i slutprojektet. Det skulle vara möjligt att implementera stöd för alla produkter som finns tillgängliga och kanske även enheter som inte har producerats ännu.

Stöd för fler/alla kommandon
Det finns runt hundra kommandon och dessa kan läggas in för att kunna hanteras av de virtuella enheterna. I dagsläget finns stöd för tre, men grunden har lagts för att enkelt och gradvis kunna implementera fler.

Implementera alla guits komponenter
Send knappen är en kvarleva från första iterationen då servern var en chat server, denna fick dock vara kvar så att det i framtiden kan bli möjligt att skriva och skicka meddelande eller kommandon till appen från servern. Sektionen för Add specifik VD är heller ej implementerad, men enkel att koppla till passande konstruktor för virtuella enheter.

Förfina GUI
Guit är framtaget för att ge en överblick för hur saker hänger samman och vad som händer, ingen tanke om utseende eller användarvänlighet har gjorts. Kanske kan Guit göras bättre mer intuitivt och på ett sätt att det är enkelt att lägga till funktionalitet.

Delad kodbas för GUI och CLI

GUI och CLI kan köra samma server klass om små förändringar görs, GUIt har ett observer mönster som kan bytas ut mot event handlers istället och detta skulle möjliggöra för GUI och CLI att dela koden.

Multiconnection
Fungerar, men inte felfritt. Detta skulle kunna förbättras så att alla enheter kan ansluta och återansluta sömlöst och problemfritt.

8.1.2 Nivå 2

Utvecklingsmöjligheter som ligger längre fram i tiden och till viss del förutsätter att flera nivå 1 funktioner redan är utvecklade.

Överföra rum och siter
I nuvarande version så ämnar servern att efterlikna verkligheten och därmed måste alla enheter installeras. I ett scenario där installationsprocessen ej är intressant skulle det kunna vara effektivt att sätta upp en hel konfiguration inuti servern och sedan ladda upp denna till cloud och app.

Support for iPhone
Projektet genomfördes med en Android telefon för att debugga, genom xamarin så finns den mesta koden redan genererad även för iphone, men vissa detaljer måste läggas till för att den editerade appen skall kunna köras även på iphone.

8.1.3 Nivå 3

Utvecklingsmöjligheter som ligger långt in i framtiden.

Webbaserad testmiljö
I och med att man kan köra appen mot en mjukvarumiljö så skulle detta kunna användas för att göra ett virtuellt system öppet för allmänheten i komersiellt syfte. Via en webbsida skulle man kunna erbjuda test av Plejd system till kund i syfte att demonstrera systemets intuitivitet och vilka möjligheter det öppnar upp för.

8.2 Svar till frågeställningar

Fråga
Kan man skapa en fungerande virtuell miljö som fungerar med appen?
Svar
Ja, detta har bevisats genom implementation av server som ger en fungerande virtuell miljö ihop med en modifierad version av appen. Med fungerande virtuell miljö menar jag då att konceptet fungerar, virtuella enheter kan genereras, upptäckas och installeras. Dock är en modifierad version av appen ett måste. Detta var förväntat från start då appen behöver koppla upp sig mot en server istället för att ansluta till de fysiska enheterna.

Fråga
Kan man skapa virtuella enheter som appen hitta, installera och kommunicera med?
Svar
Ja, detta är bevisat då funktionaliteten för att hitta, installera och kommunicera är implementerad och fungerar. Med utrymme för förbättring, kommunikationen sker inte direkt med de virtuella enheterna nu förutom för vissa förutbestämda kommandon. Att bygga ut de virtuella enheternas kommunikationsmöjligheter går att göra gradvis.

Fråga
Hur skall de kommunicera?
Svar
Server och app kommunicerar via Tcp och använder websocket protokoll. Servern tillhandahåller kommunikationen med de virtuella enheterna.

Fråga
Behövs extra mjukvara? Och isåfall vad?
Svar
Mjukvaran som implementerats och som är nödvändig är servern och en modifikation på appen.

