Digital skyltning på Androidenheter

Ett Android integrationsprojekt

Digital signage with Android devices
An Android integration project

Berglund, Simon

Fakulteten för hälsa, natur- och teknikvetenskap
Datavetenskap
C-uppsats 15hp
Handledare: Thijs Jan Holleboom
Examinator: Donald F. Ross
Oppositionsdatum: 8/6 - 2016

Digital skyltning på Androidenheter - Ett Android integrationsprojekt

Digital signage with Android devices - An Android integration project

Berglund, Simon

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Simon Berglund

Godkänd 8/6-2016

Opponenter: Mattias Skarman, Jacob Östelid

Handledare: Thijs Jan Holleboom

Examinator: Donald F. Ross

Sammanfattning

Digital skyltning är en speciell form av skyltning där information, bilder, videoklipp eller liknande media visas på digitala skärmar. Skyltningen sker vanligtvis på LCD eller LED skärmar och placeras vanligtvis på antingen offentliga platser eller på arbetsplatser.

Projektet går ut på att expandera en färdig digital-skyltningsmjukvara där koden huvudsakligen är skriven i HTML5 och JavaScript genom att göra den exekverbar och anpassad för operativsystemet Android.

Resultatet av projektet gjorde mjukvaran anpassad samt exekverbar på Android. Verktyget Cordova användes för att göra webb-koden exekverbar på Android. Anpassningen gäller huvudsakligen att ge mjukvaran tillgång till Androidenhetens funktioner och resurser som exempelvis filsystemet och skärmdumps-funktionen.

Abstract

Digital signage is a special form of signage where information, photos, videos or other media is displayed on digital screens. The signage is usually performed on LCD or LED screens and is usually placed in either public places or workplaces.

The project aims to expand a complete digital signage software created mainly from code written in HTML5 and JavaScript, by making it executable and adapted to the Android operating system.

The result made the software adapted for Android and also allowed it to execute on Android. A tool named Cordova was used to make the web-code executable on Android. The adaption applies mainly to give the software access to functions and resources such as the file system or the screenshot function from an Android device.

Innehållsförteckning

Sammanfattning....v
Abstract....vii
Kapitel 1....1
Introduktion....1
1.1 Förväntningar och resultat....1
1.2 Disposition....2
1.2.1 Introduktion....2
1.2.2 Bakgrund....2
1.2.3 Design....2
1.2.4 Implementation....2
1.2.5 Resultat och utvärdering....3
1.2.6 Slutsats....3
Kapitel 2....5
Bakgrund....5
2.1 DISE International....5
2.2 DISE....5
2.2.1 DISE XPRESS....6
2.3 Projektets uppkomst....7
2.4 Uppgiften....8
2.5 Extra....9
2.6 Sammanfattning....9
Kapitel 3....11
Design....11
3.1 Konverteringen....11
3.2 Cordova....11
3.3 Struktur....13
3.4 Crosswalk....15
3.5 Plugins....15
3.5.1 File / File-transfer....15
3.5.2 Automatisk uppstart....17
3.5.3 Screenshot....17
3.5.4 Videospelare....19
3.6 Omstart vid krasch....19
3.7 Sammanfattning....22
Kapitel 4....25
Implementation....25
4.1 Konverteringen....25
4.2 Cordova....26
4.3 Struktur....27
4.4 Crosswalk....29
4.5 Plugins....29
4.5.1 File / File-transfer....30
4.5.2 Automatisk uppstart....31
4.5.3 Screenshot....31
4.5.5 Videospelare....32
4.6 Omstart vid krasch....32

4.7 Problem....34
4.8 Sammanfattning....38
Kapitel 5....39
Resultat och utvärdering....39
5.1 Skall kraven....39
5.1.1 Starta automatiskt vid start av dator....39
5.1.2 Starta om app automatiskt vid krasch eller liknande fel....40
5.1.3 Stödja de HTML5/JavaScript funktioner som DISE XPRESS använder....40
5.1.4 Lagra filer lokalt på enheten och fungera även om enheten inte har tillgång till nätverk....40
5.1.5 Spela upp utan interaktion från användaren....41
5.1.6 Spela bilder och video blandat och oavbrutet under 24 timmar utan omstart av enhet eller applikation....41
5.1.7 Spara skärmdump samt uppladdning av denna till server. 41
5.2 Önskade krav....41
5.3 Sammanfattning....42
....42
Kapitel 6....43
Slutsats....43
6.1 Sammanfattning....43
6.1.1 Introduktionen....43
6.1.2 Arbetsprocessen....44
6.1.3 Avslutande tankar....45
Referenser....47
Bilagor....51
Bilaga 1 – Heap allokering....52
Bilaga 2 - Minne....53
Bilaga 3 - MainActivity....54
Bilaga 4 - StartAppService....55
Bilaga 5 – FileApi_Cordova....56
Bilaga 6 - Platform_Cordova....60

Illustrationsförteckning

Illustration 1: Komponenterna för en XPRESS spelare. (Avlägsen server + modern skärm) [9].....7
Illustration 2: Schema över projektets Cordova applikation.....13
Illustration 3: Val av plattform och dennes plattformsspecifika funktioner.....14
Illustration 4: Anpassade funktionen för nedladdning av fil till Androidenheten. 16
Illustration 5: Exempelkod för att spara skärmdumpen på enheten.....18
Illustration 6: Exempelkod för att hämta skärmdumpen som Data URI.....18
Illustration 7: Skärmdumps-funktion anpassad för DISE XPRESS.....19
Illustration 8: Deklarationen av Service-klassen.....20
Illustration 9: Service som startar om applikationen vid krasch.....21
Illustration 10: Funktionen som kollar om DISE XPRESS är aktiv på enhetens skärm.....21
Illustration 11: Uppstartning av applikationen och skapandet av Service-klassen.....22
Illustration 12: Medveten krasch-kod som användes för testning. 32
Illustration 13: Hanterare för ouppfångade undantag samt sparandet av aktivitet.....33
Illustration 14: JavaScript funktion för att överflöda applikationens minne.....34
Illustration 15: Minnesallokeringen med samt utan funktionen som orsakade minnesläckan.....37
Illustration 16: Minnesanvändningen från DISE XPRESS applikationen. 37

Kapitel 1

Introduktion

Målet med detta arbete är utföra en konvertering av en mjukvara för digital skyltning. Med digital skyltning menas elektroniska skärmar som finns installerade på bland annat offentliga platser eller på arbetsplatser.[19] Mjukvaran för digital skyltning visar upp önskat innehåll så som video, bild eller text på skärmar i alla dess slag. Produkten som ska hanteras i arbetet kan för närvarande inte spelas på Androidenheter men ska kunna göra det efter projektets avslut. Företaget som äger produkten vet för närvarande inte om denna konvertering kommer bli lyckad då de är osäkra på om en Androidenhet kan uppfylla alla krav vad gällande funktionalitet samt prestanda som produkten kräver. Arbetet kommer därför vara en blandning av utveckling och anpassning samt utvärdering av resultat.

1.1 Förväntningar och resultat

Det förväntade resultatet är en mjukvara för digital skyltning som har möjligheten att bli installerad på en Androidenhet och med all den huvudsakliga funktionaliteten fortfarande fungerande. De farhågorna som finns är dock att prestandan på videoklippen inte kommer vara tillräckligt bra, som att de inte visar tillräckligt många bilder per sekund och därmed kommer visas med dåligt videoflyt.

Resultatet av arbetet blev i slutändan lyckat men inte helt utan problem, farhågan om dålig prestanda på videoklippen blev dock en sanning och som först vid projektets slut kunde lösas. Men det framkom även ett större problem tidigt i arbetet där applikationen kraschade efter ca 12 timmars uppspelning och som försökte lösas genom hela projektets gång. Men det problemet och felsökningen runt det kommer tas upp i kapitel 4 (4.7 Problem).

1.2 Disposition

Denna rapport är indelad i sex kapitel vid namn Introduktion, Bakgrund, Design, Implementation, Resultat och utvärdering och Slutsats. Dessa kapitel kommer nu att introduceras kort för att ge ett helhetsperspektiv på hela rapporten.

1.2.1 Introduktion

Detta kapitel befinner du som läsare dig i för närvarande. Här introduceras projektet med en summering av arbetet och målen för arbetet. Både de förväntade resultaten och en kort summering av det faktiska resultatet nämns här.

1.2.2 Bakgrund

I kapitlet bakgrund redovisas allt från relevant information om företaget till specifikationer i uppgiften. Även all information som läsaren behöver för att kunna förstå och uppskatta resterande kapitel i rapporten redovisas här.

1.2.3 Design

I designkapitlet kommer resultatet och designen på själva produkten redovisas. Det kommer visualiseras med flera bilder och kod-stycken som även kommer förklaras. Informationen i detta kapitel ger läsaren utvecklingsresultaten och allt om den färdiga produkten.

1.2.4 Implementation

Implementationskapitlet följer samma upplägg som designkapitlet vad gällande utformning

av rubriker och underrubriker av designkapitlet. Informationen i detta kapitel är utvecklingsprocessen och implementationen av lösningarna. Varje design-del har sin respektive implementations-del för att hålla god struktur i rapporten. Implementationskapitlet har även informationen om felsökningen vilket var en stor del i arbetet.

1.2.5 Resultat och utvärdering

I detta kapitel redovisas resultaten på designen och implementationen. Alla kraven från kravspecifikationen tas upp och jämförs med resultatet av projektet. Vilka punkter som uppfyllts samt inte uppfyllts presenteras i detta kapitel.

1.2.6 Slutsats

Slutsatskapitlet presenterar en sammanställning av projektet i allmänhet. Arbetsprocessen under hela projektet sammanställs för att ge en uppfattning av vad som varit lärorikt och hur arbetet utfördes. De avslutande tankarna beskrivs även i det här kapitlet och frågan ”om jag hade gjort någonting annorlunda om jag fick göra om projektet” besvaras.

Kapitel 2

Bakgrund

Detta kapitel kommer att handla om relevant bakgrundsinformation till projektet. Företaget DISE kommer beskrivas och uppkomsten till varför de ville ha sin spelarmjukvara körbar på operativsystemet Android.

2.1 DISE International

DISE International AB är ett globalt företag inom digital skyltning. Företaget har återförsäljare och distrubitörer i över 30 olika länder utspridda över hela världen och har två dotterbolag; DISE Asia och DISE Australia. Huvudkontoret ligger i centrala Karlstad, Sverige.[1]

2.2 DISE

Produkten företaget säljer heter DISE och är en mjukvara för digital skyltning. Med digital skyltning visas innehåll på digitala skärmar för att förmedla underhållning, marknadsföring eller allmän information. DISE mjukvara kan finnas inom olika marknader såsom transport, finans, regering, sport och detaljhandel. Exempel på användningsområden är spellistor med reklam i både video och bildformat. Mer avancerade tillämpningar finns inom direktuppdaterade flyg och busstider för transport eller börsförändringar inom

finansmarknaden.

Mjukvaran finns i fyra olika versioner för att kunna passa alla önskemål; DISE XPRESS, DISE SMART, DISE PREMIUM och DISE PROFESSIONAL. Detaljerade skillnader av dessa spelare kommer rapporten inte gå djupt in på eftersom det är orelevant information för detta projekt. Fokuset kommer ligga på XPRESS spelaren eftersom det är där arbetet har utförts.

2.2.1 DISE XPRESS

XPRESS spelaren är den enklaste av spelarna och kräver ingen extra enhet utan kan installeras och köras direkt i skärmen. På det sätter skiljer sig XPRESS spelaren från de andra spelarna där det behövs en dator som sedan är kopplad till en eller flera skärmar. XPRESS spelaren kan spela upp standardinnehåll som video, bild och text men kan inte hantera dynamiska innehåll eller touch-event som de mer avancerade spelarna kan. Skillnaderna är för att det ska finnas spelare anpassade för alla kunder, stora som små.

Spelaren fungerar på det sättet att DISE mjukvaran installeras på en eller flera skärmar som i sin tur hämtar innehåll att spela upp från en avlägsen server(illustrerat i *Illustration 1*). Servern tillhandahålls av antingen DISE eller en lokal återförsäljare. Innehållet från servern laddas ner och sparas på enheten så att den sedan kan spela upp innehållet utan kontakt med servern. Användaren har en egen inloggning till ett eget användargränssnitt där han eller hon kan ladda upp och hantera det innehåll som önskas ska skickas ut till sin eller sina skärmar ifrån servern. Så länge som enheten är uppkopplad till servern så hämtar den automatiskt det nya materialet som användaren väljer i servern.

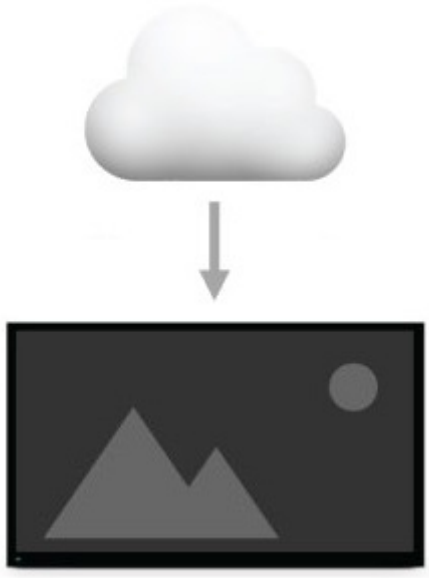


Illustration 1: Komponenterna för en XPRESS spelare. (Avlägsen server + modern skärm) [9]

2.3 Projektets uppkomst

Anledningen till att företaget vill ha DISE XPRESS spelaren körbar på operativsystemet Android är huvudsakligen för att vissa digitala skärmar har inbyggt stöd för Android direkt i skärmen. NEC och Panasonic är exempel på märken som har en integrerad Androidenhet i vissa mer avancerade skärmar. Om DISE XPRESS kan köras på Android så kan man då också installera mjukvaran direkt i dessa skärmar. Detta leder till att en större kundbas öppnas upp för företaget då det kan finnas vissa kunder som enbart vill använda sig av en viss sorts skärm. Fler kunder leder även till fler sålda produkter som därmed leder till högre inkomst för DISE.

