FH JOANNEUM (University of Applied Sciences)

**Usage possibilities of WebRTC in a cross-platform developed mobile app**

**Bachelor Thesis**

**submitted in conformity with the requirements  
for the degree of  
Bachelor of Science in Engineering (BSc)**

Bachelor’s degree program **Internettechnik**FH JOANNEUM (University of Applied Sciences), Kapfenberg

**supervisor:** Dipl. Ing. Johannes Feiner

**submitted by:** Michael Stifter  
**personal identifier:** 1310418054

06 / 2016

**Obligatory signed declaration:**

I hereby declare that the present Bachelor’s thesis was composed by myself and that the work contained herein is my own. I also confirm that I have only used the specified resources. All formulations and concepts taken verbatim or in substance from printed or unprinted material or from the Internet have been cited according to the rules of good scientific practice and indicated by footnotes or other exact references to the original source.

The present thesis has not been submitted to another university for the award of an academic degree in this form. This thesis has been submitted in printed and electronic form. I hereby confirm that the content of the digital version is the same as in the printed version.

I understand that the provision of incorrect information may have legal consequences.

Michael Stifter Graz, 10.06.2016

**Table of contents**

[Abstract 6](#_Toc453231380)

[Kurzfassung 7](#_Toc453231381)

[1 Introduction 8](#_Toc453231382)

[2 WebRTC 10](#_Toc453231383)

[2.1 History 10](#_Toc453231384)

[2.2 Architecture 10](#_Toc453231385)

[2.3 Components 11](#_Toc453231386)

[2.3.1 MediaStream 11](#_Toc453231387)

[2.3.2 PeerConnection 13](#_Toc453231388)

[2.3.3 DataChannel 14](#_Toc453231389)

[2.4 Protocols 14](#_Toc453231390)

[2.5 Functionality 15](#_Toc453231391)

[2.6 Signalling 16](#_Toc453231392)

[2.7 Advantages 18](#_Toc453231393)

[2.8 Limitations 19](#_Toc453231394)

[2.9 Potential applications 21](#_Toc453231395)

[2.9.1 Real-time communication 21](#_Toc453231396)

[2.9.2 Peer-to-peer file sharing 22](#_Toc453231397)

[2.9.3 Integrating real-time sensor data 22](#_Toc453231398)

[2.10 Usage possibilities on mobile devices 23](#_Toc453231399)

[2.10.1 Web applications 23](#_Toc453231400)

[2.10.2 Native app 24](#_Toc453231401)

[2.10.3 Native app with web views 24](#_Toc453231402)

[2.10.4 Cross-platform developed mobile app 24](#_Toc453231403)

[2.11 Conclusion 25](#_Toc453231404)

[3 Cross-platform mobile development 27](#_Toc453231405)

[3.1 Background 27](#_Toc453231406)

[3.2 User experience 28](#_Toc453231407)

[3.2.1 Context 28](#_Toc453231408)

[3.2.2 Implementation 29](#_Toc453231409)

[3.3 Advantages 29](#_Toc453231410)

[3.4 Limitations 30](#_Toc453231411)

[3.5 Approaches 32](#_Toc453231412)

[3.5.1 Web apps 32](#_Toc453231413)

[3.5.2 Hybrid apps 32](#_Toc453231414)

[3.5.3 Interpreted apps 33](#_Toc453231415)

[3.5.4 Generated apps 33](#_Toc453231416)

[3.6 Criteria for choosing a framework 33](#_Toc453231417)

[3.6.1 Support for mobile platforms 33](#_Toc453231418)

[3.6.2 Access to device-specific features 34](#_Toc453231419)

[3.6.3 Legal background 34](#_Toc453231420)

[3.6.4 WebRTC capability 35](#_Toc453231421)

[3.6.5 Side issues 35](#_Toc453231422)

[3.7 Cross-platform development frameworks 35](#_Toc453231423)

[3.7.1 Apache Cordova (PhoneGap) 35](#_Toc453231424)

[3.7.2 Xamarin 36](#_Toc453231425)

[3.7.3 Appcelerator Titanium 36](#_Toc453231426)

[3.7.4 Ionic 36](#_Toc453231427)

[3.7.5 Sencha Touch 37](#_Toc453231428)

[3.7.6 Other frameworks 37](#_Toc453231429)

[4 Prototype development 38](#_Toc453231430)

[4.1 Introduction 38](#_Toc453231431)

[4.1.1 Original state of the prototype server 38](#_Toc453231432)

[4.1.2 Necessary steps to communicate with the prototype server 38](#_Toc453231433)

[4.2 Things to consider when developing WebRTC for cross-platform app 38](#_Toc453231434)

[4.3 CONCRETE SOLUTION FOR DEMO IMPLEMENTATION? 38](#_Toc453231435)

[4.4 Implementation of web app in web view 38](#_Toc453231436)

[4.5 Implementation of cross-platform mobile apps 38](#_Toc453231437)

[4.5.1 Crosswalk 39](#_Toc453231438)

[4.5.2 OpenWebRTC 39](#_Toc453231439)

[4.6 Insights 39](#_Toc453231440)

[5 Evaluation 42](#_Toc453231441)

[5.1 Setup 42](#_Toc453231442)

[5.2 Method 42](#_Toc453231443)

[5.3 Results 42](#_Toc453231444)

[6 Outlook 43](#_Toc453231445)

[6.1 The future of WebRTC 43](#_Toc453231446)

[6.2 User management and authentication 43](#_Toc453231447)

[6.3 Multi-user sessions 44](#_Toc453231448)

[7 Conclusion 45](#_Toc453231449)

[List of tables 46](#_Toc453231450)

[List of figures 47](#_Toc453231451)

[List of listings 48](#_Toc453231452)

[List of abbreviations 49](#_Toc453231453)

[Bibliography 51](#_Toc453231454)

Abstract

Kurzfassung

# Introduction

Over the last years, Web Real-Time Communication (WebRTC) has seen a significant rise in popularity especially in browser-based web applications, and is expected to continuously grow over the following years[[1]](#footnote-1). It offers numerous advantages for communication technology because it enables real-time communication in web browsers without any additional plugins or software, which was not possible before the introduction of WebRTC (cf. Stifter 2016, p. 23). Apart from that, it uses peer-to-peer connections, where the data is transferred directly without a third-party server involved, which results in decreased network latency. However, its biggest disadvantage to date is the fact that not all web browsers support WebRTC, although the number of supporting browsers has been continuously rising over the last years, as discussed in Chapter 2.8.

This poses a problem for developers who want to use WebRTC in web applications today, because it is impractical to require users to switch to a web browser that fully supports WebRTC. On mobile devices, however, the situation is different. Besides web apps, there is also the possibility to use native apps. Native apps have become massively popular over the last years and are able to deliver substantial advantages when it comes to user experience (cf. Cailean 2013). This stems from the fact that it is possible to integrate and access many components of the user’s device, such as the list of contacts, the calendar and various sensors into an application with ease. While it is possible to develop a native app that uses WebRTC, it also increases the development effort considerably, since it is necessary to implement the same functionality on multiple platforms, such as Android, iOS and Windows Phone.

A solution to this problem could be the use of a suitable cross-platform development framework that facilitates the use of WebRTC. With a cross-platform developed mobile app it is possible to develop the application only once, instead of multiple times for each platform it should run on. However, since WebRTC is a technology that can be considered relatively new and is still under development, it is not guaranteed that cross-platform development frameworks fully support the latest version of WebRTC.

This thesis takes a deeper look into popular cross-platform mobile development frameworks and examines them on their ability to support current versions of WebRTC. To analyze this examination, a set of criteria is defined in order to identify suitable frameworks for developing cross-platform apps that use WebRTC.

# OUTLOOK TO RESULTS

The thesis is structured as follows: The first part discusses the history and functionality of WebRTC, together with its benefits and shortcomings. The second part mentions various ways of implementing a mobile app and highlights the advantages and disadvantages of each method in detail. In a third step, the possibilities of using WebRTC on mobile devices are addressed. Following that, the essential insights with regard to the implementation of a reference app are pointed out. Chapter 5 discusses the evaluation process and its results. The final section concludes the thesis by summarizing the essential findings and suggesting possibilities to expand the underlying work.