Fråga
Kan man installera de virtuella enheterna så att de känns igen och minns av systemet.
Svar
Ja, installationen fungerar fast med ett modifierat version då krypteringssteget hoppas över. Installationen sparar även de virtuella enheterna till molnet (Plejds egna servrar) på samma sätt som med de fysiska enheterna, vilket gör att de sparas på systemet i vanlig ordning.

Fråga
Kan man få dem ge liknande respons från virtuella enheter som fysiska enheter som appen kan plocka upp?
Svar
Jag tror mig ha samlat tillräckligt med kunskap och bevis för att detta till stor del är möjligt. Då systemet är simulerat så blir responsen aldrig exakt som i verkligheten, t.ex. responstid skiljer sig åt.

Fråga
Kan man i en virtuell miljö skapa funktionalitet som ej skulle vara möjligt eller opraktiskt med fysiska enheter?
Svar
Fördelen med virtuella enheter är att man kan editera dem efter sina önskemål enkelt genom att ändra i koden. Således är det möjligt att testa funktionalitet som på en fysisk enhet skulle kräva att man gjorde hårdvaru-ändringar.

Fråga
Kan man skapa en server för att erhålla den virtuella miljön?

Svar
Ja, detta har bevisats genom att utvecklad server - app funktionalitet fungerar som tänkt.

Fråga
Kan man få en eller flera klienter att ansluta och erhålla serverns miljö?
Svar
Ja detta har bevisats då test gjorts med en och flera enheter. Multiconnections är ej fullt fungerande i slutprodukten, men tillräckligt utbyggd för att man med säkerhet kan anta att ingenting skulle förhindra att den virtuella miljön att stödja flera enheter vid vidareutveckling.

Fråga
Vilken typ av kommunikationsprotokoll är lämplig för en sådan server?
Svar
Valet gjordes att använda Websocket-protokollet. Alternativt hade ett eget lite simplare protokoll kunna uppföras, men detta skulle ta extra tid. Http skulle kunna användas, det används redan i handskakningsprocessen, men http är ej full-duplex och kommunikationen skulle ej bli lika sömlös. Vidare är websocket ett väl etablerat protokoll vilken kan vara en fördel då nya användningsområden för servern kan uppkomma och då finns ett välkänt protokoll på plats för att användas.

Kan en virtuell miljö vara till nytta för utvecklarna/företaget?
Efter att ha presenterat idén för olika team på plejd så har flera användningsområden diskuterats och vid tillräckligt god funktionalitet är konsensus att en virtuell miljö skulle kunna användas på flera sätt som skulle vara till nytta för företaget. Exempel som uppkommit är smidiga demo-tillfällen, testa ny funktionalitet, hastighet, testning, kommersiella produkter.
Detta ger en viss indikation till att systemet skulle kunna vara till nytta för företaget.

Kan en virtuell miljö vara till nytta ekonomiskt?
Vid ett väl fungerande virtuellt system är ekonomisk vinning en möjlighet på speciellt 2 punkter. Den första är att genom att använda virtuella enheter så behöver man inte förlita sig lika mycket till de fysiska test enheterna och därav minska behovet att producera nya enheter för test.
Den andra punkten är tidsmässigt. Virtuell miljö ger möjlighet att skapa, testa och utveckla mot enheter väldigt tidseffektivt. T.ex så kan 50 enheter genereras med ett knapptryck och möjlighet finns till snabb installation. Motsvarande händelse förlopp skulle ta timmar vid användande av fysiska enheter och detta kan istället göras inom loppet av ett par minuter.
Detta måste ställas i relation med att det finns risker med att förlita sig på ett virtuellt system i för stor utsträckning. Ett exempel skulle kunna vara att testning med det virtuella systemet hindrar testarna att upptäcka problem med det fysiska

problemet och att detta kan ställa till större ekonomisk skada på lång sikt än vad det virtuella systemet genererat.