DISE är redan på god väg med att öka antalet skärmar som XPRESS spelaren kan installeras på. Man har tidigare löst funktionaliteten och konverteringen till operativsystem som webOS för LG-skärmar och Samsung Smart Signage Platform för Samsung-skärmar.

En annan anledning till att företaget vill ha DISE XPRESS spelaren körbar på

operativsystemet Android är för att de vill ha samma kodbas för Android som för de andra operativsystemen. Man har nämligen i dagsläget en körbar spelare för Android (XPRESS SMART) som har sin helt egna kodbas för Android. Detta betyder att så fort det skall göras en uppdatering så behöver DISE ändra i flera olika kodbaser. Detta tar onödigt mycket tid när man kan få alla att utgå från samma kod.

2.4 Uppgiften

Uppgiften är att konvertera företagets HTML5/JavaScript kodbas för DISE XPRESS till en Androidkörbar fil (apk-fil). Kod ska skrivas i Javascript, Java samt HTML5. Hur konverteringen från JavaScript/HTML5 till Android-apk fungerar ska undersökas samt vilka olika sätt det kan göras på. Olika operativsystem kan använda sig av olika API:er och eftersom spelaren ska köras på operativsystemet Android så måste ett API eller bibliotek kompatibelt med Android finnas. Uppgiften blir att finna och använda kompatibla API:er för att få tillgång till de Androidspecifika funktionerna som t.ex. att ta skärmdump eller få tillgång till filsystemet.

Förutom kodning så består uppgiften av testning och jämförelser av olika lösningar. Testning av resultatet ska även utvärderas då prestandan på uppspelningen kanske inte blir bra nog. Målet är att alla kraven som listas nedan ska bli uppfyllda. De första kraven är ett måste för att den färdiga produkten klassas som lyckad. De resterande kraven ska uppfyllas i mån av tid och om utvärderingen av resultatet från de huvudsakliga kraven anses som godkända.

Krav:

- Starta automatiskt vid start av enhet (helt utan interaktion av användare).
- Starta om applikation automatiskt vid krasch eller liknande fel.
- Stödja de HTML5/JavaScript funktioner som DISE XPRESS använder.
- Lagra filer lokalt på enheten och fungera även om enheten inte har tillgång till nätverk (både start och uppspelning).
- Spela upp utan interaktion från användaren (auto-play).

- Spela bilder och video blandat och oavbrutet under 24 timmar utan omstart av enhet eller applikation.
- Spara skärmdump (screenshot) samt uppladdning av denna till server.

Önskas om tiden räcker till:

- Göra omladdning/omstart av enhet.
- Kontrollera volym.
- Hantera systeminformation (ledigt diskutrymme, minne, serienummer, modellbeteckning).
- Uppdateringsfunktion.

Om alla kraven går att uppfylla vet inte DISE, men om så är fallet så ska det tas fram vad det är för krav som inte är möjligt att uppfylla samt anledningen till varför det inte är möjligt.

2.5 Extra

I förlängning av det huvudsakliga projektet är DISE intresserade av att konvertera DISE XPRESS till att nå fler plattformar:

- Windows App (8/10).
- Linux (Desktop, Raspberry Pi2, etc.).
- iOS

2.6 Sammanfattning

I detta kapitel har bakgrunden till projektet beskrivits samt relevant information om DISE XPRESS och produkten som arbetet kommer utföras i. Det har pratats om anledningen till varför DISE vill ha XPRESS spelaren körbar på Android och hur det leder till fler potentiella kunder med krav på specifika digitala skärmar. Det har även tagits upp vad projektet

kommer att handla om och kraven som XPRESS spelaren ska klara av för ett lyckat projekt. JavaScript och API användning samt mycket forskning inom området har varit viktiga punkter. Viktiga krav som exempelvis lokal lagring av innehåll samt 24 timmars oavbruten uppspelning har nämnts. Det har även tagits upp vilka utvecklingsområden som DISE är intresserade av att utveckla produkten mot och till vilka andra plattformar de vill expandera produkten till i framtiden.

Kapitel 3

Design

I detta kapitel kommer designen av projektet att förklaras. Konverteringsverktyget som använts kommer att beskrivas samt vilka plugins som använts. Strukturen på koden som skapats och hur de olika delarna hänger ihop kommer även att redovisas för att ge full förståelse av projektet.

3.1 Konverteringen

Projektet var som beskrivits tidigare att få företagets XPRESS spelare körbar på Androidenheter. För att installera en applikation på en Androidenhet så krävs det dock att hela programmet ligger i en apk- fil. Detta kräver därför en konvertering då XPRESS spelaren är skriven i HTML, CSS och Javascript. Verktyget som valdes för konverteringen var Cordova. Med hjälp av Cordovas konvertering skapades alltså en apk- fil som kunde köras på en Androidenhet, dock så kräver XPRESS spelaren att en del ändringar sker i koden för att den ska fungera korrekt. Men först ska verktyget Cordova beskrivas.

3.2 Cordova

Cordova [10] är i grunden ett utvecklingsramverk för mobila applikationer där man vill ge webbutvecklare tillgång till Androidspecifika mobilfunktionerna. Applikationerna från ett

Cordova projekt blir hybrida då de varken är fullt ut mobilappar eller webbappar. De är packade så att funktionaliteten ligger kvar i webb-delen men med tillgång till de mobila nativa API:er. Utvecklaren får själv välja vilka plattformar som ska stödjas efter Cordovaprojektet skapats och kan därefter kompilera koden till dessa applikationer. Cordova kan skapa applikationer för flera plattformar.

Tillgängliga plattformar:

- Amazon-fireos
- Android
- Blackberry 10
- Firefox OS
- iOS
- Ubuntu
- Windows Phone 8
- Windows 8.0, 8.1, 10

Alla plattformar har tillgång till olika många nativa API:er.

Detta projektet har som tidigare sagt utvecklats till Android och skapar därför ett Androidprojekt vid kompilering. Den skapade Androidappen skapar ett nativt WebView objekt som webbapplikationen körs i. WebView objektet fungerar som en webbläsare som körs i appen. För att få en bättre förståelse av Cordova och kopplingarna mellan delarna i applikationen har ett schema över Cordovaprojektet skapats, se *Illustration 2*.

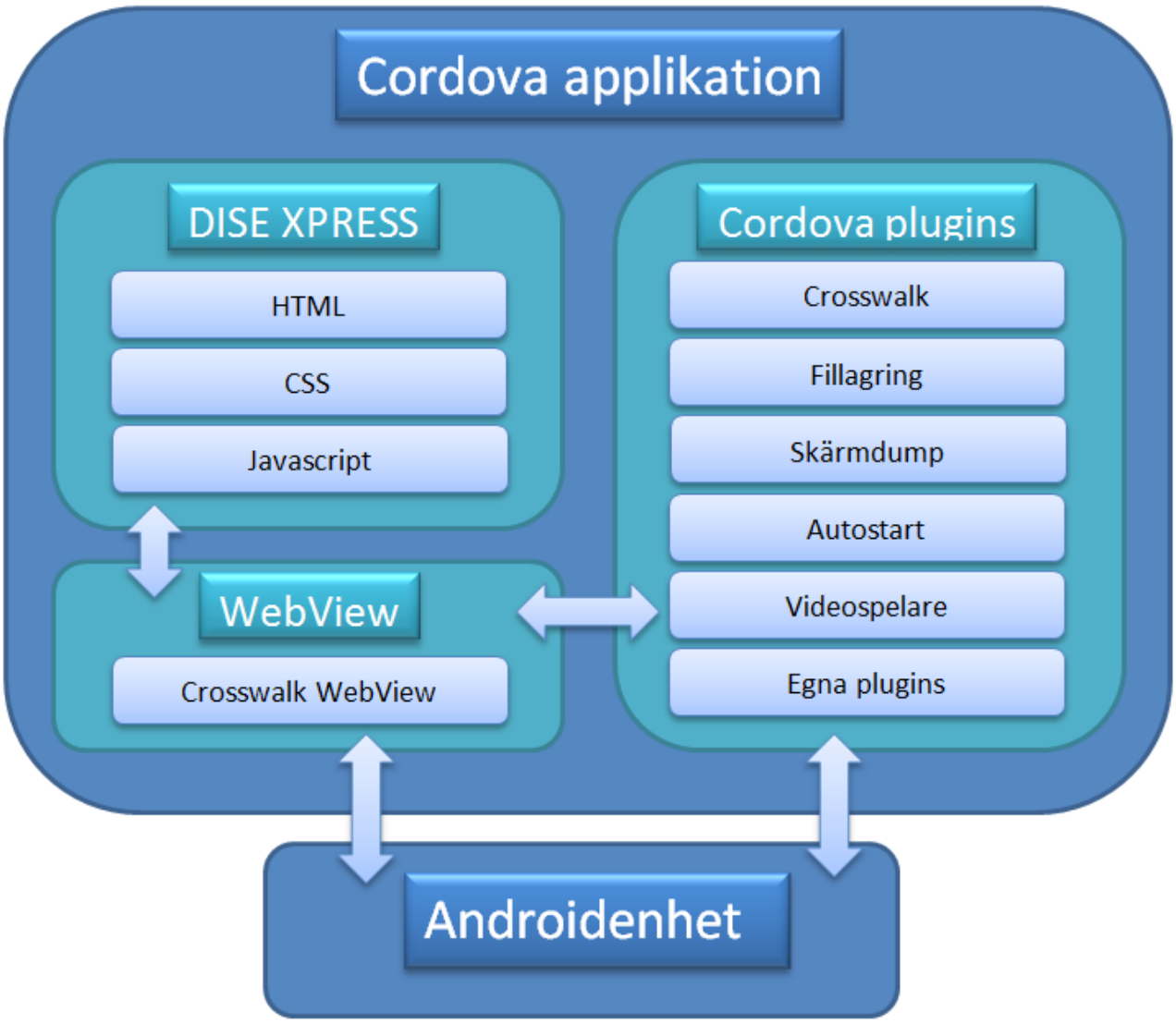


Illustration 2: Schema över projektets Cordova applikation.

3.3 Struktur

Kodbasen som används i WebViewn är DISE XPRESS och har därför mycket funktionalitet i sig som inte kommer användas när spelaren körs i en Androidenhet. Det är funktioner som är specifika för andra plattformar och enheter och som inte är anpassade för Android. Exempel är funktioner för Samsung eller LG skärmar. Nya plattformsspecifika funktioner anpassade för Android har skapats som ska användas när mjukvaran är installerad på en Androidenhet. Men applikationen måste också veta vilken sorts enhet den exekveras i för att använda rätt specifika funktioner vid den specifika installationen, se

Illustration 3.

Det har skapats en egen väg i programmet som tas om programmet känner igen att den exekveras i ett Cordovaprojekt. Programmet exekverar i Cordova projektet om detta returnerar sant:

```
!!window.cordova;
```

Det fungerar på det viset att en fil som heter cordova.js skapas automatiskt i ett Cordova projekt och sätter en global variabel som kallas window.cordova. Denna variabel kan då inspekteras genom att köra dubbla utropstecken (not, not) vilket resulterar i att om window.cordova objektet existerar returnerar den sant, annars falskt.[2] Om sant så väljer programmet att köra de Cordova specifika filer som skapats. En fil som hanterar nedladdningen av det material som ska spelas på skärmen(FileApi_Cordova.js, se Bilaga 5) och en som hanterar resterande funktionalitet för plattformen(Platform_Cordova.js se Bilaga 6). De viktiga delarna i dessa filer kommer förklaras i detta kapitel.

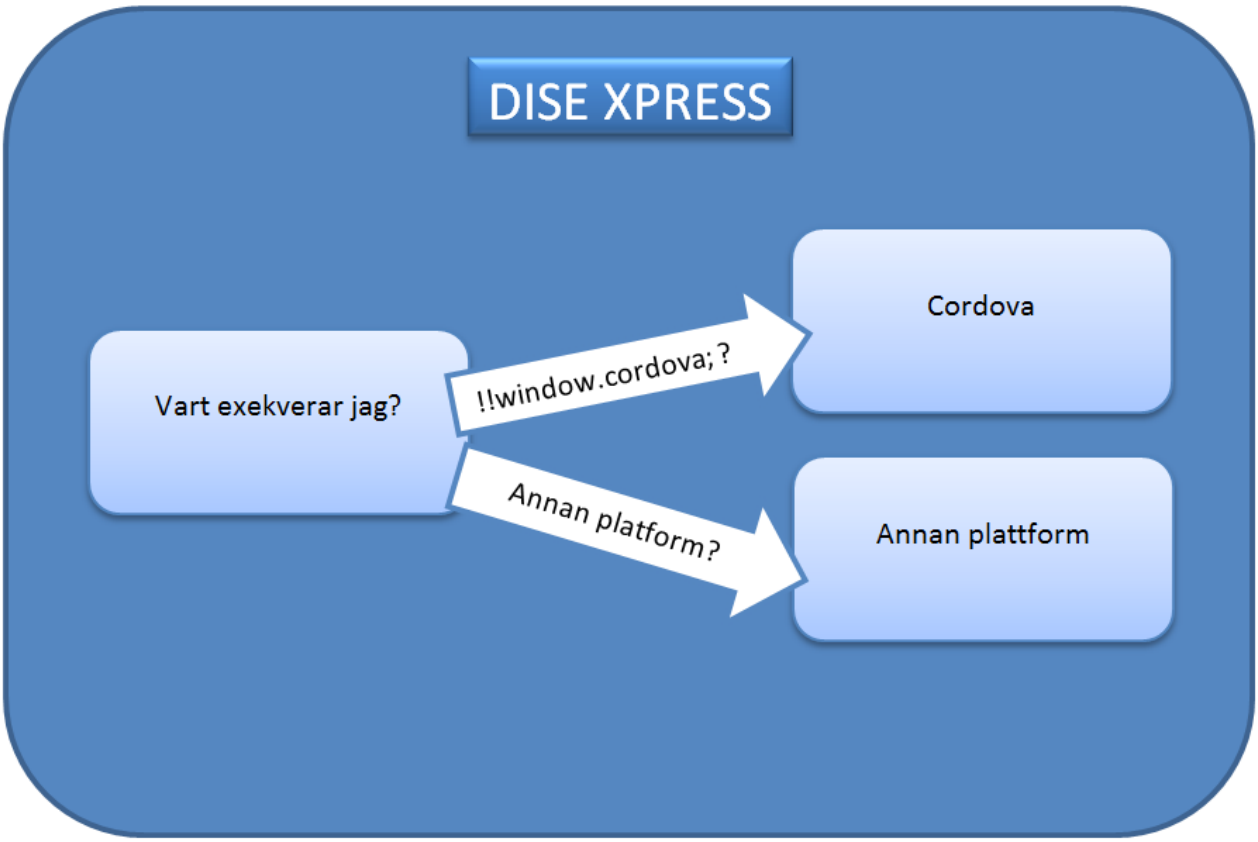


Illustration 3: Val av plattform och dennes plattformsspecifika funktioner.