# WebRTC

“Web Real-Time Communication (WebRTC) is a new standard that lets browsers communicate in real time using a peer-to-peer architecture” (Loreto & Romano 2014, p. vii). This technology development is particularly promising since it enables real-time telecommunication applications within web browsers, without the need of third-party extensions or plugins. Furthermore, WebRTC is open source software, which means that the entire source code is publicly available. This is beneficial for developers because they can get a full understanding of the inner functionality and, additionally, the code can be extended and improved by an online community.

The following chapter provides an overview of the history and functionality of WebRTC, along with its benefits and limitations. The end of chapter takes a look on the possibilities of using WebRTC outside of web browsers.

## History

WebRTC started as a project conducted by Google. The first time that it was presented to the public was in May 2011. Later that year, the company decided to publish the entire source code under a permissive Berkeley Software Distribution (BSD) license, enabling the internet community to contribute ideas to the project. At the same time, in November 2011, the first rudimentary version of WebRTC was added to the Google Chrome browser. At the beginning of 2013 the technology passed an important milestone, as Mozilla published its implementation of WebRTC in their Firefox browser and it was possible to start peer-to-peer connections from Chrome to Firefox and vice versa. In that same year, both companies also released the first mobile versions of their browsers supporting WebRTC (cf. WebRTC Tutorial 2014).

## Architecture

WebRTC is built upon a rather complex architecture, which can be interacted with through an Application Programming Interface (API) with a simple set of functions. An outline of the whole architecture is illustrated in Figure 1. On top of it stands the Web Application API, which is written in JavaScript and can be accessed in a standard web page in a browser. This is the only layer that developers have to work with, while all other architecture layers fulfil their tasks independently upon requests on this top layer. The Web Application API interacts with the internal WebRTC API, which is written in C++. The internal API is responsible for the handling of PeerConnections and their session management (cf. Stifter 2016, p. 22).

The concern of the following layer is the management of media-related components. This includes the codecs of audio and video engines, echo cancellation, image enhancement and, most importantly, the correct synchronization of media tracks in order to provide a valuable user experience. The layer below handles the capturing of audio and video streams, and is therefore directly communicating with the lowest level of the architecture, which is the physical device hardware, e.g. cameras and microphones that are built-into or attached to the device (cf. Grigorik 2013, p. 311).



Figure 1: WebRTC architecture (Grigorik 2013, p. 311)

## Components

WebRTC is based upon three different components, namely MediaStream, PeerConnection and DataChannel. While the first two are mandatory, DataChannel is an additional optional component. The components are described in more detail in the following part.

### MediaStream

The MediaStream interface is responsible for everything related to streaming of audio and video. It can hold any number of MediaStreamTracks. In a traditional video conferencing scenario, this would be one video track and two audio tracks, a left and a right channel (see Figure 2). However, developers have the option to add or remove tracks (cf. Loreto & Romano 2014, p. 12).



Figure 2: A WebRTC MediaStream object that contains one video and two audio tracks (Loreto & Romano 2014, p. 13)

To create a MediaStream object, developers can use navigator.getUserMedia() API, which obtains access to media equipment attached to the user’s device. This typically includes cameras and microphones. Listing 1 describes a simple example on how to request access to an audio and video stream.

When this request is issued in line 21, the web browser informs the user about the request from the web page. The user can then grant access to the requested media sources or deny it. In other words, it is necessary to explicitly get the user’s approval for using the device’s media components. Furthermore, browsers display a red recording icon on top of the web page’s tab to indicate that it has currently access media resources. This adds an additional security layer on behalf of the user, since it is not possible to use the media devices without knowledge and consent from the user.

If the request was approved, the browser tries to access the requested resources and subsequently calls the success callback function described in line 8. In case the user denied the request or if an error occurred during the initialization stage, the error callback function in line 16 will be called.



Listing 1: Simple example for requesting access to camera and microphone of user device

### PeerConnection

A PeerConnection object in WebRTC “represents an association with a remote peer, which is usually another instance of the same JavaScript application running at the remote end” (Loreto & Romano 2014, p. 7). In other words, it holds the actual WebRTC peer-to-peer connection between two users. It is responsible for managing every aspect of the connection, from initialization to teardown. The developer is only required to implement the initial startup and the termination of a connection, the management part in between is automatically handled by the WebRTC API (cf. Leaver, Iwase & Katsura 2015).

The initialization of a PeerConnection is accomplished over a signalling channel, which is usually JavaScript code inside the web page. The data transfer at this stage is handled by the web server. The whole signalling process is mentioned in more detail in Chapter 2.6. When the PeerConnection between two users has been successfully established, it is possible for both parties to exchange MediaStream objects. This could mean, for instance, that they can now see and talk to each other in a video chat directly from browser to browser (cf. Loreto & Romano 2014, p. 7f).

### DataChannel

The DataChannel API is the only optional component and is therefore not required to be implemented by the developer. Its purpose is to provide an additional communications layer, in which developers can send arbitrary data between the two users. One PeerConnection object can hold any number of DataChannels. The API functions of DataChannels were modelled after the ones from WebSockets and resemble them closely (cf. Leaver, Iwase & Katsura 2015).

The main configuration options for a DataChannel is its priority inside the PeerConnection and if the messages should be delivered in reliable or unreliable mode. In reliable mode, messages sent over the DataChannel are guaranteed to be delivered in the order they were sent, adding some administration overhead to the data transfer, which might result in slower transmission times. On the other hand, in unreliable mode this overhead is not included, thus resulting in faster transmission without guaranteeing successful delivery (cf. Ristic 2014).

## Protocols

WebRTC uses several protocols that are essential to deliver its real-time communications functionality. As can be seen in Figure 3, WebRTC uses User Datagram Protocol (UDP) at the transport layer, since it offers low latency and has little protocol overhead. It does, however, not guarantee the order of the packets or that a packet has been delivered at all. In an audio and video streaming environment, this is a compromise that application designers are willing to accept, since the human brain is able to fill small gaps easily, while it is highly sensitive to transmission delays (cf. Grigorik 2013, p. 315f).

With UDP alone, however, it is not possible to establish and maintain peer-to-peer connections. WebRTC needs Interactive Connectivity Establishment (ICE), Session Traversal Utilities for Network Address Translation (STUN) and Traversal Using Relays around Network Address Translation (TURN) as mechanisms to determine public Internet Protocol (IP) addresses and traverse Network Address Translation (NAT) layers and firewalls. On top of that layer, all data transferred between two peers is secured with Datagram Transport Layer Security (DTLS). After this stage, the Stream Control Transport Protocol (SCTP) and the Secure Real-Time Transport Protocol (SRTP) handle higher-level networking tasks like multiplexing of streams and provide congestion and flow control (cf. Leaver, Iwase & Katsura 2015). Together, all layers described in this section, provide the functionality of the PeerConnection and DataChannel API.



Figure 3: WebRTC protocol stack (Leaver, Iwase & Katsura 2015)

## Functionality

The majority of web applications is based upon the client-server principle, which means that the client (i.e. the web browser) requests a web page from the server, who in turn fulfils the request by delivering the Hypertext Markup Language (HTML) source code of the page. In WebRTC, this model is extended by introducing a peer-to-peer communication layer (cf. Loreto & Romano 2014 p. 2). As depicted in Figure 4, both peers Alice and Bob request a web page from a server, which also acts as the signalling server. The signalling server is responsible for various tasks, such as determining the best possible option for a direct network path between the peers and finding suitable audio and video stream encodings and resolutions. After this initial signalling stage, Alice and Bob now share a peer-to-peer connection, where all media data is exchanged directly between them, without the server being involved. The media data consist of audio and video streams and, optionally, arbitrary data transferred over a DataChannel.



Figure 4: WebRTC call topology (Leaver, Iwase & Katsura 2015)

## Signalling

An example of the signalling process is described in Figure 5. Sticking to the call setup from above, Alice is trying to call Bob. To start a peer-to-peer connection, Alice’s web application first creates a new PeerConnection object and, upon success, adds her own media stream tracks to it. This could either be audio or video tracks, or both. Afterwards, a signalling offer message is sent to the remote peer, in this case Bob. This offer message includes meta data about Alice’s media streams, such as codecs and media types, information about the network that Alice is part of as well as key data used to create secure connections (cf. Loreto & Romano 2014, p. 9).