Kan en virtuell miljö vara till nytta utvecklingsmässigt?
Möjlighet finns att skapa virtuella enheter efter egna önskemål. Man kan alltså skapa enheter som fortfarande är i planeringsstadiet från hårdvarutillverkarna. Detta möjliggör att utveckling sker parallellt med hårdvaruutveckling och att en produkt kan tas fram snabbare.

Hur mycket funktionalitet från originalpuckarna kan man göra virtuellt?
Vad gäller funktionaliteten så har inget hinder påträffats som omöjliggör utveckling av all funktionalitet som hårdvara besitter. Detta ger en indikationen till att all funktionalitet som de fysiska enheterna besitter borde kunna simuleras. Det största problemet som skulle kunna uppkomma är en situation där det simulerade systemet ej tar hänsyn till hårdvarans begränsningar, vilket skulle kunna leda till att tester i det virtuella systemet fungerar men samma tester

9 Slutsats

I detta arbete har jag visat att det är möjligt att skapa ett virtuellt system av Plejds hårdvara som fungerar ihop med en modifierad version av appen. Det virtuella systemet som byggdes är i många avseenden begränsat, men det visar att grundidén fungerar och systemet kan byggas ut för att återspegla mer funktionalitet från enheterna.

Källor

[1 ny] The AnyLogic Company, “Use of Simulation – AnyLogic Simulation Software”, 2020. [Online]. Tillgänglig: https://www.anylogic.com/use-of-simulation/, hämtad: 2020-06-03.

[2 klimatsim] SimoneFatichiab, et al., “Simulation of future climate scenarios with a weather generator - ScienceDirect”, 2011. [Online]. Tillgänglig: https://www.sciencedirect.com/science/article/abs/pii/S0309170811000042, hämtad: 2020-05-29

[3 biotech]Montanino, A., Kleiven, S., "Role of lipid composition on the structural and mechanical features of axonal membranes: a molecular simulation study", KTH, Stockholm, Sverige, Springer, 2019. Vol. 18, s. 27-39. [Online]. Tillgänglig: http://kth.diva-portal.org/smash/get/diva2:1322253/FULLTEXT01.pdf, hämtad: 2020-05-25.

[4 carcrashsim] CHM, “Car Crash Simulation | Make Software, Change the World! | Computer History Museum”, 2020. [Online]. Tillgänglig: https://www.computerhistory.org/makesoftware/exhibit/car-crash-simulation/, hämtad: 2020-05-24.

[5 emulering] unknown, “Emulator - Wikipedia”, 2020. [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Emulator, hämtad: 2020-05-27

[6] Xcelgo, “Emulation vs simulation – what is the difference?” 2020. [Online]. Tillgänglig: https://xcelgo.com/emulation-vs-simulation/
hämtad: 2020-05-26

[7] Microsoft Corp, “What is Xamarin? | .NET”, 2020. [Online]. Tillgänglig: https://dotnet.microsoft.com/learn/xamarin/what-is-xamarin, hämtad: 2020-06-02.

[8] Apple inc., “What Is Cocoa?”, 2013. [Online]. Tillgänglig: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html, hämtad: 2020-06-03

[9] Microsoft Corp, “.NET Core intro and overview | Microsoft Docs”, 2020. [Online]. Tillgänglig: https://docs.microsoft.com/en-us/dotnet/core/introduction, hämtad: 2020-06-03

[10] Microsoft Corp, “Overview of Visual Studio | Microsoft Docs”, 2020. [Online]. Tillgänglig: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019, hämtad: 2020-06-03

[11] Apple inc, "Xcode - IDE - Apple Developer" 2020. [Online]. Tillgänglig: https://developer.apple.com/xcode/ide/, hämtad: 2020-05-24.

[12] Apple inc, "Xcode - Interface Builder - Apple Developer" 2020. [Online]. Tillgänglig: https://developer.apple.com/xcode/interface-builder/, hämtad: 2020-05-25.

[13] Yamuna Navada, "What are WebSockets?? - Frontend Weekly - Medium", 2018. [Online]. Tillgänglig: https://medium.com/front-end-weekly/what-are-websockets-7bf0e2e1af2, hämtad: 2020-06-01.