3.4 Crosswalk

Crosswalk[12] är ett tillägg som ger ett annat alternativ för WebView objektet. Det Crosswalk gör är att den byter ut den nativa webbläsaren i appens WebView till en annan webbläsare som är baserad på Chromium.[11] Valet av Crosswalk är baserade på flera faktorer där huvudorsakerna är bättre prestanda samt funktionalitet för äldre Androidenheter. Eftersom det finns väldigt många olika Androidenheter så reagerar även deras nativa WebViews olika. Det finns olika många APIer tillgängliga, CSS syntaxen kan varierar samt hur applikationen renderas på skärmen. Men med bättre prestanda och stabilitet så tillkommer det även högre mängd minnesanvändning. Apk-filen ökar med ca 17MB och när den installeras på enheten så tar den upp ca 50MB extra minne. Detta utgör dock inte några problem då minnet i Androidenheterna inte ska innehålla något annat än appen och uppspelningsinnehållet och har därför gott om plats.

3.5 Plugins

För att få tillgång till de Androidspecifika funktionaliteterna för den valda plattformen så används plugins. Plugins kopplar de Androidspecifika funktionaliteterna med Javascript genom ett bibliotek av funktioner. Det har använts ett flertal plugins i detta projekt för att uppnå den funktionaliteten som krävs för att DISE XPRESS ska vara körbar på Android. De plugins som använts ska beskrivas i kommande underrubriker.

3.5.1 File / File-transfer

Eftersom ett av kraven för DISE XPRESS är spelning i online samt offline-läge så krävs det att innehållet som ska visas finns lagrad på enheten. Detta är en funktionalitet som måste skrivas för varje specifik plattform eftersom alla laddar ner filer och sparar dem på olika sätt. Nedladdningen av innehållet till Android var inte helt felfri på grund av Cordova, då nedladdningen skulle ske till en Androidenhet men i Javascript kod och inte Java som Androidenheter vanligtvis är utvecklade med. Detta kommer dock tas upp mer i Kapitel 4 där implementationen är i fokus.

Lösningen för nedladdningen blev en omgjord variant av en redan existerande nedladdningsmetod av DISE. Basen kom från deras asynkroniska[13] HTML nedladdning men för att den skulle fungera till en Androidenhet så gjordes det ändringar i några specifika funktioner.

Den huvudsakliga ändringen var i downloadFile funktionen (se *Illustration 4*) som sköter hela nerladdningen. Applikationen hade inte till en början tillåtelse att ladda ner filer till enheten genom Javascript, men med hjälp av en plugin som kallas cordova-plugin-file-transfer kunde detta genomföras.

När en nedladdning ska ske så används FileTransfer och dennes download-funktion. Förutom url-en för innehållet som ska laddas ner så kräver även download-funktionen en extra parameter som ska vara platsen på enheten där filen ska sparas. Tyvärr så fick inte Javascriptet tillåtelse till filkatalogen direkt vilket var anledningen till att ännu en plugin adderades som kallas cordova-plugin-file. Denna plugin gav tillträde till enhetens filkatalog vilket gjorde att en plats i enheten kunde väljas och filen kunde laddas ner.

```
this.downloadFile = function(fileItem, url, doneCallback, progressCallback) {
    var thisThis = this;
    try {
        var fileTransfer = new FileTransfer();
        fileTransfer.onprogress = function(progressEvent) {
            if (progressCallback)
                progressCallback(progressEvent.loaded / progressEvent.total);
        };
        var externalRootDirectory = cordova.file.externalRootDirectory.replace('file://', '');
        fileTransfer.download(url, externalRootDirectory+'/storage/' + fileItem.hash,
            function(fileEntry) {
                doneCallback();
            },
            function(e) {
                doneCallback(new Error(e.exception));
            }
        );
    }
    catch (e) {
        doneCallback(e);
    }
}
```

Illustration 4: Anpassade funktionen för nedladdning av fil till Androidenheten.

Som man även kan se i *Illustration 4* så instansieras ett FileTransfer objekt som sedan också används för att kalkylera framstegen i nedladdningen genom att dela storleken på det som laddas ner på den totala storleken på filen. Sen skapas en variabel med root adressen till Androidenheten som sedan skickas in i nedladdningsfunktionen (fileTransfer.download) plus mappen vi vill lägga filen i (/storage/) plus filens hash(vilket blir filnamnet).

De andra ändringarna som gjordes för själva nedladdningsfunktionerna i filen FileApi_Cordova var att länka till de rätta filerna och sökvägarna i enheten. Som exempelvis funktionen getContentUrl som returnerar sökvägen till den nerladdade filen.

Hela FileApi_Cordova kan ses i Bilaga 5.

3.5.2 Automatisk uppstart

Ett annat av kraven för att få XPRESS spelaren godkänd vid körning på en Androidenhet var att applikationen ska starta automatiskt i följd av start av enheten. Användaren ska alltså inte behöva klicka på ikonen för att starta applikationen, det ska ske automatiskt.

Pluginen som använts för att uppnå detta heter cordova-plugin-autostart och innehåller ett bibliotek med funktioner för automatisk start av applikationen. Genom att addera denna plugin till Cordova projektet gav det möjligheten att kalla på två funktioner från Javascripts koden; *cordova.plugin.autoStart.enable()* samt *cordova.plugin.autoStart.disable().* Genom att kalla på enable så aktiverar vi autostart-funktionaliteten och om vi skulle vilja stänga av den så kallar vi på disable. När autostart är aktiverat så startar applikationen både vid uppstart av enheten, samt efter en automatisk uppdatering av applikationen. Se Bilaga 6 - Platform_Cordova.

3.5.3 Screenshot

Ett annat krav på XPRESS spelaren är att enheten ska kunna ta skärmdumpar och skicka dessa till servern. Detta för att användaren ska kunna se vad som spelas upp på dennes skärm utan att fysiskt behöva titta på skärmen. Användaren ska helt enkelt kunna logga in

och undersöka servern från vilken webbläsare som helst och få skärmdumpar från önskad display.

För att lyckas med denna funktionalitet har en plugin vid namn com.darktalker.cordova.screenshot använts. Genom denna plugin kan vi kalla på två olika sorters skärmdumpar. Den ena tar en skärmdump och sparar den på valt ställe i enheten med valt namn och format(myScreenShot i jpg format i exemplet), se *Illustration 5*. Den andra tar en skärmdump returnerar skärmdumpen som en Data URI, se *Illustration 6*. [3] En Data URI är en kombination av bokstäver som representerar en resurs av något slag och som hänvisar till resursens data.

```
navigator.screenshot.save(function(error,res){
  if(error){
    console.error(error);
  }else{
    console.log('ok',res.filePath); //should be path/to/myScreenshot.jpg
  }
},'jpg',50,'myScreenShot');
```

Illustration 5: Exempelkod för att spara skärmdumpen på enheten.

```
navigator.screenshot.URI(function(error,res){
  if(error){
    console.error(error);
  }else{
    html = '<img style="width:50%;" src="'+res.URI+'">';
    document.body.innerHTML = html;
  }
},50);
```

Illustration 6: Exempelkod för att hämta skärmdumpen som Data URI.

Funktionen för att hämta skärmdumpen som Data URI valdes att användas då vi vill skicka bilden till DISE servern. Funktionen var dock tvungen att modifieras då vi måste skicka

bilden i det format som servern kan ta emot. Som man kan se i *Illustration 7* så ändrar vi format till jpeg och skickar med storleken på bilden till servern. Funktionen getScreenShot kallas av programmet kontinuerligt för att konstant ha en ny bild från skärmen till DISE servern.

```
this.getScreenShot = function(callback) {
    navigator.screenshot.URI(function(error,res){
        if(error){
            console.error(error);
            callback(false);
        }else{
            var stripStr = '';
            var data = res.URI.substr(stripStr.length, res.URI.length);
            callback(data, 'image/jpeg', document.body.offsetWidth, document.body.offsetHeight);
        }
    },50);
};
```

Illustration 7: Skärmdumps-funktion anpassad för DISE XPRESS.

3.5.4 Videospelare

När video ska spelas så skapas ett nativt videospelarobjekt som läser in och spelar upp video. Men när video inte spelas så blir videospelarobjektet svart och måste döljas för att inte blockera synen på bilder och text som inte kräver en videospelare. Anledningen till att ett videospelarobjekt måste skapas är på grund av att Androidenheterna inte har tillräckligt hög prestanda för att ge tillräckligt bra bild och stabilitet på videoklipp som visas direkt igenom WebViewn. Men om man tar och spelar upp videon i enhetens nativa videospelare så blir det bättre flyt och stabilitet i videon. En plugin vid namn com-moust-cordova-videoplayer användes för att få tillgång till videospelaren genom Javascript.

3.6 Omstart vid krasch

Applikationen ska starta om automatiskt om den skulle krascha av någon anledning. Java-kod har därför utvecklas för att känna av en krasch och då starta om applikationen igen.

Lösningen består av en sorts åskådare som konstant tittar på applikationens huvudaktivitet

(MainActivity) för att se om den körs som den ska eller inte. Om den inte körs som den ska så startar åskådaren upp applikationen igen från grunden. Om applikationen körs som den ska så väntar den i fem sekunder och tittar sedan på huvudaktiviteten igen (se *Illustration 9*).

Den så kallade åskådaren skapades i en Android Service och är en speciell Androidklass. En Service är en klass som kan köras utan ett grafiskt användargränssnitt och som även kan köras utan att huvudaktiviteten är aktiv [4]. Service-klassen som använts i det här fallet är en IntentService och betyder att den kör innehållet i en ny tråd och stängs av automatiskt när det har utfört sin uppgift. För att Service-processen inte ska avslutas samtidigt som applikationen kraschar så ges Service-klassen en egen process (se *Illustration 8*).

```
<service android:enabled="true" android:name="StartAppService" android:process=":service_process" />
```

Illustration 8: Deklarationen av Service-klassen.

Funktionen ”isTopScreen” är funktionen som hämtar enhetens alla aktivitetsprocesser och lägger dem i en lista (se *Illustration 10*). Om aktiviteten som är aktiv på skärmen(först i listan) inte är DISE XPRESS applikationen så returnerar funktionen ”false” tillbaka till iterationen i *Illustration 9*. Detta leder till att iterationen avslutas och Service-klassen försöker starta huvudapplikationen. Därefter avslutas Service-klassen automatiskt eftersom den gått igenom all sin kod. Hela Service-klassen kan ses i Bilaga 4.

```
    @Override
    protected void onHandleIntent(Intent intent) {
        Intent launchIntent = getPackageManager()
                .getLaunchIntentForPackage("com.dise.xpress");
        do{
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }while(isTopScreen());

        try {
            launchIntent.addFlags( Intent.FLAG_ACTIVITY_NEW_TASK );
            startActivity(launchIntent);
        } catch (ActivityNotFoundException e) {
            Log.e("---------------------->", "Could not start: "+ e);
        }
    }
}
```

Illustration 9: Service som startar om applikationen vid krasch.

```
private boolean isTopScreen(){
    ActivityManager aManager = (ActivityManager) this.getSystemService(ACTIVITY_SERVICE);
    List<ActivityManager.RunningAppProcessInfo> appProcesses = aManager.getRunningAppProcesses();
    if (appProcesses.get(0).processName.equals("com.dise.xpress")) {
        Log.i("---------------------->", "App is running");
        return true;
    }else{
        Log.i("---------------------->", "App is not running");
        return false;
    }
}
```

Illustration 10: Funktionen som kollar om DISE XPRESS är aktiv på enhetens skärm.

Uppstarten av Service-klassen måste dock ske på rätt sätt från huvudaktiviteten för att kunna fungera som den ska i DISE XPRESS applikationen (se *Illustration 11*). När applikationen startar på nytt så laddar den in sitt innehåll i WebViewn med LoadUrl(launchUrl). OnResume funktionen körs även vid uppstart och där skapas och startas Service-klassen. Om applikationen inte är aktiv på skärmen men inte kraschat så kör den bara onResume funktionen och använder samma URL som tidigare när den blir aktiv igen. Men användandet av den gamla URLen skapar dock problem som gör att innehållet inte spelas upp som det ska. Därför skapas en funktion som avslutar applikationen om det inte är en nystartad uppstart, vilket leder till att nästa omstart som

den nya Service-klassen gör blir en nystart som fungerar korrekt. Se hela MainActivity-klassen i Bilaga 3.

```
boolean firstResume = true;
boolean firstLaunch = true;
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    if(firstLaunch){
        loadUrl(launchUrl);
        firstLaunch = false;
    }else{
        notFirstLaunch();
    }
}
public void notFirstLaunch(){
    this.finish();
    System.exit(0);
}
@Override
protected void onResume() {
    super.onResume();
    Intent intent = new Intent(this, StartAppService.class);
    this.startService(intent);

    if(!firstResume){
        notFirstLaunch();
    }
    firstResume = false;
}
```

Illustration 11: Uppstartning av applikationen och skapandet av Service-klassen.