Figure 5: Signalling process to start a PeerConnection with another user (Loreto & Romano 2014, p. 10)

Bob’s application receives this offer and starts a similar process, where the same information about Bob’s endpoint is added to the PeerConnection object, which is then sent back to Alice as a signalling answer message. This process can be repeated several times, until both peers have found a suitable network path for their peer-to-peer connection. To determine this path, WebRTC uses ICE, which is responsible for locating the public IP address of both peers. Since this might be problematic if one or both users are part of a network that uses NAT, this is accomplished by using STUN and TURN servers (cf. Loreto & Romano 2014 p. 8, p. 37).

The technical implementation of the signalling process can be achieved by a variety of options. One popular approach is to use WebSockets. This offers the advantage that the client and the signalling server share a persistent full-duplex connection, on which both parties can send data at any time. This is beneficial since the whole signalling process is of an asynchronous nature and requires the sending and receiving of multiple session description offers and answers. A different approach is to use XMLHttpRequest (XHR), which is as viable as the WebSockets approach from a technical point of view. For a developer, however, the use of XHR requires a more complex application architecture since it is built upon stateless, unidirectional Hypertext Transfer Protocol (HTTP) requests (cf. Khan 2015).

## Advantages

One significant advantage of WebRTC is the fact that it is platform independent (cf. Grégoire 2015). With classic desktop applications, developers have to ensure that they are functioning across a number of operating systems, such as Microsoft Windows, Linux or Apple’s Mac OS. This is not necessary with web applications because they are running in web browsers. The fact that the main execution environment of WebRTC is a web browser brings the additional advantage that there are several device types that have browsers installed, which further broadens the number of platforms where web applications can be used on. This includes desktop computers, laptops, smartphones and lately also wearable devices like smart glasses and smart watches. It has to be noted, however, that not all web browsers offer the same range of features.

Another large benefit of WebRTC is that it can be used free of charge. This is especially interesting for companies who need to use real-time communication as part of their daily business, for instance for exchanging information with branch offices abroad. Several proprietary business solutions, such as Skype Business[[2]](#footnote-2), bill the use of their service per customer and per month. WebRTC could offer businesses an option to easily develop such a real-time communication application themselves and save monthly license charges for a proprietary product.

Additionally, WebRTC brings the advantage that all components of the communication process are securely encrypted. This is of special interest to a large number of users, who are concerned about data privacy on the internet. WebRTC uses DTLS to encrypt the data transfers between two users in a PeerConnection. On top of that, the SRTP is used together with the SCTP to handle the real-time communication functionality, such as reliable delivery, flow control and multiplexing of media streams (Leaver, Iwase & Katsura 2015).

To further improve the security of the whole WebRTC environment, Google Chrome in December 2015 removed the possibility to obtain access to a device’s camera and microphone via the MediaStream API if the web page was not loaded using Hypertext Transfer Protocol Secure (HTTPS) (cf. What’s next for WebRTC 2015). Consequently, developers who want to use this feature are encouraged to run their applications in a more secure environment. If a web page does not use HTTPS, user inputs, such as data submitted in a form, are transferred to the web server in plain text. With applications like Wireshark[[3]](#footnote-3), it is possible for anyone in the same network to read the submitted data. When using HTTPS, on the other hand, the entire data transfer is encrypted with Transport Layer Security (TLS).

The design of the WebRTC architecture relies on a peer-to-peer model. In reality, this means that once a PeerConnection between two users has been established, there are no third-party servers involved in the data transfer. This offers two advantages at the same time. First, it reduces network latency because the peers are connected directly, and second, it increases security by removing one component in the transfer that could be a potential target for attacks (cf. Azevedo, Lopes Pereira & Chainho 2015).

## Limitations

A significant limitation to WebRTC is its browser compatibility. As depicted in Figure 6, the web browsers that fully support WebRTC are Google Chrome, Mozilla Firefox, both together with its mobile counterparts, and Opera.



Figure 6: Overview of browsers that have a working implementation of WebRTC PeerConnections[[4]](#footnote-4)

As can be seen in Figure 7, these three browsers represent approximately 61% of the total web browser market share in Austria in 2015[[5]](#footnote-5). By comparison, this figure has risen by three percentage points from 58% in 2014 (cf. Stifter 2016, p. 26).



Figure 7: Web browser market share in Austria in 2015

As for the other browsers, Microsoft’s Internet Explorer has been discontinued and cannot be installed on the current operating system Windows 10. As a result, the market share of Internet Explorer is expected to decline in the near future. Its successor, Microsoft Edge, has started to implement Object Real-Time Communication (ORTC), which is compatible to WebRTC in its current state with some additional workarounds. In September 2015, the first features of ORTC were integrated into Edge[[6]](#footnote-6). Apple, on the other hand, has not yet revealed any plans on integrating WebRTC into their Safari browser.

One reason for the incomplete browser compatibility is that WebRTC is still under development. It has a working draft API definition[[7]](#footnote-7), which is maintained by the World Wide Web Consortium (W3C) and is updated on a non-regular basis. The fact that the definition is not yet finished could possibly discourage developers, since some API functions and methods might be changed in the future. Additionally, some web browsers still use vendor prefixes for certain methods, which makes development difficult. For instance, the method for obtaining camera access is called navigator.getUserMedia() in the W3C specification, however, if developers want to ensure that it works across all browsers, they have to use the following lines of code in their application.



Listing 2: Necessary variable assignment to deal with vendor prefixes in web browsers

There are a number of other methods that suffer from the same issue, especially when it comes to WebRTC-specific functions, for instance RTCPeerConnection or RTCSessionDescription, which both must be assigned in the same way as described in Listing 2 for navigator.getUserMedia.

## Potential applications

Due to its technical design, WebRTC excels when it comes to providing real-time peer-to-peer communication. In the near future, this will most likely open up a number of new potential use cases, especially for web applications. Three possible use cases are briefly outlined in the following.

### Real-time communication

An evident use case for WebRTC is the field of real-time communication. Before the introduction of WebRTC, developing a real-time communication application meant that programmers had to obtain vast knowledge about audio and video codecs, communication protocols, data transfer and encryption. WebRTC simplifies this process significantly by providing a plain JavaScript API (cf. Leaver, Iwase & Katsura 2015). As a result, web developers can easily build such an application simply by calling the right API methods at the right time, without needing special knowledge about telecommunications technology.

Furthermore, due to the fact that WebRTC runs natively within web browsers, it is not necessary for the users to install any kind of software or plugin to use it. This could be an additional encouragement for people to use it, as there is no entry barrier in the form of downloading and installing software from the internet. This opens up new opportunities for e-commerce businesses to communicate with their customers personally and face-to-face, directly on their web page. For instance, these communication opportunities could include customer support or personal consulting.

In general, the field of real-time communication ranges from plain text messaging, telephone-like audio connections to video streaming. In its simplest form, there are only two users involved, but especially in a business environment, it is likely that there are a larger number of people involved in a conference call (cf. Leaver, Iwase & Katsura 2015).

### Peer-to-peer file sharing

An interesting application use case for WebRTC is peer-to-peer file sharing. Nurminen et al. (2013) examined the feasibility of such an application. The idea behind it was that popular video-on-demand platforms, for example You Tube, need to invest heavily in content delivery networks (CDN) in order to provide their videos promptly to a constantly increasing number of users. They designed a system where all users are part of a peer-to-peer network. Theoretically, it is only necessary for one user of the network to download a video from the server. If another user requests the same file, the web application asks if an active peer already has downloaded the file. If that is the case, it gets the file directly from this user, saving a considerable amount of bandwidth for the video platform and its content delivery network.

Nurminen et al. (2013) concluded that it was not possible due to the fact that web browsers at the time did not support the DataChannel API. Nowadays, however, a large number of browsers already support this API, including Mozilla Firefox, Google Chrome and Opera, making it a feasible option for platforms dealing with enormous bandwidth traffic.