[14] Mozilla and individual contributors, "Writing WebSocket servers - Web APIs | MDN" 2020. [Online]. Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers, hämtad 2020-04-17

[15] The WebSocket Protocol
Internet Engineering Task Force
2020. [Online]. Available:
https://tools.ietf.org/html/rfc6455#page-6

[16] D. Eastlake, et al., "RFC 3174 - US Secure Hash Algorithm 1 (SHA1)", 2001. [Online]. Tillgänglig: https://tools.ietf.org/html/rfc3174, hämtad: 2020-05-25.

[17] base64.guru, "What is Base64? | Learn | Base64", 2020. [Online]. Tillgänglig: https://base64.guru/learn/what-is-base64, hämtad: 2020-05-21.

[simulering] Nationalencyklopedin AB, "simulering - Uppslagsverk - NE.se", 2020. [Online]. Tillgänglig: https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/simulering, hämtad: 2020-06-02.

Bilagor

Readme

Starting the server
The server can be run from the development environment or directly from a built app (only tested on visual studio and mac).
Shared libraries such as Plejd.Shared must be imported to the project.

---Server-Windows---

When server is running info is displayed on 3 windows.

The left window is showing communication from plejd-app.

The middle window shows the servers ip and connected clients.

The right window shows the virtual devices sever is hosting and info about their dim level, which groups they belong to and wether they are targeted (connected to) or not.

---Server-Buttons---

RestartS: Restarts the server, drops all clients and reload devices from file, a time-saving feature when something went wrong and a fast reboot might be needed.

Send: Not yet implemented, could be used to send arbitrary data to connected devices.

Update: Updates all windows, shouldn't be needed in final version but can come in handy if some functions not updating windows after done.

Filter: Filters the devices, only showing those who belongs to the certain groups

Add VD: This button and section is under development, should in future version add more specific virtual devices.

Add and VirtualDevice: Adds virtual devices of selected dim and quantity.

Clear All: Deletes all virtual devices from server. (App needs to be done manually)

---About the server---

The server runs and accept multiple connections from web socket clients. The functionality is prepared for use with pled-app, running with App.StorageManager.VirtualDevicesServerEnabled set to true and App.StorageManager.VirtualDevicesServerHost set to the IP of the server.

When connected server provides scanning, installation and basic communication features.

The server will always respond using simple rules that app seems to expect, this should be looked into more deeply.

Commands can be forwarded to Virtual Devices that can react to them properly. When writing this Virtual Devices supports 3 commands. Adding more supported commands can be done in Virtual-Devices and server will pick them up and forward the commands supported. Server is still answering after Devices handle the command. In future versions it might be desirable to let Virtual Devices have their own logic for answering.

When virtual devices are added, edited or removed it saves a list of the current setup to the apps folder in Contents/Resources/savedDevices/devices.txt . Rarely, but has happened when debugging, this file can become corrupt if server crash or is terminated in the progress of writing to file. Clearing the text file deletes all saved devices and solves the problem.

---Future Features---
*The save file should be handled so there can be no errors.
*Ability to save and load different setups.
*Server only answers on communication, in future virtual devices and server might want to initialize contact as pucks do in reality.
*Server should display its messages.
*Devices should be able to be deleted faster.
*Thinking larger, server could potentially provide rooms or site setups.
*Thinking even larger, server could be run in web browser, making rooms or setups or whatever fun available for customers to try out with special test app or so.s
*Send button should send message to specified clients.
*Add specified VD should be implemented
*Dim2 devices should be added with different logic that accurately mimics real world behavior
*Server GUI could use some love
*GUI and CLI should share serverCodeBase. Probably easy to fix if GUI uses eventhandling instead of observer.
*Multiconnection needs to be improved with removal of unused threads and streams
*Manufacturerdata could be generated from id in a backwards process from ParseID
*Server or VD should respond more authentic, right now server sends expected response even if we try to dim a non existing device.
*Support for iPhone.