3.7 Sammanfattning

I det här kapitlet har hela strukturen och designen på projektets presenterats. Hur produkten konverterats till en apk-fil för att kunna köras i Androidenheter har också beskrivits. Verktyget Cordova har förklarats och nämnts igenom hela kapitlet. Illustrationer som beskriver struktur och kopplingar i projektet har visualiseras. Alla plugins, från filhantering till automatisk uppstart av applikationen har presenterats med respektive

funktionalitet och eventuell kod.

Det har även beskrivits hur XPRESS spelarmjukvaran själv kan känna igen vilken plattform den exekveras på och på det viset välja rätt vägar och funktioner i koden.

Kapitel 4

Implementation

I detta kapitel ska tillvägagångssättet för all implementation presenterats. Detta kapitel kommer ha en liknande design som kapitel 3 men innehållet kommer inte handa om slutlig design utan handlar om vägen till den designen och andra försök och tester som valdes att inte användas. Detta kommer beskrivas för att ge en förståelse om varför vissa metoder valdes bort och varför andra bestämdes att användas.

4.1 Konverteringen

Projektets huvudsakliga mål bestod i att konvertera och därmed behövdes ett konverteringsverktyg då företaget ville behålla sin kodbas för XPRESS spelaren. Det var alltså inte någon fråga om att skriva om hela produkten i ett annat språk som var anpassat för Android då det inte var vad företaget ville ha. Detta eftersom de redan hade en sådan lösning och de var inte nöjda med den. De var inte nöjda med den eftersom den inte var baserad på XPRESS spelarens kodbas och behövde därför uppdateras varje gång XPRESS spelaren behövde uppdateras.

Företaget nämnde vid första mötet att verktyget Cordova skulle kunna vara en möjlig lösning. Men att utforska efter andra verktyg ansågs positivt då det skulle kunna leda till

finnandet av ett bättre verktyg. Ett väldigt liknande verktyg som heter Adobe PhoneGap[14] undersöktes i sökandet efter ett konverteringsverktyg. Adobe PhoneGab är väldigt liknande Cordova då det är byggt helt och hållet på Cordova, man kan säga att Cordova är motorn som driver Adobe PhoneGap. De uppmärksammade skillnaderna var att om man sökte mer kontroll över appen rekommenderades Cordova, men om lättarbetad konvertering var det man sökte så var Adobe PhoneGab rekommenderat. Det som Adobe PhoneGap erbjuder utöver Cordova är några förlängningar och kopplingar till Adobe[15], den använder även ett eget CLI (Command Line Interface), har en skrivbordsapp, en utvecklarapp och annat system för kompilering. Eftersom inga av förlängningarna i Adobe PhoneGab lockade för projektet och eftersom Cordova rekommenderades högre på onlineforum så valdes Cordova.

4.2 Cordova

För att lättare kunna installera Cordova installerades Node.js[16] som ger möjligheten att köra kommandot npm från Windows kommandotolk (cmd). För att installera Cordova kördes kommandot:

```
npm install -g cordova
```

Eftersom projektet ska konverteras till en Androidapplikation så behövdes nödvändiga Androidverktyg installeras. Java Development Kit 8[17] (senaste versionen) installerades för att ge möjlighet till att installera Android SDK Tools[18] som är ett måste för att skapa Androidapplikationer.

Innan Cordova-kommandon kunde förstås i kommandotolken så behövdes två Cordova-mappar läggas till som PATH i datorn för att ge den möjlighet att använda de i kommandotolken, nämligen ”platform-tools” och ”tools”. För att ändra PATH i Windows:

Start → PC → Properties → Advanced System Settings → Environment Variables → Path → Edit.

Väl tryckt på ”Edit” kan URL vägar adderas och kommer då vara tillgängliga i kommandotolken (Utfört i Windows 10).

Ett nytt Cordova-projekt skapas genom att köra kommandot:

```
cordova create Cordova com.dise.xpress DiseApp
```

Här döper man mappen för projektet till det som är skrivet efter ”create” nämligen Cordova och installationsplatsen blir arbetskatalogen. Varje Androidapplikation behöver även ett igenkänt id som brukar representeras som en omvänd domän. Det sista i kommandot representerar namnet på appen som skapas. Här döps appen till ”DiseApp”.

Cordova-projektet skapas med flera nödvändiga mappar och filer för att få hybridappen att fungera som den ska. De viktigaste mapparna som skapas är ”www”, ”plugins” och ”platforms”. I www lägger man in all HTML, CSS och JavaScript men även bilder och andra resurser som används för webbappen. I plugins mappen kommer alla plugins som adderas till projektet att placeras. Mappen platforms är den mappen där själva appen och all kod som hör till för respektive plattform placeras. Eftersom detta projekt endast fokuserat på Android så kommer det endast ligga en mapp med Android resurser där, dock inte förrän Android blivit tillagd som plattform med kommandot:

```
cordova platform add android
```

Www mappen fylldes därefter med koden för DISE XPRESS. Endast de nödvändiga filerna fick placeras i www mappen då det annars resulterade i ”error” vid kompilering. Onödiga filer var bland annat komprimerade filer och andra plattformsspecifika filer som inte ska användas för Androidapplikationen.

```
cordova build android
```

När kompileringen och bygget av appen är klar så har en apk-fil skapats som sedan kan installeras på Androidenheter.

4.3 Struktur

Strukturen i DISE XPRESS spelaren är välstrukturerad och genomtänkt, men ändå komplicerad att förstå till en början med endast koden som underlag. En hel del tid lades ner i början av projektet för att hitta relevanta script-filer och funktioner och sedan spåra

dem genom programmet för att försöka förstå strukturen. Väl när koden som kollar nuvarande exekveringsplattform blev förstådd så klarnade det mesta och den övergripande strukturen blev tydlig. Det finns som tidigare nämnts i Kapitel 3 (3.3 Struktur), kod som avgör vilken sorts enhet den exekverar på och kan genom den välja ut de script-filer som ska användas för att allt ska fungera. Det skapades därför en ny if-sats i denna fil som kollade om programmet exekverar på en Androidenhet (kör Cordova versionen). Därefter behövde nya filer med plattformsspecifika funktioner skapas som ska väljas när programmet exekveras från en Androidenhet (Se *Illustration 3* i Kapitel 3).

Implementationen av Cordova-selektionen skapades genom att replikera sättet som de flesta andra plattformsselektioner bestämdes på, vilket var genom att använda Javascripts navigator. Eftersom applikationens WebView kör en webbläsare baserad på Chromium så gav det möjligheten att inspektera applikationen från utvecklingsdatorn. Därmed kunde navigator kommandot köras från konsolen i Chrome för att få information om enheten. Kommandot som användes var:

```
navigator.userAgent
```

Detta returnerar en sträng med information om enheten som bland annat ordet ”Android”. Detta var vad som behövdes då selektionen då kunde bestämmas på samma sätt som andra enheter fast med sökandet efter strängen Android;

```
navigator.userAgent.indexOf(”Android”)
```

Denna kodrad letar i strängen från navigator.userAgent efter ordet ”Android” och returnerar sant om strängen hittats eller falskt om inte. Applikationen kunde nu känna igen om den exekverade på en Androidenhet.

Denna lösning blev dock inte den slutgiltiga lösningen då de erfarna programmerarna på företaget konstaterade att alla enheter som Cordova versionen skulle köras på inte alltid returnerade ordet Android från navigatorn och kommer därför inte fungera på alla enheter. Cordova-versionen ska exempelvis kunna utökas till en annan plattform som exempelvis iOS och skulle därmed inte returnera ”Android”. Det bestämdes därför att ändra till ett annat tillvägagångssätt som kollar om webbapplikationen kör Cordova versionen istället för specifik enhet. Den nya lösningen kollade om applikationen var ett Cordovaprojekt eller

ej och görs med följande kod:

```
!!window.cordova;
```

Förklaringen för hur den nya lösningen fungerar läses i Kapitel 3 (3.3 Struktur).

4.4 Crosswalk

När en apk-fil väl var skapad var nästa steg att anpassa den för Androidenheter. Som nämnts i Kapitel 3 så kräver XPRESS spelaren en del plattformsspecifika funktioner för att kunna utföra all funktionalitet. Eftersom selektionen av plattform inte hade implementerats ännu så känner applikationen inte av att den körs på en Androidenhet utan kan helt enkelt inte hitta någon matchning av vad för enhet den exekverar på. Den valde då standard funktionerna som är anpassade för spelning online i webbläsaren. Men efter kunskapen om detta kunde selektionen bli förstådd och senare utökad för Cordova.

Genom att köra programmet förväntades den kunna spela upp innehåll så länge som enheten har tillgång till internet då innehållet som ska spelas måste konstant strömmas från DISE servern. Dock fungerade det inte utan gav en svart skärm på den Androidenhet som testades. Ett försök gjordes dock på en modernare mobiltelefon (Sony Z2) som även har Android som operativsystem där innehållet fungerade och visades. Men eftersom programmet måste fungera på fler än de nyare Androidenheterna så adderades Crosswalk. Som beskrivits i Kapitel 3 byter Crosswalk bort den nativa webbläsaren i WebVew objektet till en webbläsare baserad på Google Chromium. Fördelarna med Crosswalk har beskrivits i Kapitel 3 (3.4 Crosswalk). Efter detta byte klarade båda enheterna av att spela upp innehållet.

4.5 Plugins

Det stod tydligt redan i början av projektet att det skulle behöva användas plugins för att kunna utföra den mesta funktionaliteten som skulle vara specifik för Android. Det är nämligen den enklaste vägen för att få tillgång till de specifika funktionaliteterna på enheterna.

Vid installation av en plugin användes Cordovas CLI där det enda som behövdes skrivas var ”cordova plugin add” följt av namnet på pluginen eller förvaringsplatsen (exempelvis github-länk). Exempelvis för att installera Crosswalk (som förövrigt även är en plugin).

```
cordova plugin add cordova-plugin-crosswalk-webview
```

Efter installation kan respektive plugins bibliotek och funktioner användas.

4.5.1 File / File-transfer

Plugins löste problemet med nedladdningen av filer till enheten och var en av de momenten som tog längst tid i projektet. Det var nämligen denna funktionalitet som var den första som implementerades efter att applikationen blivit körbar i Androidenheterna. Detta betydde att det krävdes en lång inlärningskurva för att få förståelse för hur layouten på nedladdningen och hur XPRESS spelaren kallade på nedladdningsfunktionen.

Det rekommenderades av uppdragsgivarna att först testa några av deras redan befintliga nedladdningsmetoder. Eftersom spelaren redan kunde köras på flera olika plattformar så hade de ett flertal olika implementationer för nedladdning av filer. Den första som testades var en fil vars nedladdningsmetoder byggde på IndexedDB APIet. IndexedDB[7] är kompatibel med Google Chrome vilket gjorde den lockande då WebViewn som användes körde Chromium på grund av Crosswalk. Men tyvärr var IndexedDB inte kompatibel med Android för Cordova vid tillfället som arbetet tog plats(men blev det senare) och byttes därför ut omgående[8].

Den andra implementationen som testades var en variant som hade en asynkron HTML nedladdning. Denna implementation laddar ner filen och sparar den på enheten som en blob (Binary Large Object). Testningen av denna version såg till en början ut att gå felfri då filerna laddades ner och bilderna visades upp som det skulle vid spelning även i offline-läge, dock spelades inte videoklipp upp. Felsökningen för uppspelningen av video tog upp en hel del tid då ett antagande av orsaken till problemet togs som visade sig vara inkorrekt. Eftersom bild visades under spelningen gjorde antagandet att all nedladdning hade fungerat bra och det var uppspelningen av video som var felet. Men efter flera timmars felsökning så valdes det att skapa en simpel webbserver i appen som skulle

användas som lagringsplatsen för videoklippen. Teorin var att Cordova inte fick tillåtelse att hämta videoklippen från enheten och att webbservern skulle vara en lösning för att förbipassera det hindret. Webbservern löste dock inte problemet, men gav informationen som visade det riktiga problemet. När innehållet laddats ner till webbservern och webbservern inspekterades så stod det tydligt att filerna som lagrats där var 0kb stora. Det var alltså inte uppspelningen som var problemet utan nedladdningen. Beslutet togs därefter att skapa en egen nedladdningsmetod specifik för Codova (FileApi_Cordova, se Bilaga 5) som blev en version lik HTML versionen fast med några specifika funktioner för nedladdning och hämtning av fil-adress. Dessa specifika funktioner skapades med hjälp av de två pluginsen som beskrivs i Kapitel 3 (3.5.1 File / File-transfer). Efter att nedladdningen fungerade behövdes inte webbservern längre och togs därför bort.

4.5.2 Automatisk uppstart

Implementationen av pluginen som gav applikationen automatisk uppstart vid start av enheten var väldigt simpel i jämförelse med de andra pluginsen. Pluginen kallas cordova-plugin-autostart och förklarats i Kapitel 3 (3.5.2 Automatisk uppstart) och krävde endast en aktivering i Javascript för att få funktionaliteten att aktiveras. Aktiveringen valdes att göras i filen Platform_Cordova.js då det är den som ska hålla i de plattformsspecifika funktionerna(Se Bilaga 5).