### Integrating real-time sensor data

Azevedo, Lopes Pereira & Chainho (2015) propose a solution for a standardised API for the integration of real-time sensor data into web applications. They mention an example from the field of medicine, where a patient is able to remotely send data from a medical device, for instance a blood pressure sensor, to a doctor. In order for this concept to work, it is necessary to implement a middleware that handles the communication between the sensors and the web browser. This could be realised with a web browser extension (cf. Azevedo, Lopes Pereira & Chainho 2015).

Another field that could benefit greatly from such a solution is manufacturing, namely large factories. Especially since the rise of the “Industry 4.0” era, there have been several attempts to create “smart factories”, which offer increased flexibility in the production process. In such environments, malfunctions can be discovered significantly faster. They could even be detected before they happened, when the devices are able to issue warnings if measurement readings are not within pre-defined thresholds (cf. Dhungana et al. 2015).

## Usage possibilities on mobile devices

The main goal of WebRTC is to bring real-time communications to the web browser (cf. Loreto & Romano 2014, p. 1). This raises the question if there are other ways to use WebRTC on mobile devices.

### Web applications

One option that requires little effort is a browser-based web application that uses Responsive Web Design. Responsive Web Design aims at the “creation of web sites that take into account different types of devices, usually ranging from mobile phones to desktop, and optimise viewing experience for the device at hand” (Voutilainen, Salonen & Mikkonen 2015). It makes use of a special Cascading Style Sheets (CSS) version 3 feature, media queries. Media queries enable developers to define CSS rules based on custom criteria which have to apply in order to be used by the web browser. These criteria include the screen size, the orientation of a device or the pixel density of the screen (cf. Johansen, Pagani Britto & Cusin 2013).

Broadly speaking, Responsive Web Design can be achieved by two different approaches: First, it can be implemented from the ground up using media queries. This allows the developer a maximum of flexibility, since it is possible to define the query breakpoints and the responsive behaviour of components as detailed as desired. Second, it is possible to use a responsive web design framework. The advantage of such a framework is that it can be used without additional effort for setup. On the other hand, it can be laborious compared to the first approach to add customised behaviour.

A popular example of such a framework is Bootstrap[[8]](#footnote-8). Bootstrap focuses heavily on a “mobile-first” approach, and has a considerable amount of functionality built in. It makes use of a grid system, which by default contains twelve columns per row (cf. Voutilainen, Salonen & Mikkonen 2015). The behaviour of the columns can be adapted according to the size of the display the web page is viewed on. Bootstrap classifies device types into four categories, namely extra small devices, small devices, medium devices and large devices.

### Native app

WebRTC offers the possibility to be used in native mobile apps. As described in Chapter 2.2, the internal WebRTC API is written in C++, and offers an API for web applications. For native apps, however, there are no native WebRTC APIs yet. Google offers a WebRTC library for both the Android and iOS operating system. In order to use this library, the source code first must be compiled on the development system. The official WebRTC page offers step-by-step guidelines[[9]](#footnote-9) for the compilation process. It must be noted that for compiling the native Android code, it is necessary to use a machine which is running the Linux operating system. For the iOS code, a machine with Mac OS X is required, respectively.

### Native app with web views

Another option would be to use a standard native app that contains only one web view, which loads and displays a web application like a browser does. As a result, it is possible to provide users with a native app, without having to implement the application logic for the appropriate operating system.

Android features an implementation of WebRTC since version 4.4 of its operating system. However, as Hart (2015) points out, it is based on Chromium 36 and it is not guaranteed that it is up to date with the latest WebRTC version in web browsers. On iOS, on the other hand, there is no possibility to use WebRTC inside a UIWebView in a native app. This is due to the fact that it uses the Safari browser as an underlying foundation, which does not feature any implementation of WebRTC at all, as mentioned in Chapter 2.8.

### Cross-platform developed mobile app

When it comes to using WebRTC inside a cross-platform developed mobile app, developers currently have two options.

First, there is Crosswalk[[10]](#footnote-10), which is an open source web application runtime. The Crosswalk Project is backed by technology company Intel and is built upon Apache Cordova. The substantial advantage of using Crosswalk is the fact it always uses the latest version of the Google Chrome browser for its web view. This is a considerable benefit when developing a WebRTC application, since Google Chrome is generally the first web browser to implement new WebRTC features (cf. Hart 2015). Due to the fact that Crosswalk uses Apache Cordova internally, applications for it are written using HTML5, JavaScript and CSS.

The second option is OpenWebRTC, which is a project by Swedish company Ericsson Research and describes itself as a “mobile-first WebRTC client framework for building native apps” [[11]](#footnote-11). It is built on top of the GStreamer multimedia framework. Like Crosswalk, the source code is open source and publicly available on the internet. OpenWebRTC cannot be considered a traditional cross-platform mobile development framework. It rather offers natively compiled libraries that provide the WebRTC functionality with API methods similar to the ones used in web applications in the browser. Using OpenWebRTC, developers can decide to either use it in a native or a hybrid app. When used in native apps, it offers the advantage of a compiled library, without the need to build the Google WebRTC library. In hybrid apps, the OpenWebRTC helper library provides custom web views for Android and iOS, which are derived from their parent classes, WebView in Android and WKWebView in iOS. As a result, it is possible to use WebRTC inside a web view on an iOS device.

## Conclusion

It has been established that the WebRTC technology can be used in a telecommunications environment and due to the fact that it is independent of specific operating systems is a considerable benefit. Furthermore, 100% of the data transfer is securely encrypted, which is particularly compelling to companies concerned about the privacy of their data.

However, the limitations mentioned in this chapter still remain of serious nature, especially in a consumer environment. It is hardly feasible to coerce an end customer to switch to a certain web browser in order to use a web application. The same applies to a business context. While native mobile apps can provide a reliable solution for this problem, they also come with a significantly intensified effort, both in the development and maintenance stage.

On the whole, one potential compromise that can be considered both economic and user-friendly, is the use of cross-platform apps. While they suffer from certain detriments with regard to user experience, the development effort is minimised and it is guaranteed that users will be able to use the app, on condition that they have a smartphone or tablet that is not older than five years.

This chapter took a closer look on the architecture and technology behind WebRTC and compared its strengths and weaknesses. Furthermore, potential use cases were discussed and the possibilities to use it in applications on mobile devices examined with the conclusion that cross-platform developed apps might offer a practicable approach to achieve widespread dissemination on mobile devices by using a single codebase. The following chapter discusses cross-platform mobile development in detail along with possible options for cross-platform development frameworks.

# Cross-platform mobile development

Cross-platform mobile development aims at providing developers with opportunities to write code for mobile applications once and run it on all platforms. In theory, this method results in significantly reduced development resources compared to native app development. However, it also includes some possible drawbacks, such as diminished user experience (cf. Angulo & Ferre 2014).

This chapter provides an overview of cross-platform mobile development. It discusses the advantages and limitations of this technology along with its different approaches and presents several possible framework options to simplify the mobile development process.

## Background

Beginning with the introduction of Apple’s iPhone back in 2007, mobile applications have become massively popular. Back then, it was self-evident to implement an app natively because there were no other feasible options. However, the following years saw a substantially increasing number of popular mobile platforms, such as Android, Windows Phone and the aforementioned iOS. Since all these platforms use different programming languages, there was no possibility to reuse the programming code written for one platform. As a result, it had to be rewritten in the exact same way for all platforms that should be supported. Furthermore, making changes to an app again meant going through all platforms and implementing the changes separately for each platform, which results in increased development effort and expenses (cf. Xanthopoulos & Xinogalos 2013).

This problem has led to a change of thinking in application development. A common approach nowadays is for an application to use an API for retrieving and manipulating the underlying data. This offers the benefit that there is only one standardised way of interaction that various application types, such as a web application, a desktop application and mobile apps can use (cf. Azevedo, Lopes Pereira & Chainho 2015). However, each application type still needs to develop the business logic individually.

Another solution to this problem is cross-platform mobile development. It enables developers to write code for an app only once and subsequently use it from that code base in applications for all desired platforms. In most cases, the code is written using web technologies, such as HTML5, JavaScript and CSS. Incidentally, as Charland & LeRoux (2011) point out, it was the original plan for apps for the iPhone to be written using these tools. In the end, however, Apple decided that third-party apps for their operating system have to be written natively in Objective-C, which was followed by Swift in 2014.