4.5.3 Screenshot

Implementationen av skärmdumpar gjordes med hjälp av en plugin vid namn com.darktalker.cordova.screenshot och förklaras i Kapitel 3 (3.5.3 Screenshot). Efter att pluginen var installerad kunde skärmdumpar tas från enheten med någon av de exempel som skaparen av pluginen gjort (se illustration 5 och 6 i kapitel 3). Båda varianterna av exempel-funktionerna testades genom att skapa varsin funktion i koden för var och en av varianterna och sedan genom att kalla på funktionerna från konsolen för Javascripts-koden. Båda funktionerna fungerade som det skulle, där den ena sparade bilden i enheten och den andra skapade en URI-länk för bilden.

Målet med skärmdumpen var som nämndes i kapitel 3 (3.5.3 Screenshot) att skicka bilden till servern för att ge användaren en blick på vad som för tillfället visas på displayen utan att behöva se den fysiska skärmen. Funktionen som skickar bilden till servern är

densamma för hela DISE XPRESS eftersom det inte är något plattformsspecifikt. Det räckte därför att anpassa bilden som skapas för funktionen som skickar den till servern för att det skulle fungera. De ändringarna handlade bara om att ändra filformat till jpeg och att göra bilden så stor som möjligt.

4.5.5 Videospelare

Implementationen av videospelarobjektet gjordes för att ge bättre flyt i videoklippen som spelades vilket förklarats i kapitel 3 (3.5.5 Videospelare). Beslutet och implementationen av den nativa videospelaren gjordes av en av företagets utvecklare och kommer därför inte att tas upp mer i detta kapitel. Anledningen till detta kan läsas vidare i kapitel 6 (6.1.2 Arbetsprocessen).

4.6 Omstart vid krasch

Implementationen av omstart vid krasch gjordes i Java kod då ingen plugin med den funktionaliteten hittades. Implementationen gjordes i MainActivity.java (se Bilaga 3) klassen vilket är den klass som startas vid uppstart av applikationen och den klass som skall startas utifall det skulle ske en krasch.

Valet av en lyssnare som fångar upp ouppfångade undantag (uncaughtException) kändes till en början som en självklarhet då en krasch vanligtvis är ett undantag som inte tagits hand om på rätt sätt.

Testningen av den första lösningen gjordes genom att skapa en krasch för att sedan se om applikationen startade upp på nytt eller inte. Kraschen som skapades kastar ett undantag som inte fångas upp i applikationen. Se *Illustration 12*, [5].

```
throw new RuntimeException("This is a crash");
```

Illustration 12: Medveten krasch-kod som användes för testning.

Illustration 13 visar koden som skapades för att fånga upp undantaget och sedan starta upp applikationen på nytt. En ny tråd skapas som lyssnar på uncaughtException vilken kallas på när det sker ett undantag av något slag som inte fångas upp av programmet. Det betyder att när det sker en krasch av Androidapplikationen så ska det fångas upp av uncaughtException som sedan kör funktionen som låter applikationen starta om med hjälp av en sparad aktivitet.

Aktiviteten som ska startas sparas när applikationen startar i ett så kallat intent, vilket är det objekt Android använder för att starta nya aktiviteter. Intentet restartIntent hämtar applikationens startintent, som sedan används för att kunna starta upp rätt aktivitet i applikationen. PendingIntent är det intent som vi sparar vår aktivitet i med hjälp av restartIntent och som används i uncaughtExceptionHandler när vi startar upp applikationen igen (se *Illustration 13*).

```
    Intent restartIntent = this.getPackageManager()
            .getLaunchIntentForPackage(this.getPackageName() );
    intent = PendingIntent.getActivity(this, 0,
            restartIntent,Intent.FLAG_ACTIVITY_CLEAR_TOP);
    Log.i("---------------------->", "Intents saved.");

    Thread.setDefaultUncaughtExceptionHandler(onRuntimeError);
}
private Thread.UncaughtExceptionHandler onRuntimeError= new Thread.UncaughtExceptionHandler() {
    public void uncaughtException(Thread thread, Throwable ex) {
        Log.i("---------------------->", "UncaughtException.");
        AlarmManager mgr = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
        mgr.set(AlarmManager.RTC, System.currentTimeMillis() + 2000, intent);
        System.exit(2);
    }
};
```

Illustration 13: Hanterare för ouppfångade undantag samt sparandet av aktivitet.

DISE XPRESS applikationen har dock den mesta funktionalitet i WebViewn och om det sker ett undantag eller fel av något slag i Javascriptet så lyckas inte tråden som lyssnar på uncaughtException fånga detta. Detta leder tyvärr till att applikationen inte startar om med denna lösning utan dör direkt. Detta problem noterades när lösningen testades mot en minneskrasch skapad i Javascript (se *Illustration 14*). Funktionen är en oändlig iteration som endast allokerar minne.

```
function hangMe() {
    for (;;) {
        var div = document.createElement('div');
        document.body.appendChild(div);
    }
}
```

Illustration 14: JavaScript funktion för att överflöda applikationens minne.

En ny lösning var därför tvungen att skapas som kunde hantera alla sorters kraschar och förklaras i kapitel 3 (3.6 Omstart vid krasch). Implementationen fungerade inte till en början då Service-klassen som användes för att starta upp applikationen dog tillsammans med huvudaktiviteten när den kraschade. En lösning på detta problem gjordes då genom att ge Service-klassen en egen process. Detta gjorde att Service-klassen fortfarande kördes efter att huvudaktiviteten kraschat och kunde då starta upp den på nytt. Hela Service-klassen kan ses i Bilaga 4.

Eftersom den nya lösningen klarade av alla undantag och kraschar så togs den första lösningen bort.

4.7 Problem

Den största delen av projektet har gått till att fixa funktionalitet för Android spelaren, men en väldig stor del har även gått till felsökning. Något som inte har nämnts tidigare i rapporten är att samtidigt som funktionaliteten har implementeras så har även felsökning pågått. Ända sedan Crosswalk pluginen implementerades och spelaren kunde spela upp innehåll så har ett problem existerat. Om applikationen lät stå på och spela upp innehåll i en längre tid så kraschade den till slut. Tiden som det tog för applikationen att krascha varierade på alla olika Androidenheter som testades. Det varierade mellan ca 12 till 48 timmar. Felsökningen bestod av att finna orsaken till kraschen och att fixa den.

Att problemet var en minnesläcka [6] antogs tidigt då alla faktorer i kraschen tydde på det och då Androidenhetens loggning inte gav någon anledning för att den avslutat applikationen. När samma kod itererar om och om igen utan fel så handlar det inte om ett syntaxfel utan feldesign i koden. Programmet tog alltså upp mer och mer minne för varje iteration och som sedan följdes upp av en krasch när Androidenheten tyckte att applikationen tog upp för mycket av dennes minne.

Eftersom applikationen är en Cordova applikation med den mesta funktionaliteten i WebViewn så blev det till en början två olika sorters felsökningar. En för JavaScripts-koden i WebViewn och en för Androidappen i sig. För att se detaljer från minnet för JavaScripts-koden användes inspekteringsfunktionen för Chrome, eftersom denna kunde kopplas till applikationens WebView. För att se minnet från Androidappen användes Android Debug Monitor då denna applikation kunde ge en grafisk bild på minnesanvändningen av de exekverande applikationerna i Androidenheten. Minnet undersöktes noggrant i båda verktygen utan några stora resultat. Alla temporära objekt och värden som skapades kontinuerligt i applikationen såg ut att tas bort som de skulle efter sin användning, då inget objekt kontinuerligt ökade i applikationen. Småsaker som stack ut noteras dock för att till slut kunna finna problemet. En notering som gjordes från inspektionen av minnet i WebViewn var att ett objekt som kallades ”playbackStatistics” togs bort och lades till kontinuerligt under exekvering, dock ökade inte antalet av objekten kontinuerligt.

Eftersom inget fel kunde hittas i inspektionen av minnet så började misstankarna som att det verkligen var en minnesläcka förfrågas. Det skapades därför ett script som kontinuerligt hämtade minnessummor från applikationen på Androidenheten som i sin tur arrangerades i ett Excel dokument för att kunna skapa tydliga grafer. Kommandot som användes för att hämta applikationens minnessummor var:

```
adb -s ENHETENS_IP shell dumpsys meminfo com.dise.xpress >>
C:/temp/mem.log
```

Utifrån graferna var det enkelt att se att minnet för applikationen kontinuerligt ökade och det var som tidigare förutspått ett minnesproblem.

Nästa steg som valdes att göras var att stänga av all funktionalitet i applikationen. Inget innehåll visades eller någon statistik eller liknande skapades utan applikationen kördes utan att något förutom själva uppspelnings-iterationen gjordes. När en graf på detta sedan skapades så ökade inte minnet i applikationen utan den stannade på samma värden hela tiden. Sista steget som behövde göras var då att börja lägga tillbaka små bitar av funktionaliteten tills minnesproblemet väl kom tillbaka.

Turligt nog var statistikmetoden den första funktionalitet som valdes att startas igen i första testet då författaren tyckte den var misstänksam från tidigare felsökningar. Minnesproblemet uppstod nämligen direkt efter statistiken startades. Ett sista test gjordes där all funktionalitet var aktiverad förutom statistiken för att försäkra att det var den som var orsaken till problemet, vilket den också var.

Statistiken som orsakade minnesproblemet var en funktionalitet som existerade i XPRESS spelaren innan Cordova projektet började och var enligt företagets utvecklare onödig för just detta Cordova projekt och kunde därför inaktiveras permanent. En genomgång av koden för statistikfunktionen gjordes med en erfaren utvecklare på företaget, dock utan något resultat av vart i funktionen som minnesläckan inträffade. Men på grund av låg prioritet av företaget på att hitta vart i koden som felet låg, så gick projektet vidare med utveckling efter detta.

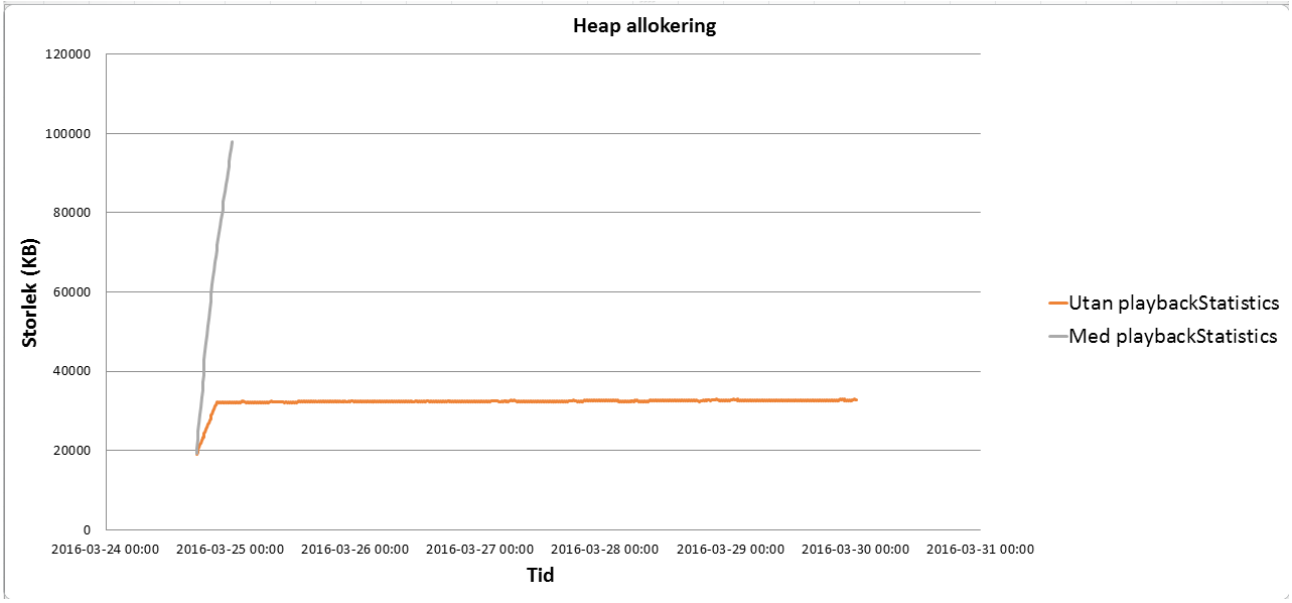


*Illustration 15: M*innesallokeringen med samt utan funktionen som orsakade minnesläckan.

Illustrationen 15 visar grafer på minnesallokeringen för DISE XPRESS i två olika enheter. Den ena visar körningen av applikationen utan statistikfunktionen(orange). Den andra visar en graf för minnesallokeringen när statistikfunktionen kördes(grå).

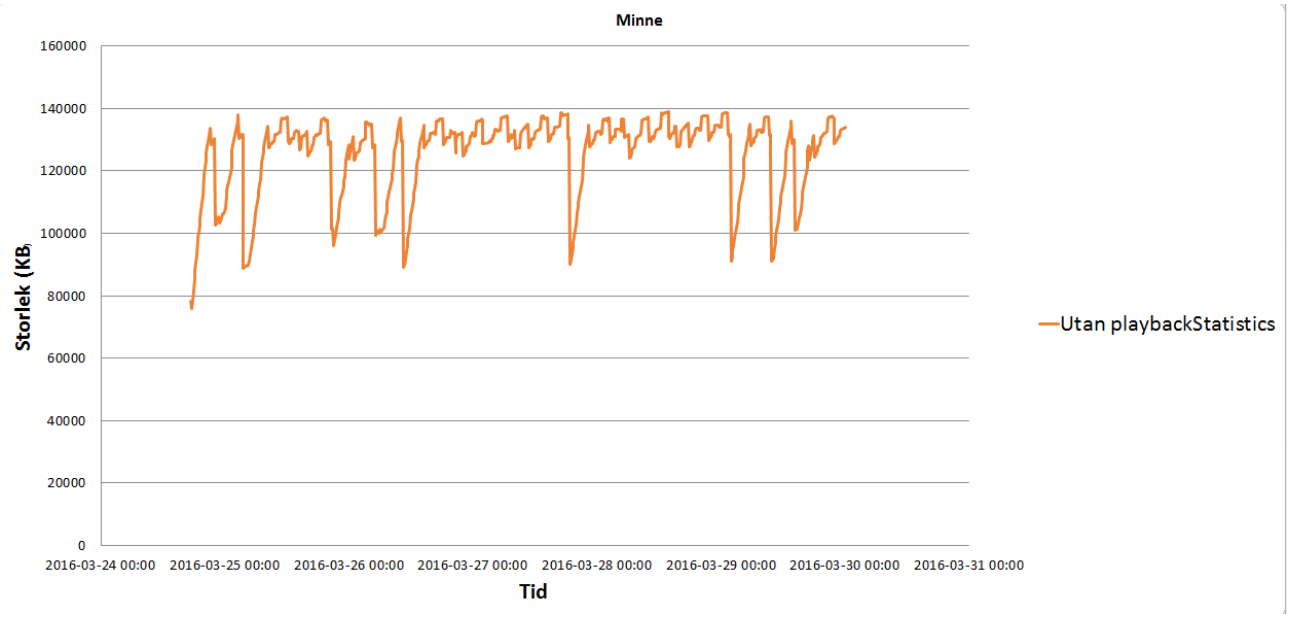


Illustration 16: Minnesanvändningen från DISE XPRESS applikationen.

Illustrationen 16 visar den faktiska minnesanvändningen för samma enhet som från *Illustration 9*. Detta för att visa att minnet inte kontinuerligt ökar. Anledningen till att minnet går mycket upp och ner är på grund av applikationens skräpsamlare (garbage collector). Dessa grafer (*Illustration 15* och *Illustration 16*) kan även ses i bilagorna med högre kvalitet (Bilaga 1 och Bilaga 2).

4.8 Sammanfattning

Kapitel 4 har nu gått igenom implementationen av projektets alla delar. Valet av konverteringsverktyg har tagits upp och varför det var Cordova som valdes. Hur Cordova applikationen skapats har visats samt hur installationen av plugins för projektet gått till. Implementationen och och en genomgång på strukturen har redovisats samt hänvisningar till relevanta sektioner och illustrationer från kapitel 3 har refererats till. En stor del av kapitel 4 har varit en redovisning på hur de plattformsspecifika funktionaliteterna implementerats. Det kan konstateras att många funktionaliteter blivit implementerade med hjälp av plugins. Till sist så har även den viktiga punkten angående felsökning av minnesläckan redovisats och även visats med grafer.

Kapitel 5

Resultat och utvärdering

I detta kapitel kommer resultat på både design och implementation presenteras samt en utvärdering på resultatet i jämförelse med förväntningarna. Punkterna från kravspecifikationen kommer tas upp och jämföras med resultatet för att få en helhetsbild på resultatet i allmänhet.

5.1 Skall kraven

I projektets specifikation fanns det tydliga skall krav, vilka var de kraven som krävdes för att projektet och produkten skulle klassas som lyckade. Det fanns även önskade krav vilka var kraven som skulle implementeras i mån av tid om projektet blivit lyckat med tid över. Projektet och kraven kan läsas i kapitel 2 (2.4 Uppgiften).

5.1.1 Starta automatiskt vid start av dator

En Androidenhet startar inte applikationer automatiskt utan kräver en knapptryckning på applikationens ikon för att starta den. Detta hade dock företaget som ett krav då en skärm som kör deras produkt alltid ska ha applikationen spelandes, och om skärmen inte körs så ska inte en fysisk knapptryckning behövas. Förväntningen på detta krav var att det skulle

vara möjligt då företaget nämnt att det är möjligt på de flesta andra plattformar och att Android inte borde vara annorlunda. Implementationen av denna funktionalitet blev också lyckad och kan läsas i kapitel 4 (4.5.2).

5.1.2 Starta om app automatiskt vid krasch eller liknande fel.

Om applikationen skulle krascha av någon anledning så ska inte Androidenhetens hemskärm eller ett error-meddelande visas på skärmen då det ser oprofessionellt ut för alla åskådare som kollar på skärmen samt så tappar skärmen sin funktionalitet. Förväntningen om en lyckad implementation blev även uppfylld, dock så var det en mer avancerad implementation i jämförelse med vissa andra krav. Detta kan läsas mer om i kapitel 4 (4.6 Omstart vid Krasch).

5.1.3 Stödja de HTML5/JavaScript funktioner som DISE XPRESS använder.

Detta krav handlar om att de funktioner som DISE XPRESS spelaren använder i form av HTML5 och JavaScript kod även ska fungera på Androidversionen som skapas i projektet. Det betyder att de vill behålla kod-basen för XPRESS spelaren och endast utveckla den för att fungera på Android. Kravet löstes med att göra ett Cordova projekt som använder sig av en WebView för att köra JavaScript samt HTML kod. Kravet blev uppfyllt och implementationen kan läsas i kapitel 4 (4.1 Konverteringen och 4.2 Cordova).

5.1.4 Lagra filer lokalt på enheten och fungera även om enheten inte har tillgång till nätverk.

När skärmen inte ska spela upp dynamiskt innehåll utan bara visar samma information om och om igen så ska inte den behöva en internetuppkoppling för att göra detta. Enheten ska därför spara innehållet den laddar ner på enheten lokalt och sedan kunna spela upp det utan internetuppkoppling. Detta krav förväntade sig att lyckas då man visste att det ska gå att ladda ner och använda lagrad information på en Androidenhet. Dock var detta svårare än planerat eftersom nedladdningen och uppspelningen inte skedde direkt från

Androidapplikationen utan genom WebViewn och JavaScript. Men i slutändan blev även detta krav uppfyllt och implementationen kan läsas i kapitel 4 (4.5.1 File / File-transfer).

5.1.5 Spela upp utan interaktion från användaren.

Detta krav handlar om att alla videoklipp som spelas ska göra det utan att en fysisk knapptryckning ska behöva göras för att få videon att starta. Tar man upp ett videoklipp från en webbläsare så kräver den ofta att användaren trycker på spela upp knappen innan videon börjar rulla. Detta blev dock aldrig ett problem när uppspelningen av video valdes att göras i en nativ videospelare. Detta kan läsas mer om i kapitel 4 (4.5.5 Videospelare).

5.1.6 Spela bilder och video blandat och oavbrutet under 24 timmar utan omstart av enhet eller applikation.

Detta krav var det krav som till en början inte förväntades vara ett problem men som blev det största problemet i hela projektet. Uppspelningen borde inte haft några problem om det inte vore för en minnesläcka som orsakade applikationen att krascha under 24 timmars uppspelning. Detta lyckades ändå lösas efter mycket tid och kan läsas om i kapitel 4 (4.7 Problem).

5.1.7 Spara skärmdump samt uppladdning av denna till server.

Detta krav var det kravet där tidsåtgången på implementationen blev som förväntad. Kravet uppfylldes vilket gjorde att bilder (skärmdumpar) kontinuerligt skickas till servern så att användaren kan se det som visas på skärmen utan att behöva stå fysiskt framför den. Kravet uppfylldes och implementationen kan läsas mer om i kapitel 4 (4.5.3 Screenshot).

5.2 Önskade krav

De önskade kraven färdigställdes inte då det inte fanns tid nog. Det var huvudsakligen felsökningen av minnesläckan som tog upp mer tid än förväntat, men det resulterade ändå i en funnen minnesläcka i deras kod som funnits där sedan innan projektets början. Det kan ses som bättre än att implementera en önskad funktion.

Det önskade kravet om att hantera systeminformation var dock implementerat under projektets gång men av en annan programmerare som var anställd på företaget. Men mer om det och varför kommer tas upp i projektutvärderingen i nästa kapitel.

5.3 Sammanfattning

I det här kapitlet har resultatet presenterats i form av uppfyllda samt inte uppfyllda krav. Varje skall krav från projekt specifikationen har presenterats med resultat samt hänvisning till respektive detaljerad implementation från kapitel 4. Anledningen till tidsbristen och bristen på uppfyllda önskade krav har även förklarats.

Alla krav som var ett måste för att resultera i ett lyckat projekt har blivit uppfyllda vilket resulterar i att projektet blev lyckat.

Kapitel 6

Slutsats

I detta kapitel kommer projektet i allmänhet att summeras och värderas. Vad i projektet som gick bra samt mindre bra kommer presenteras samt vad som skulle göras annorlunda om projektet skulle göras om på nytt. Vilka erfarenheter som lärts under projektet gång samt allmänna kunskapar som kommer vara användbara för framtida projekt kommer nämnas.

6.1 Sammanfattning

Jag kommer nu förklara och sammanfatta arbetsprocessen och vad för erfarenheter och kunskaper som skapats från starten till slutet av projektet. Sammanfattningen delas upp i tre steg där introduktionen, arbetsprocessen och till sist mina avslutande tankar kommer att beskrivas.

6.1.1 Introduktionen

Arbetsprocessen under projektet har varit varierande och givande på många sätt. Det allra första arbetet som gjordes var att förbereda en arbetsmiljö på företaget. Innan kodningen kunde börja så behövdes program installeras och användarkonton skapas. Det första som

gjordes var att jag blev inbjuden till företagets Github projekt. För att kunna genomföra projektet behövdes Git och TortoiseGit för att kunna skapa en så kallad Git Clone av deras Github projekt. TortoiseGit är ett verktyg som ger ett grafiskt användargränssnitt vid användningen av Git. Eftersom jag inte hade någon stor erfarenhet i användandet av ett versionshanteringssystem(Git) sedan tidigare så var det väldigt givande. Det gav mig god erfarenhet för framtiden då de flesta programmeringsprojekt använder ett versionshanteringssystem.

Efter inlärningen av versionshanteringssystemet så tilldelades hårdvara som skulle användas för testning. Hårdvaran bestod av två olika Androidenheter och en skärm som kunde kopplas till enheterna. En genomgång på hur kompileringen av koden samt hanteringen av koden gavs även av uppdragsgivarna. För att summera den introduktionen så var det en väldigt givande introduktion för mig som inte tidigare haft några större erfarenheter av att jobba som utvecklare i ett riktigt företag.

6.1.2 Arbetsprocessen

Till en början arbetade jag mycket själv då jag hade fria händer med allt i projektet vad gällande valet av konverteringsverktyg samt utvecklingsmiljö. Det blev helt enkelt en hel del forskande på nätet efter bästa möjliga verktyg, men också upplärning i användandet av kommandon i kommandotolken. Det fanns inga tidigare kunskaper inom utveckling mot Androidenheter från kommandotolken så Android Debug Bridge(adb) kommandon och övriga kommandon blev erfarenheter som utvecklades utmed hela projektets gång.

Eftersom projektet huvudsakligen handlade om att konvertera deras projekt, eller rättare sagt tillåta deras projekt att exekveras på en Androidenhet utan att mista funktionalitet så var det inte så mycket kod som behövde skrivas. I efterhand skulle jag tolka projektet som ett integrations-projekt. Projektet hade till en viss del kodning i sig, men det handlade mestadels om att lägga till rätt plugins och anpassa allt för exekvering på en Androidenhet. Angående nya erfarenheter så var det inte kod-skrivning som vägde tyngst utan

integration och erfarenheten med att jobba och kommunicera på en arbetsplats. Men en av de större erfarenheterna som jag utvecklat under projektets är själva felsökningen. Felsökning pågick konstant under projektets gång, ända från användandet av kommandon och kommunikationen med Androids debug-verktyg (adb) till felsökningen av minnesläckan som förklarats i kapitel 4 (4.7 Problem). Efter varje oklart error så krävdes det en felsökning som i sin tur gav kunskap som inte tidigare fanns och som kommer göra framtida projekt enklare och bättre.

En annan del i arbetsprocessen som blev något oväntat var projektets plötsliga ökning i prioritet av företaget. Tidigt in i projektets gång så började kunder till DISE trycka på för att få se och testa den nya Androidversionen. Detta gjorde att mer tid var tvungen att läggas på projektet för att ge kunderna en duglig första version. Utöver att jag spenderade mer tid till projektet så sattes även en annan utvecklare från företaget på att hjälpa till med projektet. Det som gjordes utan min inblandning var bland annat implementationen av videospelaren, som jag nämnde i kapitel 4 (4.5.5 Videospelare) samt några felsökningar av minnesläckan. Dock gav det mig enormt mycket kunskap då jag fick möjligheten att arbeta tillsammans med en erfaren utvecklare. Jag fick möjligheten att parprogrammera samt bolla idéer och lösningar med någon med betydligt högre kunskap än mig själv, vilket bara var positivt och lärorikt. Jag fick även en genomgång av företagets utvecklare om något skett utan min inblandning gällande Androidprojektet.

6.1.3 Avslutande tankar

Som nu nämnts i detta kapitel så har projektet varit väldigt givande och lyckat. Alla huvudsakliga krav har blivit uppfyllda och resultatet ser lovande ut för framtiden. Det finns en del funktionaliteter som kommer adderas innan projektet är redo att släppas som färdig produkt till kunder, men grundstenarna är nu färdiga. Framtida funktionaliteter kommer bland annat bestå av volymkontroll och funktion för omstart av Androidenheten.

Jag är väldigt nöjd med projektet och tycker resultatet blev lika bra som jag hade hoppats på innan utvecklingen började. Men det finns alltid småsaker man kan kunde förbättra om man skulle göra projektet igen. Jag skulle bland annat se till att jag förstod hur deras kod

fungerade helt och hållet innan jag började försöka implementera egen kod i den. Det var nämligen svårt att kunna hitta lämpliga ställen att addera kod på, vilket gjorde att implementationen tog längre tid än nödvändigt. Det tog även lång tid att sätta sig in i all den redan existerande koden då projektet är väldigt stort och komplext. Om jag hade gjort om projektet igen hade jag frågat företaget om en genomgång av strukturen på koden eller bett om en översiktlig dokumentation över koden för att snabbare kunna hitta kopplingarna mellan alla funktioner och filer i programmet. Men i det stora hela är jag väldigt nöjd och jag tror inte att jag hade kunnat gjort det mycket bättre en andra gång förutom på kortare tid.