A common approach used in this context is Model-Driven Development (MDD). In MDD, the business logic is separated from the technical implementation and a model is used to describe the problem domain (cf. Charkaoui, Adraoui & Benlahmar 2014). To achieve that, the functionality of the application and how it behaves under different circumstance is described using a modelling language (cf. Xanthopoulos & Xinogalos 2013). A variation of this approach is used for generating cross-platform mobile apps, which is discussed in more detail in Chapter 3.5.4.

## User experience

One area that was heavily influenced by the spread of smartphones over the last years is user experience, a term which refers to a person’s overall experience when using a software application or system (cf. Mullins 2015). Charland & LeRoux (2011) define two primary categories for user experience: The context and the implementation of the application or system. Both categories are described in more detail in the following section.

### Context

The context of the application consists of components which need to be understood by the user, such as platform-specific user interface conventions. These conventions can differ across mobile platforms. Charland & LeRoux (2011) describe a common functionality that is implemented differently on Android and iOS: the ability to go back to a previous view inside an app. On Apple devices for instance, there are virtual buttons that provide this functionality, typically in the top left corner of the screen. The majority of Android devices, on the other hand, are equipped with a physical back button in the lower right area of the device. In cross-platform apps, developers need to consider these conventions, as users of a mobile platform might expect an app to behave in a way they are used to from native apps (cf. Charland & LeRoux 2011). If it is crucial that the user interface components align with the respective platform’s conventions, developers can use interpreted or generated cross-platform apps, which are described in more detail in Chapters 3.5.3 and 3.5.4.

### Implementation

The implementation of the application involves all elements “that can be controlled in an application” (Charland & LeRoux 2011). This part is solely the responsibility of the developer and includes the handling of input errors, the graphical layout of an application and additional helpful features that improve the app’s workflow. With smartphones it is possible, for instance, that an application behaves differently based on the location it is used in. An app that offers information about restaurants could use the device’s GPS position to automatically show high-rated restaurants nearby. Another simple example for a valuable user experience is a chat app that lets users select contacts from the smartphone’s address book to start chats, without having to manually add them to the app. These examples enhance the user’s experience while interacting with the app by fulfilling basic tasks automatically.

In some scenarios, traditional web apps struggle to provide a similar user experience in comparison to native apps: For instance, of the two examples described previously it is only possible to determine the user’s GPS position in the browser, after getting explicit consent from the user for this action. There is, however, no possibility to access the contact list of the device inside a traditional web app, even if it is running on a smartphone. The same is true for accessing the phone’s built-in calendar app or issuing push notifications about events of interest to the user, like a new chat message.

Cross-platform developed apps offer developers the possibility to introduce a similar level of user experience to web apps. In most cases, they can access native device features through a special API of the cross-platform development framework. These features typically include the previously mentioned contact list, the calendar and geolocation as well as usage of native dialog elements, the device’s battery status, vibration control, network information and the motion of the device. During app execution, the framework then calls the respective native methods depending on which operating system the app is run on.

## Advantages

A significant advantage of cross-platform mobile development is reduced development time and costs. This is due to its approach of “write once, run anywhere” (cf. Xanthopoulos & Xinogalos 2013). Angulo & Ferre (2014) identify several further implications of this circumstance, for instance the reusability of developer skills and one common shared codebase to work on. With cross-platform mobile development, it is also considerably simpler to synchronise app releases. Furthermore, the advantage of diminished development effort affects all stages in the product life-cycle, for all subsequent app updates or maintenance works.

Additionally, cross-platform mobile development provides developers with a simple possibility to achieve a uniform app “look and feel” easily across all supported operating systems. Although users generally “seek apps that resemble native apps” (Heitkötter, Hanschke & Majchrzak 2012) in terms of user interfaces, it can be an advantage to have an app that looks and behaves the same way on all platforms, especially if the users are already thoroughly familiar with the functionality, for instance from an existing web app. Facebook, for example, deliberately decided to ignore certain platform-specific interaction conventions when they developed the mobile apps for their web application in order to provide users with interaction paradigms they were familiar with (cf. Angulo & Ferre 2014).

Another benefit of cross-platform mobile development stems from the fact that there is less platform-specific special knowledge necessary when developing an app with HTML5 technologies (cf. Xanthopoulos & Xinogalos 2013). In general, HTML5 can be considered as easier to learn than native app development, since in most cases it is not necessary to study specific platform development practices, which for instance include the build process for compiling the app, proper memory management, user interface conventions and a multitude of devices that might behave differently under certain scenarios, especially in the Android environment.

Furthermore, as Charkaoui, Adraoui & Benlahmar (2014) point out, cross-platform developed apps can be downloaded from its respective platform’s app market place, such as the Android Play Store or the iOS AppStore. This is a considerable advantage since it provides a single point of access for the users of the platform where they can obtain the applications. Cailean (2013) refers to this characteristic as the “discoverability” of an app. Traditional web apps that run solely in a browser do not share this trait.

## Limitations

Cross-platform developed apps commonly do not exhibit the same performance measures as native apps do. A reason for this is the fact that native apps are usually compiled, which in most cases results in faster execution times because the programming code is translated into machine code before the execution of the program. Cross-platform developed apps, on the other hand, mostly use interpreted languages such as JavaScript, which execute its code instructions step-by-step without compiling them first (cf. Charland & LeRoux 2011). This circumstance particularly affects hybrid apps, since their user interfaces are not utilising the optimised native components (cf. Xanthopoulos & Xinogalos 2013). Other types of cross-platform mobile developed apps, such as generated apps and to some extent interpreted apps, are not concerned by this circumstance.

Apart from that, cross-platform developed apps are not capable of directly accessing device-specific features, such as contact list, file system and various sensors. The communication to this underlying hardware is achieved by using an API layer (cf. Singh & Buford 2016). For developers, this process can be inconvenient and might not provide results with the same accuracy compared to native app development. Because of the additional API layer between the application and the device hardware, the execution of the app’s business logic might be further slowed down.

In addition, the fact that there is no standardised interface for the API that bridges the communication between the native operating system and the cross-platform developed app has led to the situation that each framework has implemented their own version of such an API (cf. Charland & LeRoux 2011). This can be inconvenient for developers because although there are few differences in terms of the functionality, the programming syntax can vary widely. As a result, switching from one cross-platform development framework to another can be an arduous task.

Another detail to consider when the aim of cross-platform mobile development involves the deployment in Apple’s App Store is that it might not be enough to simply wrap an existing web page inside the web view of a native app. According to Xanthopoulos & Xinogalos (2013), it is possible that such an app will be rejected by Apple’s reviewing team. In its official review guidelines, Apple clearly states that apps will be rejected if they are “simply web sites bundles as apps”[[12]](#footnote-12). Consequently, it is necessary that an app provides some sort of additional value in comparison to a simple web page. This could be achieved, for instance, by enhancing the app’s functionality by integrating the device’s contact list or calendar to provide supplementary functions and enhance the user experience.

## Approaches

Xanthopoulos & Xinogalos (2013) define four different categories for cross-platform developed apps: Web, hybrid, interpreted and generated apps. All four approaches are discussed in detail in this section.

### Web apps

Web apps are applications that run within a web browser. Typically, they use HTML5 and JavaScript. The advantage of web apps is that nowadays, almost any smart mobile device has a web browser installed, thus providing a broad range of dissemination. One disadvantage of web apps is the limited access to the device’s sensors, file system and features like contact list and calendar. Native apps, on the other hand, can exploit the device’s full potential when it comes to these features.

Unlike native apps, web apps do not need to be physically installed on the device and, furthermore, also do not have to be upgraded when a new version is available. At the same time, this becomes a disadvantage when users are not connected to the internet. In this case, the web app is not accessible to the user (cf. Xanthopoulos & Xinogalos 2013). There are modern HTML5 technologies like the Application Cache (AppCache) to eradicate this problem. AppCache allows developers to store programming logic and data on the user’s device. However, this technology requires substantial additional programming effort (cf. Xinogalos, Psannis & Sifaleras 2012).

### Hybrid apps

Hybrid apps are a combination of native apps and web apps. They are “primarily built using HTML5 and JavaScript, and a detailed knowledge of the target platform is not required” (Xanthopoulos & Xinogalos 2013). The essential difference to web apps is that they are running within a native app container. The code is still executed by a web browser, but can be bundled together with the application, thus removing the necessity of an active internet connection to download the programming logic. With hybrid apps, it is also possible to access the device’s special features through APIs provided by the cross-platform development framework (cf. Xanthopoulos & Xinogalos 2013).

### Interpreted apps

Interpreted apps use pre-defined commands to build the user interface with native components when the app is started. This means that on the Android platform users will interact with typical Android-styled buttons, while on iOS users will interact with iOS-styled buttons, without any additional effort of the developer. Despite this advantage in user experience, the developer is completely dependent on the used framework. This could especially pose a problem when a new version of an operating system is released, because it is not clear if the app will automatically have access to new features or if all previously used components will look and behave the same way (cf. Xanthopoulos & Xinogalos 2013).

### Generated apps

This type of cross-platform developed apps use the code to generate native apps from it. The main difference to interpreted apps is the fact that they are generated before the app is started. They benefit from a high overall performance due to the use of compiled native code. One downside of generated apps is the increased build time that has to be carried out each time a change is made to the app (cf. Xanthopoulos & Xinogalos 2013).

## Criteria for choosing a framework

In order to determine a suitable cross-platform development framework, it is necessary to define a set of criteria that a framework needs to fulfil to be considered a useful option for developing a mobile app that includes WebRTC.

### Support for mobile platforms

A significant factor for selecting a cross-platform mobile development framework is the support of mobile platforms (cf. Angulo & Ferre 2014). Ideally, such a framework should support a variety of platforms, and most importantly include the ones that offer the highest dissemination. The three most popular mobile platforms, namely Android, iOS and Windows Phone, represent 99.6% of the global smartphone operating system market share, according to recent statistics (see Figure 8). Consequently, developers can assume that their application reaches a vast majority of available smartphones if a framework supports these three platforms.

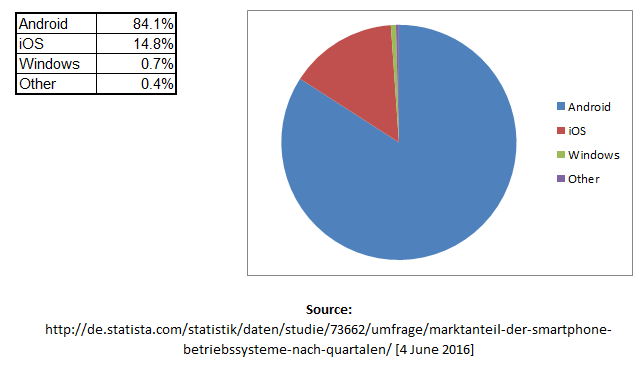


Figure 8: Global market share of smartphone operating systems in the first quarter of 2016

### Access to device-specific features

Angulo & Ferre (2014) argue that a fundamental feature of a cross-platform mobile development framework is its ability to offer access to underlying device-specific features, such as the current Global Positioning System (GPS) position, the address book, the calendar and various other sensors like the accelerometer and the device orientation. These features are commonly provided by an API. It is essential that a framework offers a similar level of functionality compared to native app development, although the program execution is likely to be slower in comparison since the access to the features happens over the API of the framework.

### Legal background

Another important criterion for choosing a cross-platform mobile development framework is the legal background. This includes the question whether the framework is distributed as free software or if there are usage fees, which can be further categorised into one-time and recurring payments. Furthermore, a crucial issue is the license under which the framework is published. This determines the fact whether a developer is allowed to create commercial software products with the framework without additional fees, which could be essential especially for companies that want to sell software products created with the framework (cf. Heitkötter, Hanschke & Majchrzak 2012).

### WebRTC capability

Due to the fact that this thesis aims at developing a WebRTC-capable cross-platform developed app, an essential criterion in this case is that a framework facilitates the use of WebRTC. Ideally, it should possible to use the same JavaScript programming logic for creating peer connections as in browser-based web applications, without having to add method prefixes for a plugin. This would minimise the development effort for creating a cross-platform developed app when a fully functional web application already exists.

### Side issues

Angulo & Ferre (2014) further define some less important side issues, for instance if the framework in question can be integrated with a widespread Integrated Development Environment (IDE) or even includes an own solution for such an environment. Furthermore, the programming language that the framework uses for developing cross-platform apps could be of interest for companies, who might favor a framework that uses a well-known programming language in order for new developers to be become easily acquainted with the project. Lastly, it could be beneficial if the framework offers an integrated solution for automatically building and distributing apps (cf. Heitkötter, Hanschke & Majchrzak 2012).

## Cross-platform development frameworks

Over the last years, a multitude of cross-platform development frameworks has emerged. This section gives a brief overview of some of the most popular frameworks and mention their particular characteristics.

### Apache Cordova (PhoneGap)

One of the most popular cross-platform development frameworks is Apache Cordova[[13]](#footnote-13). Apps built with this framework belong to the category of hybrid apps. Developers can write applications with HTML5, JavaScript and CSS files, which are then be executed inside a native application in a web view. Due to the fact that these tools are also used to develop web applications, this framework offers a relatively comfortable entry point into cross-platform mobile development. Access to underlying features such as sensors and file system is provided via an API, the *Cordova Plugins*. Apache Cordova provides support for numerous platforms, such as Android, iOS, Windows Phone, Blackberry and Ubuntu.

Apache Cordova is open-source, although its owner, the software company Adobe, also released a similar version of it called PhoneGap*[[14]](#footnote-14)*. PhoneGap is built on the same core application as Apache Cordova, but is part of a product package that also offers various additional tools, for instance a desktop application, a build tool and a variety of third-party libraries and plugins.

### Xamarin

Xamarin[[15]](#footnote-15) is another popular framework that builds generated native apps. Instead of web technologies it uses the programming languages C# or Ruby. Xamarin apps use native user interface components, thus providing app users with well-known interaction tools. It supports the most popular operating systems, namely Android, iOS and Windows Phone and also offers a native API to access device sensors. Xamarin also offers additional services such as an automated build tool.

### Appcelerator Titanium

Appcelerator Titanium[[16]](#footnote-16) is an example for a framework that creates interpreted apps, which means that apps created with this framework will use native user interface components. However, it also features some aspects from hybrid apps by providing developers with the possibility to write reusable modules in JavaScript.

Titanium is one product of the Appcelerator platform, together with tools like *Arrow*, which is a framework for easily building APIs or *Push*, which is a pre-built service for push notifications that can be integrated into apps. Furthermore, Appcelerator provides a multitude of analytics tools. It has to be noted that while Titanium itself is open-source and free-to-use, all other previously described tools from Appcelerator are only available in paid plans.

### Ionic

Ionic[[17]](#footnote-17) is a relatively new cross-platform mobile development framework that relies heavily modern web technologies like AngularJS[[18]](#footnote-18), Sass[[19]](#footnote-19) and virtual Document Object Model (DOM) rendering for data-intensive apps with rapidly changing user interfaces. It is built upon Apache Cordova. Ionic is an open-source project and its entire source code can be found on Github, where users are also able to report bugs or suggest improvements to the code.

### Sencha Touch

Sencha Touch[[20]](#footnote-20) focuses primarily on creating user interfaces. It has special features to simulate user interface components from Android and iOS within a web application. It does not, however, provide tools to build native apps. In order to use the code in a native app, it is either necessary to create a blank native app containing a web view for each platform the app should run on or use another framework like the previously mentioned Apache Cordova to fulfil the task.

### Other frameworks

There are also a number of smaller, lesser known cross-platform mobile development frameworks. These include jQuery Mobile[[21]](#footnote-21), Mobile Angular UI[[22]](#footnote-22), Kendo UI[[23]](#footnote-23) or the lightweight app.js[[24]](#footnote-24)

In conclusion, this chapter highlighted important aspects of cross-platform mobile development. User experience was identified as a considerable factor in mobile application development and some examples were provided on how it can be applied to cross-platform developed apps to provide convenient additional features in comparison to traditional web apps. Afterwards, the advantages and limitations of cross-platform mobile development were discussed along with framework options and the approaches they use and a set of criteria which they need to fulfil in order to enable programmers to develop a WebRTC-capable cross-platform app. The following chapter describes the process of integrating WebRTC into a cross-platform developed app.