Referenser

[1] DISE. Om företaget. https://www.dise.com/about-us/, Januari 2016.

[2] Damien Antipa. Känn igen Cordova projekt. http://damien.antipa.at/blog/2014/02/08/3-ways-to-detect-that-your-application-is-running-in-cordova-slash-phonegap/, Februari 2016.

[3] gitawego. Skärmdumps plugin. https://www.npmjs.com/package/com.darktalker.cordova.screenshot, Mars 2016.

[4] Android. Service-klass. http://developer.android.com/guide/components/services.html, April 2016.

[5] Crashlytics. Tvinga en Androidkrasch. http://support.crashlytics.com/knowledgebase/articles/112848-how-do-i-force-a-crash-using-the-android-sdk, Mars 2016.

[6] Diverse. Minnesläcka. https://en.wikipedia.org/wiki/Memory_leak, Maj 2016.

[7] Mozilla. IndexedDB. https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API, Februari 2016.

[8] Cordova. Fillagring. https://cordova.apache.org/docs/en/5.4.0/cordova/storage/storage.html, Februari 2016.

[9] DISE. Produkter. https://www.dise.com/products/, Januari 2016.

[10] Cordova. Cordova. https://cordova.apache.org/, Februari 2016.

[11] Chromium. Chromium https://www.chromium.org/, Februari 2016.

[12] Crosswalk. Cordova plugin. https://github.com/crosswalk-project/cordova-plugin-crosswalk-webview, Februari 2016.

[13] Diverse. Asynkron kommunikation. https://sv.wikipedia.org/wiki/Asynkron_kommunikation, Mars 2016.

[14] Adobe Phonegap. Adobe Phonegap. http://phonegap.com/, Februari 2016.

[15] Adobe. Adobe. http://www.adobe.com/se/, Februari 2016.

[16] Node.js. Node.js. https://nodejs.org/en/, Februari 2016.

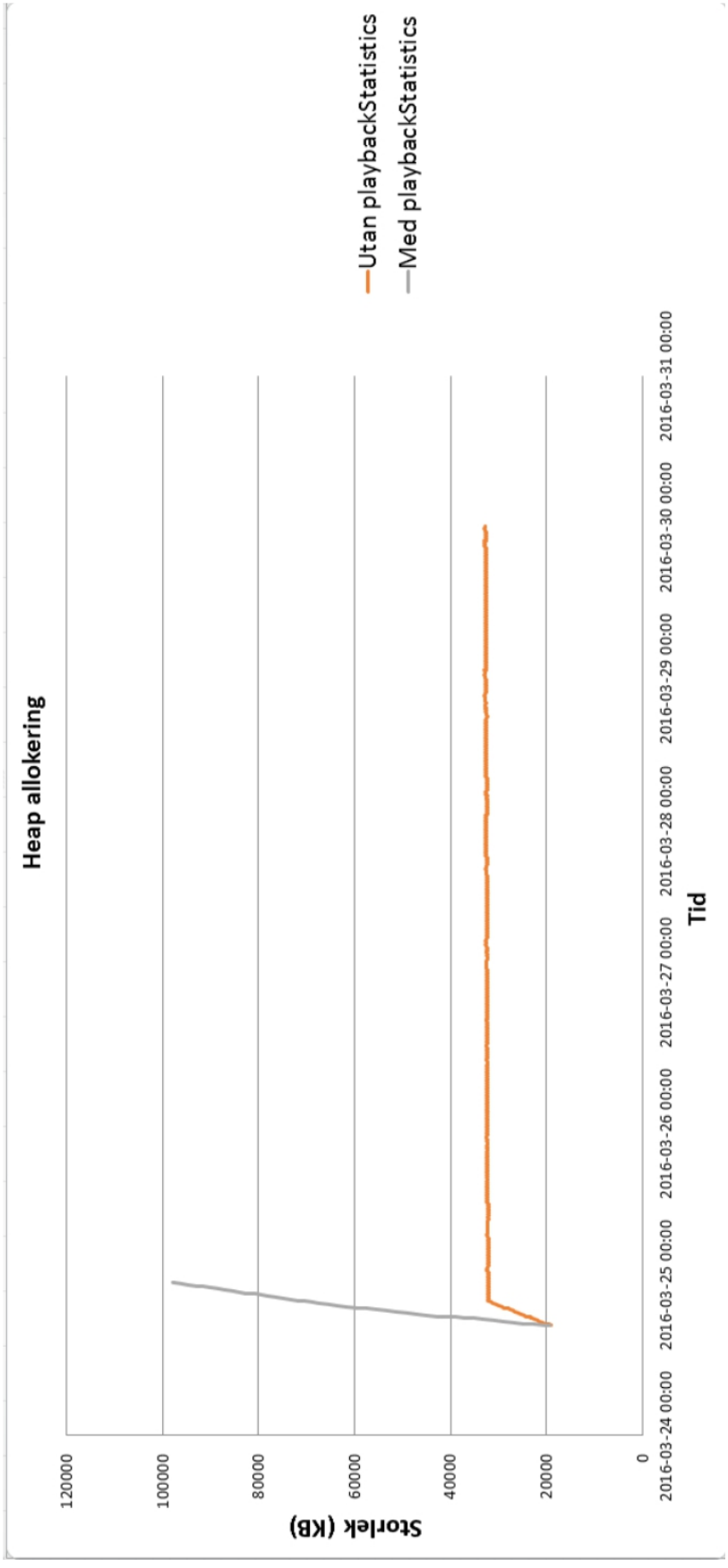
[17] Oracle. Java Development Kit 8. http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, Februari 2016

[18] Android. Android SDK Tools. https://developer.android.com/studio/releases/sdk-tools.html, Februari 2016.

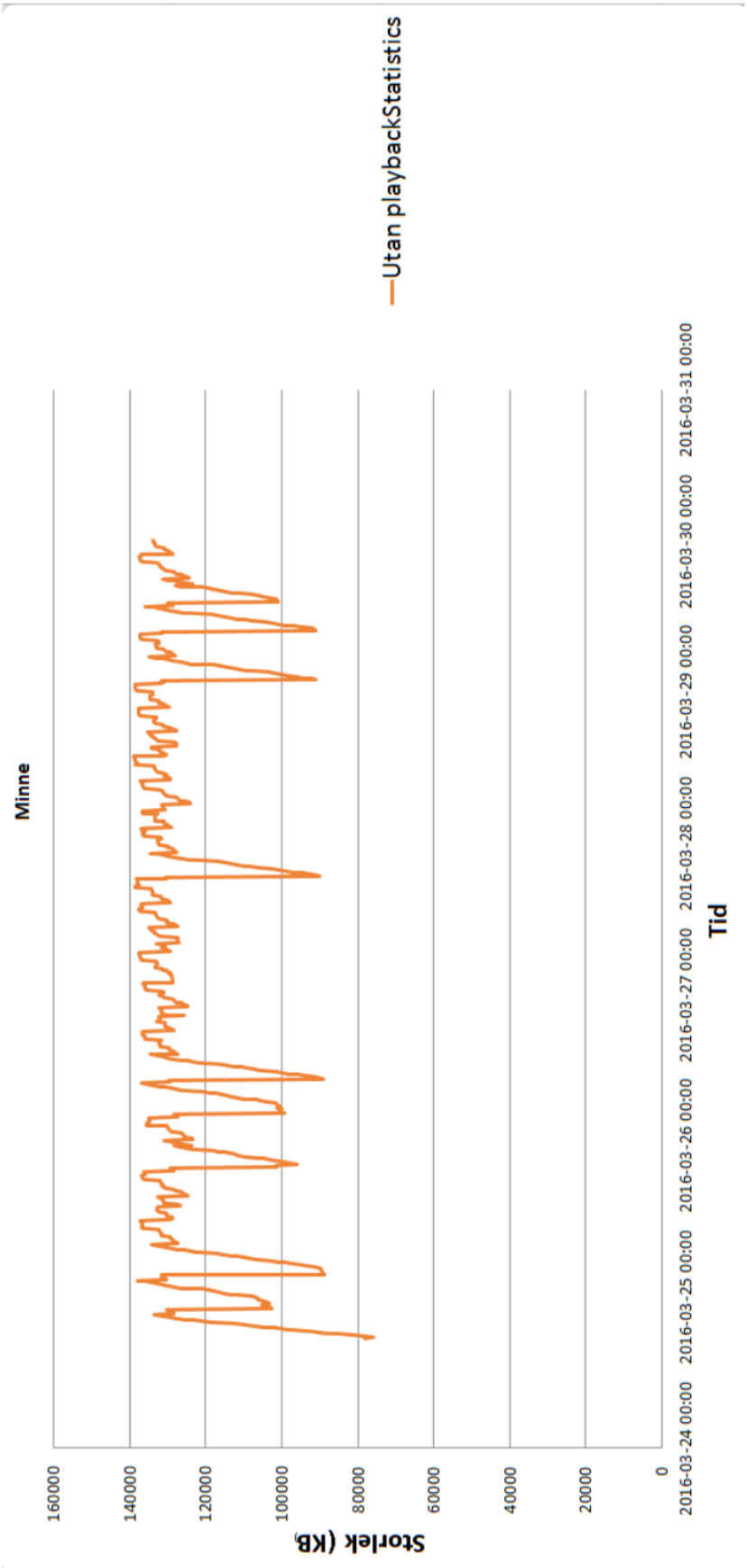
[19] Diverse. Digital skyltning. https://sv.wikipedia.org/wiki/Digital_skyltning, Juni 2016.

Bilagor

Bilaga 1 – Heap allokering



Bilaga 2 - Minne



Bilaga 3 - MainActivity

```
public class MainActivity extends CordovaActivity
{
    boolean firstResume = true;
    boolean firstLaunch = true;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        if(firstLaunch){
            loadUrl(launchUrl);
            firstLaunch = false;
        }else{
            notFirstLaunch();
        }
    }
    public void notFirstLaunch(){
        this.finish();
        System.exit(0);
    }
    @Override
    protected void onResume() {
        super.onResume();
        Intent intent = new Intent(this, StartAppService.class);
        this.startService(intent);

        if(!firstResume){
            notFirstLaunch();
        }
        firstResume = false;
    }
}
```

Bilaga 4 - StartAppService

```
public class StartAppService extends IntentService {

    public StartAppService() {
        super("StartAppService");
    }

    private boolean isTopScreen(){
        ActivityManager aManager = (ActivityManager)
this.getSystemService(ACTIVITY_SERVICE);
        List<ActivityManager.RunningAppProcessInfo> appProcesses =
aManager.getRunningAppProcesses();
        if (appProcesses.get(0).processName.equals("com.dise.xpress")) {
            Log.i("---------------------->", "App is running");
            return true;
        }else{
            Log.i("---------------------->", "App is not running");
            return false;
        }
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Intent launchIntent = getPackageManager()
                .getLaunchIntentForPackage("com.dise.xpress");
        do{
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }while(isTopScreen());

        try {
            launchIntent.addFlags( Intent.FLAG_ACTIVITY_NEW_TASK );
            startActivity(launchIntent);
        } catch (ActivityNotFoundException e) {
            Log.e("---------------------->", "Could not start: "+ e);
        }
    }
}
```