# App development

Considering the findings of the underlying research, two cross-platform developed apps were created in order to evaluate their viability to be used in WebRTC-capable applications. The results of this evaluation are presented in Chapter 5. Additionally, the same application logic was integrated into a native Android app containing a web view, in order to enable the possibility to compare the results to those of the two cross-platform apps.

## Introduction

A WebRTC application commonly consists of two parts: First, a management server that is responsible users to discover each other and assist in the creation of peer connections between them. Second, a user interface that displays local and remote video streams and enables users to interact with each other, for instance start a call or send text messages. This user interface can either be a web application running in a browser, or a mobile app, which is in this case a cross-platform developed app.

## Previous work

Stifter (2016) implemented a browser-based real-time remote support application that uses WebRTC for peer-to-peer communication. This application consists of a management server written in Node.js and a web application using HTML5 web pages for the user interface. A significant advantage of this technical setup is the fact that there is only one programming language, JavaScript, involved, which simplifies development and facilitates reuse of code components between frontend and backend (cf. Stifter 2016, p. 35).

With this application, users can start peer-to-peer audio and video conferences with each other. The video feed can be paused if one user wants to take a closer look at the video image, which might be difficult with a moving video feed. Additionally, they are able to exchange plain text messages, which can be useful in case one person is located in a loud environment where it is difficult to understand the other person. A special feature of the remote support application is that users are able to assist each other by drawing coloured indicators on top of the video feed. This is especially helpful when one person has more experience with the matter at hand than the other person.

Since the aim of this thesis is to evaluate possibilities of using WebRTC in cross-platform mobile apps, the previously described existing management server from Stifter will be used for the signalling part. The user interface will be realised with cross-platform apps. In the end, it should be possible to start video calls from the existing web application to the cross-platform apps and vice versa. Furthermore, the same should be possible for the apps among themselves.

## Necessary adjustments

In order to successfully communicate with the management server, it is first necessary to open a secure WebSocket connection. This is achieved by calling the URL of the server with the “wss://” protocol prefix instead of the “https://” used when requesting a web page. When the WebSocket connection has been successfully established, the management server responds with a welcome message that contains the new client’s unique ID. This ID will further be used to start peer connections between users.

Each time a new user connects to the management server, a message is broadcast to all connected clients in order to inform them about the available peers. This process is repeated when a user disconnects from the server, for instance by closing the application. The application is then responsible for providing a user interface that enables users to start calls with connected peers. When one user decides to call another, this is achieved by sending a call message with containing the ID of the collocutor to the management server. The server then relays this message to the peer with the matching ID (cf. Stifter 2016, p. 36). After that, the two peers start the signalling process that was described in Chapter 2.6 by exchanging their respective session description offers and answers, which results in the creation of a shared peer connection. Additionally, a data channel is opened to enable the two users to send text messages to each other. This data channel is also used for exchanging the drawing path information for the overlay indicator feature.

Due to the fact that the web application from Stifter (2016) was written in JavaScript, large portions of the existing code could be reused inside the cross-platform app with only small adaptations necessary to facilitate a working signalling communication between management server and app. As a result, the main task of the app development was to ensure a functioning integration of the cross-platform development framework and the plugins that enable the WebRTC functionality.

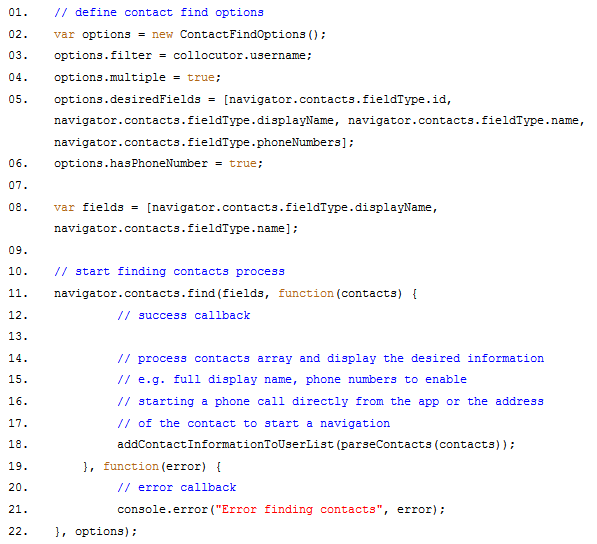
## Additional features to enhance user experience

As mentioned in Chapter 3.2, a substantial advantage of mobile apps is the circumstance that they can deliver significantly enhanced user experience compared to web apps. Therefore, it was decided to implement two additional features in order to create an appealing user experience for the cross-platform developed apps.

### Contact list

First, the Apache Cordova contact plugin (“cordova-plugin-contacts”) was used to enhance the information displayed about available users. The feature retrieves all entries from the device’s contact list and tries to match the first and last names with the usernames provided by the management server. If a match is found, additional information about this contact is displayed in the available user section. This includes this user’s phone numbers and address. Consequently, it is possible that users of the cross-platform app start a phone call directly from the app, for instance when the internet connection has been lost and no WebRTC video streams are possible. Furthermore, app users could start navigation to the address of the other user if there are issues to be discussed in person. The technical implementation of the contact retrieval process is outlined in Listing 3.

It is important to note that this feature does not pose a threat with regard to data privacy, since only information can be displayed that has been entered manually into the contact list by the user. It is not possible to access or retrieve information from the other person’s device.



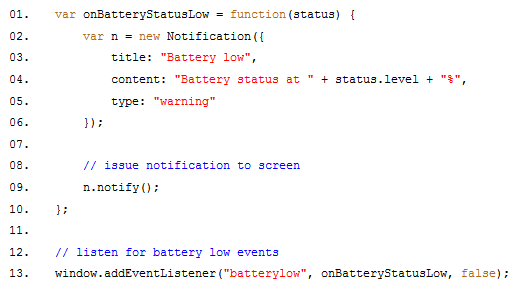
Listing : Retrieve information for another user from the contact list of the device

### Battery status

Another useful feature in the context of video streaming are notifications about the device’s battery status. In the implementation of the apps, users are notified if they try to start a video stream when the battery status drops beneath a threshold that varies across different device types, but typically ranges within fifteen to twenty percent of the total battery charge. As a result, users are informed about the low battery status beforehand, and are less likely to use the app until the battery is completely drained.

For instance, information about the device’s battery status can be easily obtained using Apache Cordova’s plugin “cordova-plugin-battery-status”. It adds three events to the app’s *window* object. First, *batterystatus*, which returns an object containing the remaining percentage of the battery’s charge available as well as information whether the device is plugged in or not. This event is called every time the percentage value changes. Second, *batterylow*, which is called when the battery charge drops beneath a device-specific threshold value. Third, *batterycritical*, which works in the same way as *batterylow*, but means that the device should be charged immediately.

Listing 3 provides a simple example of how a user can be informed about the device’s low battery status using a built-in notification inside the app. Additionally, it would be possible to further enhance the programming logic using this information, for instance by automatically requesting a reduced video resolution and frame rate in order to prolong the battery’s performance if the battery status drops below an adjustable threshold value.



Listing : Example of informing the user about a low battery status inside the application

## Implementation of web app in web view

# DESCRIPTION/INSIGHTS OF PROCESS

## Implementation of cross-platform mobile apps

# DESCRIPTION/INSIGHTS OF PROCESS

# PROBLEMS DURING SETUP

# CODE LISTINGS

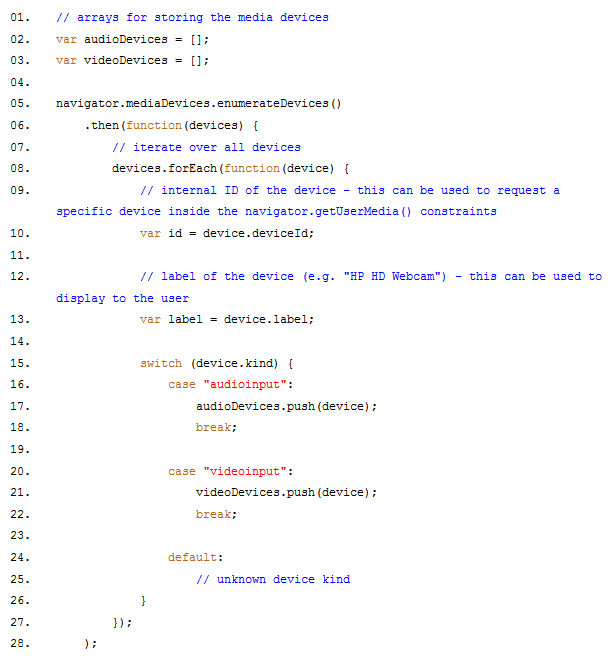
### Crosswalk

### OpenWebRTC

## Insights

A valuable insight during the cross-platform development process was that unlike desktop computers and laptops, smartphones are commonly equipped with multiple media devices, such as cameras and microphones. Thus, it is helpful to provide users with the possibility to select the desired media input. Especially for video conferencing, users might want to switch the camera mode between front-facing for face-to-face communication and back-facing for sharing the camera feed with another person.