Bilaga 5 – FileApi_Cordova

```
"use strict";

function FileApi_Cordova() {
    var fs_ = null;
    var contentDir_ = null;
    this.instance = '';
    var that = this;

    self.persistentStorage = navigator.webkitPersistentStorage ||
navigator.persistentStorage;
    self.requestFileSystem = self.requestFileSystem ||
                             self.webkitRequestFileSystem;

    this.initMainThread = function(callback) {
        persistentStorage.requestQuota(0, function(size) {
            callback();
        }, function(e) {
            log("Error requesting storage quota: " + e.message, LogLevel.ERROR);
        });
    }

    this.init = function(callback, instance) {
        that.instance = instance || '';
        requestFileSystem(1, 0, function(fs) {
            fs_ = fs;
            if (fs_) {
                persistentStorage.queryUsageAndQuota(
                    function(currentUsageInBytes, currentQuotaInBytes) {
                        log("Storage quota: "+(currentQuotaInBytes / 1024 /
1024).toFixed(2)+" MB. "+(currentUsageInBytes / 1024 / 1024).toFixed(2)+" MB used.");
                        fs_.root.getDirectory("/storage"+that.instance, {create:
true, exclusive: false}, function(directoryEntry){
                            contentDir_ = directoryEntry;
                            callback(true);
                        }, onError);
                    }, function() {
                        log("Unable to get storage quota!", LogLevel.ERROR);
                    }
                );
            }
            else {
                log("Failed to create file system object!", LogLevel.ERROR);
            }
        }, onError);
    }

    function onError(error) {
        log(error.message+". "+JSON.stringify(error), LogLevel.ERROR);
    }

    this.downloadFile = function(fileItem, url, doneCallback, progressCallback) {
        var thisThis = this;
        try {
```

```
            var fileTransfer = new FileTransfer();
            fileTransfer.onprogress = function(progressEvent) {
                if (progressCallback)
                    progressCallback(progressEvent.loaded / progressEvent.total);
            };
            var externalRootDirectory =
cordova.file.externalRootDirectory.replace('file://', '');
            fileTransfer.download(url, externalRootDirectory+'/storage/' +
fileItem.hash,
                function(fileEntry) {
                    doneCallback();
                },
                function(e) {
                    doneCallback(new Error(e.exception));
                }
            );
        }
        catch (e) {
            doneCallback(e);
        }
    }

    this.getContentAsString = function(fileItem, doneCallback) {
        try {
            contentDir_.getFile(fileItem.hash, {create: false}, function(fileEntry) {
                var fileReader = new FileReader();
                fileReader.onload = function(e) {
                    doneCallback(null, fileReader.result);
                }
                fileReader.onerror = function(e) {
                    doneCallback(e, null);
                }
                fileEntry.file(function(file) {
                    fileReader.readAsText(file);
                }, function(e) {
                    doneCallback(e, null);
                });
            });
        }
        catch (e) {
            log("GetContentAsString() failed: "+e.message+". Hash: "+fileItem.hash,
LogLevel.ERROR);
            doneCallback(e, null);
        }
    }

    this.getContentUrl = function(fileItem, doneCallback) {
        try {
            //doneCallback(null, 'http://127.0.0.1:8080/'+fileItem.hash);
            doneCallback(null,
cordova.file.externalRootDirectory+'storage/'+fileItem.hash);
        }
        catch (e) {
            log("GetContentUrl() failed: "+e.message+". Hash: "+fileItem.hash,
LogLevel.ERROR);
            doneCallback(e, null);
        }
```

```
}

this.clearAll = function(doneCallback) {
    var ret = true;
    try {
        var dirReader = contentDir_.createReader();
        readDirectoryEntries(function(err, entries) {
            async.each(entries, function(entry, callback) {
                entry.remove(callback, callback);
            }, function() {
                if (doneCallback)
                    doneCallback();
            });
        });
    }
    catch (e) {
        log("ClearAll() failed: "+e.message, LogLevel.ERROR);
        if (doneCallback)
            doneCallback(e);
    }
}

function readDirectoryEntries(callback) {
    var dirReader = contentDir_.createReader();
    var entries = [];
    var readEntries = function() {
        dirReader.readEntries(function(results) {
            if (!results.length) {
                // Done
                callback(null, entries);
            }
            else {
                var count = results.length;
                for (var i=0; i<count; i++) {
                    entries.push(results[i]);
                }
                readEntries();
            }
        }, function(e) {
            callback(e, null);
        });
    };
    readEntries();
}

this.getFileList = function(doneCallback) {
    var dirReader = contentDir_.createReader();
    readDirectoryEntries(function(err, entries) {
        var availableFiles = [];
        var count = entries.length;
        for (var i=0; i<count; i++) {
            var entry = entries[i];
            availableFiles.push(entry.name);
        }
        doneCallback(null, availableFiles);
    });
}
```

```
    this.fileExists = function(file, doneCallback) {
        var dirReader = contentDir_.createReader();
        readDirectoryEntries(function(err, entries) {
            var availableFiles = [];
            var count = entries.length;
            for (var i=0; i<count; i++) {
                var entry = entries[i];
                availableFiles.push(entry.name);
            }
            doneCallback(null, (availableFiles.indexOf(file) >= 0));
        });
    }


    this.deleteFile = function(file, doneCallback) {
        contentDir_.getFile(file, {create: false}, function (fileEntry) {
            fileEntry.remove(
                function() {
                    doneCallback(null, file);
                },
                function(e) {
                    // Error
                    doneCallback(e, file);
                }
            );
        },
        function (e) {
            // Error
            doneCallback(e, file);
        });
    }

    this.getFreeStorageSpace = function(callback) {
        callback(null);
    }

    this.getUsbDeviceList = function(callback) {
        if (callback) {
            callback(null, []);
        }
    };

    this.getFileList2 = function(path, callback) {
        if (callback) {
            callback(null, []);
        }
    };

    this.readFile = function(path, callback) {
        if (callback) {
            callback('Not Implemented', '');
        }
    };
}
```

Bilaga 6 - Platform_Cordova

```
"use strict";

function Platform_Cordova() {
    var serverUtcTimeDiff_;
    var timeZone_;
    var nativeMediaPlayerStarted_ = false;
    var nativeMediaPlayerPlaylist_ = [];
    var hasNativeMediaPlayerSupport_ = !!(device && (device.platform === 'Android'/* ||
device.platform === 'iOS'*/) && VideoPlayer);
    var nativeMediaPlayerForceMode_ = null;
    var nativeMediaPlayerLastStartOptions_ = {};
    var networkInfo_ = null;
    var that = this;
    var hardwaitInitOk_ = false;

    this.setMute = function(mute) {
    };

    this.getMute = function() {
        return false;
    };

    this.setVolume = function(volume) {
    };

    this.getVolume = function(volume) {
        return 100;
    };

    this.registerAllKeys = function() {
    };

    this.registerAllKeys = function() {
    };

    this.turnScreenOn = function() {
        document.body.style.visibility = 'inherit';
        console.warn('Display -> Show');
    };

    this.turnScreenOff = function() {
        document.body.style.visibility = 'hidden';
        console.warn('Display -> Hide');
    };

    this.blockNavigation = function() {
    };

    this.getCurrentTime = function() {
        if (typeof serverUtcTimeDiff_ != 'undefined') {
            var ret = new Date();
            var tzOffset = ret.getTimezoneOffset() * 60 * 1000; // We get this to
reverse compensate for the "built in" time zone. (This offset might be wrong, so we
```

```
cannot use it to calculate time)
                switch (timeZone_) {
                    default: break;
                }
                var m = moment().utc().subtract(serverUtcTimeDiff_,
'milliseconds').tz(timeZone_ || 'UTC'); // Get the server adjusted UTC time and convert
it to the correct time zone
                ret.setTime(((m.unix() + m.utcOffset() * 60) * 1000) + tzOffset); //
Create a normal Date object. Unix timestamp + time zone offset + Date "built in" time
zone compensation
                return ret;
            }
            else {
                return new Date();
            }
        };

        this.setServerUtcTime = function(serverUtcTime, serverUtcTimeLocalSet) {
            var dt = new Date();
            var tzOffset = dt.getTimezoneOffset() * 60 * 1000;
            dt.setUTCFullYear(serverUtcTime.substr(0, 4));
            dt.setUTCMonth(parseInt(serverUtcTime.substr(5, 2), 10)-1);
            dt.setUTCDate(serverUtcTime.substr(8, 2));
            dt.setUTCDate(serverUtcTime.substr(8, 2));
            dt.setUTCHours(serverUtcTime.substr(11, 2));
            dt.setUTCMinutes(serverUtcTime.substr(14, 2));
            dt.setUTCSeconds(serverUtcTime.substr(17, 2));
            dt.setTime(dt.getTime() + tzOffset);
            var localSetTime = new Date();
            localSetTime.setTime(serverUtcTimeLocalSet.getTime() + tzOffset);
            serverUtcTime = dt;
            serverUtcTimeDiff_ = localSetTime.getTime() - serverUtcTime.getTime();
        };

        this.setTimeZone = function(timeZone) {
            timeZone_ = timeZone;
        };

        this.getHardwareInfo = function(discovery) {
            return {
                available: device.available,
                cordova: device.cordova,
                isVirtual: device.isVirtual,
                manufacturer: device.manufacturer,
                model: device.model,
                platform: device.platform,
                serial: device.serial,
                uuid: device.uuid,
                version: device.version,
                networkInfo: discovery ? null : networkInfo_
            };
        };

        this.shouldInstallDspk = function(callback) {
            callback(false);
        };
```

```
    this.installNewDspk = function(done) {
        done(true);
    };

    this.reboot = function() {
        if (location) {
            // Fake reboot
            location.reload();
        }
    };

    this.getScreenShot = function(callback) {
        try {
            navigator.screenshot.URI(function(error,res){
                if(error){
                    console.error(error);
                    callback(false);
                }else{
                    var stripStr = '';
                    var data = res.URI.substr(stripStr.length, res.URI.length);
                    callback(data, 'image/jpeg', document.body.offsetWidth,
document.body.offsetHeight);
                }
            },50);
        } catch (e) {
            log('getScreenShot() failed: '+e.message, LogLevel.ERROR);
            callback(false);
        }
    };

    this.enableDebug = function() {
    };

    this.isKey = function(event, keyName) {
        var keyCode = event.keyCode;
        if (!keyCode) {
            if (event.key.length == 1) {
                keyCode = event.key.charCodeAt(0);
            } else {
                switch (event.key) {
                    case 'Escape': keyCode = 27; break;
                }
            }
        }
        switch (keyName) {
            case "toggleInfoOverlay": return (keyCode == 73 && event.ctrlKey) ||
keyCode == 167 || keyCode == 220; // Ctrl + i or ยง (ยง == 220 for Chrome 167 for
FireFox)
            case "checkConnection":   return (keyCode == 50 && event.altKey);  // Alt
+ 2
            case "toggleFullScreen":  return (keyCode == 84 && event.altKey);  // Alt
+ t
            case "reload":            return ((keyCode == 82 && event.altKey) ||
keyCode == 116);  // Alt  + r / F5
            case "forward":           return (keyCode == 39);                  //
right
            case "backward":          return (keyCode == 37);                  // left
```

```
            case "rightArrow":      return (keyCode == 39);                 //
right
            case "leftArrow":       return (keyCode == 37);                 // left
            case "downArrow":       return (keyCode == 40);                 // up
            case "upArrow":         return (keyCode == 38);                 // down
            case "enter":           return (keyCode == 13);                 //
enter
            case "loadTestingPage": return false;                           // Not
implemented!
            case "blanker":         return false;                           // Not
implemented!
            case "audioMute":       return false;                           // Not
implemented!
            case "volumeUp":        return false;                           // Not
implemented!
            case "volumeDown":      return false;                           // Not
implemented!
            case "clickFooter":     return (keyCode == 53);                 // 5
            default:                return false;
        }
    };

    this.nativeMediaPlayer = {};
    this.nativeMediaPlayer.addPlaylistItem = function(url) {
        nativeMediaPlayerPlaylist_.push(url);
        return true;
    };

    this.nativeMediaPlayer.clearPlaylist = function() {
        nativeMediaPlayerPlaylist_ = [];
        if (nativeMediaPlayerStarted_)
            that.nativeMediaPlayer.stop();
        return true;
    };

    this.nativeMediaPlayer.start = function(options) {
        options = options || {};
        if (!hasNativeMediaPlayerSupport_)
            return false;
        nativeMediaPlayerLastStartOptions_ = options;
        var opt = {
            volume: options.volume || 1.0,
            scalingMode: options.scalingMode || VideoPlayer.SCALING_MODE.SCALE_TO_FIT
        }
        function playMedia() {
            if (!nativeMediaPlayerStarted_)
                return;
            var url = nativeMediaPlayerPlaylist_.shift();
            if (!url)
                return;
            console.log('VideoPlayer.play()', url, opt);
            VideoPlayer.play(
                url, opt,
                function cordovaPlayerSuccess() {
                    console.log('VIDEO PLAY DONE SUCCESS');
                    if (options.success)
                        options.success(url);
```

```
                    // Play next item
                    playMedia();
                },
                function cordovaPlayerError(err) {
                    if (err) {
                        log('Native media player error: ' + err, LogLevel.ERROR);
                    }
                    if (options.error)
                        options.error(url, err);
                    // Play next item
                    playMedia();
                }
            );
        }
        nativeMediaPlayerStarted_ = true;
        playMedia();
        return true;
    };

    this.nativeMediaPlayer.stop = function(options) {
        options = options || {};
        if (!hasNativeMediaPlayerSupport_)
            return false;
        nativeMediaPlayerStarted_ = false;
        console.log('VideoPlayer.close()');
        VideoPlayer.close(
            function cordovaPlayerSuccess() {
                console.log('VIDEO STOP DONE SUCCESS');
                if (options.success)
                    options.success();
            },
            function cordovaPlayerError(err) {
                if (err) {
                    log('Native media player error: ' + err, LogLevel.ERROR);
                }
                if (options.error)
                    options.error(err);
            }
        );
        return true;
    };

    this.nativeMediaPlayer.next = function() {
        return false;
    };

    this.nativeMediaPlayer.isStarted = function() {
        return nativeMediaPlayerStarted_;
    };

    this.nativeMediaPlayer.shouldUseNativeMediaPlayer = function() {
        if (hasNativeMediaPlayerSupport_ && nativeMediaPlayerForceMode_ !== null)
            return nativeMediaPlayerForceMode_;

        return hasNativeMediaPlayerSupport_;
    };
```

```
    this.nativeMediaPlayer.forceUseNativeMediaPlayer = function(forceMode) {
        nativeMediaPlayerForceMode_ = forceMode === null ? null : forceMode == true;
        log('Native media player force mode set to: '+(nativeMediaPlayerForceMode_ ===
null ? 'Default(Not forced either on or off)' : nativeMediaPlayerForceMode_ === false ?
'Force Off' : 'Force On'));
    };

    this.playerUnload = function() {
    };


    this.getStatusInfo = function(callback) {
        callback({});
    };

    this.toggleOrientation = function() {
        log('Toggle orientation not supported on this platform');
    };

    this.doFirmwareUpgrade = function(data) {
        log("Firmware upgrade not supported on this platform");
    };

    this.applyDownloadedFirmware = function (firmwareUpgradeData) {
        log("Firmware upgrade not supported on this platform");
    };

    this.isEmulator = function() {
        return false;
    };

    this.getName = function() {
        return 'Browser';
    };

    this.addImeHandlers = function() {
    };

    // Startup
    try {
        networkinfo.getNetworkInfo(
            function(networkInfo) {
                networkInfo_ = networkInfo;
            },
            function(err) {
                console.log('Failed to get network info', err);
            }
        );
        cordova.plugins.autoStart.enable();
        cordova.plugins.insomnia.keepAwake();
    } catch (e) {
        console.error('Cordova platform startup failed! ', e);
    }

    this.waitForHardwareReady = function waitForHardwareReady(doneCallback,
infoCallback) {
        if (hardwaitInitOk_)
            return doneCallback();
```

```
            hardwaitInitOk_ = true;
            return doneCallback();
    }

    this.startExternalInput = function startExternalInput(url, domNode, uniqueId,
callback) {
        if (callback)
            return callback();
    }

    this.runExternalInput = function stopExternalInput(url, domNode, uniqueId) {
    }

    this.stopExternalInput = function stopExternalInput(url, domNode, uniqueId,
callback) {
        if (callback)
            return callback();
    }
}
```