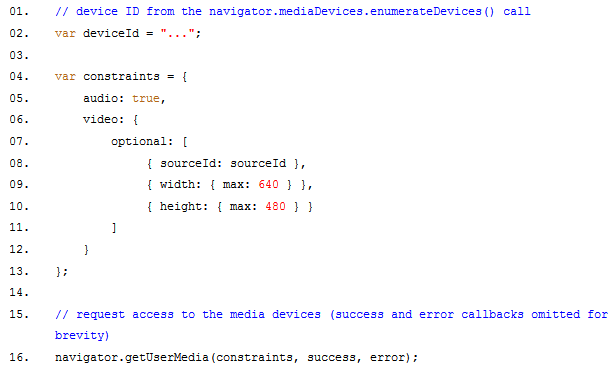
Listing 3 describes a simple example of how a list of all attached media devices can be obtained. In order to enable the user to select a specific camera, the content of the *videoDevices* array could be displayed inside a select box. Upon selection of an option, the device ID from line 10 could be passed inside the constraints object of the navigator.getUserMedia() call, resulting in the selection of the desired camera. Listing 4 below in line 8 provides insight on how this device ID can be used to request a specific camera.



Listing : Determine all available media devices connected to the physical device

Furthermore, developers can provide additional user experience in their apps by considering changing network conditions and react to them in a sensible way. During the WebRTC connection setup, for instance, it is possible to request lower video quality and framerates when a device is located inside a network with low bandwidth. Additionally, the app could automatically try to re-establish a PeerConnection if it was interrupted due to network loss (cf. Singh & Buford 2016).

An example of how lower video quality can be requested using navigator.getUserMedia() is outlined in Listing 4. In line 7, an additional optional parameter array is passed to the video access request, containing special attributes such as a specific camera and a video resolution of 640 x 480 pixels at most, thus limiting the data transfer for the video stream.



Listing : Example of requesting media devices with specific attributes

# Evaluation

COMPARISON OF THE THREE DEVELOPED APP

* SETUP FRAMEWORK
* DEVELOPMENT DURATION
* UX (UI COMPONENTS, …)
* ADDITIONAL TOOLS OF THE FRAMEWORK (DEPLOYMENT, TESTING, …)
* SIZE AND MEMORY CONSUMPTION OF APP
* ACCESS TO DEVICE FEATURES (CONTACTS, SENSORS, …)

## Setup

## Method

## Results

# Outlook

In May 2016, the WebRTC technology turned five years old. Although there have been little new technical additions to WebRTC lately, it still is a long way before it can be considered finished. The majority of the processes at the moment focus on raising the number of WebRTC-capable web browsers. Similarly, the underlying work on the prototype application is far from being finished. There are numerous possibilities for further extending the current project. Some forecasts on technical extensions are mentioned in this chapter along with possible enhancements for the prototype.

## The future of WebRTC

WebRTC evangelist Tsahi Levent-Levi assumes that Apple will introduce the first implementation of WebRTC into their Safari browser within the next year, around the end of 2016. This is based on the fact that the development team of WebKit, which is Safari’s rendering engine, have started adding WebRTC functionality. He further believes that there will be no advance notice from Apple with regard to this matter, but rather an official announcement once the functionality is fully implemented into Safari. The same holds true for Microsoft, who have announced at the BUILD conference in April 2016 that in addition to their implementation of ORTC, which was mentioned in Chapter 2.8, they are also working on adding WebRTC as it is currently specified (cf. Levent-Levi 2016).

## User management and authentication

The backend server that handles the WebRTC connection setup and distribution of information about available peers is currently not authenticating user requests. This does not mean that communication with the server is insecure, all requests to and from the server use HTTPS and are therefore encrypted with TLS. However, there is currently no user management system implemented, which would allow users to log into the application with conventional username and password combinations. For now, anyone who knows the Uniform Resource Locator (URL) of the application is able to use it.

To eradicate this problem, it is either possible to design and implement an authentication solution from the ground up or use an existing application like for instance Passport[[25]](#footnote-25), which is an authentication middleware for Node. With Passport, it is possible to use local username and password authentication as well as authenticating users with an authorization protocol like OAuth[[26]](#footnote-26).

## Multi-user sessions

At this time the application allows for any number of parallel peer-to-peer sessions, meaning that one session cannot contain more than two users. While this entails a number of advantages previously discussed in this thesis, it might sometimes be necessary to invite more than two users to a session. Especially in remote support environments it might be beneficial to get the opinion of another expert to solve certain problems.

Due to its peer-to-peer design, WebRTC only supports two users in one session. If three or more users want to participate in a session, one solution would be to use a Multipoint Control Unit (MCU) as a central communication point which handles the routing of audio and video streams between all participating parties. There are publicly available open-source solutions like Erizo[[27]](#footnote-27) or Janus[[28]](#footnote-28) which could perform this task without requiring considerable development efforts.

# Conclusion

List of tables

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

List of figures

[Figure 1: WebRTC architecture (Grigorik 2013, p. 311) 11](#_Toc453231455)

[Figure 2: A WebRTC MediaStream object that contains one video and two audio tracks (Loreto & Romano 2014, p. 13) 12](#_Toc453231456)

[Figure 3: WebRTC protocol stack (Leaver, Iwase & Katsura 2015) 15](#_Toc453231457)

[Figure 4: WebRTC call topology (Leaver, Iwase & Katsura 2015) 16](#_Toc453231458)

[Figure 5: Signalling process to start a PeerConnection with another user (Loreto & Romano 2014, p. 10) 17](#_Toc453231459)

[Figure 6: Overview of browsers that have a working implementation of WebRTC PeerConnections 19](#_Toc453231460)

[Figure 7: Web browser market share in Austria in 2015 20](#_Toc453231461)

[Figure 8: Global market share of smartphone operating systems in the first quarter of 2016 34](#_Toc453231462)

List of listings

[Listing 1: Simple example for requesting access to camera and microphone of user device 13](#_Toc453231463)

[Listing 2: Necessary variable assignment to deal with vendor prefixes in web browsers 21](#_Toc453231464)

[Listing 3: Determine all available media devices connected to the physical device 40](#_Toc453231465)

[Listing 4: Example of requesting media devices with specific attributes 41](#_Toc453231466)

List of abbreviations

AJAX Asynchronous JavaScript And XML

API Application Programming Interface

AppCache Application Cache

BSD Berkeley Software Distribution

CSS Cascading Style Sheets

DOM Document Object Model

DTLS Datagram Transport Layer Security

GPS Global Positioning System

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protcol

HTTPS Hypertext Transfer Protocol Secure

ICE Interactive Connectivity Establishment

IDE Integrated Development Environment

IETF Internet Engineering Task Force

IP Internet Protocol

MCU Multipoint Control Unit

MDD Model-Driven Development

NAT Network Address Translation

ORTC Object Real-Time Communication

RAM Random Access Memory

RPC Remote Procedure Calls

SRTP Secure Real-time Transport Protocol

STUN Session Traversal Utilities for Network Address Translation

TLS Transport Layer Security

TURN Traversal Using Relays around Network Address Translation

URL Uniform Resource Locator

W3C World Wide Web Consortium

WebRTC Web Real-Time Communication

WLAN Wireless Local Area Network

XML Extensible Markup Language

Bibliography

Angulo E & Ferre X 2014. ‘A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX’. In Proceedings of the XV International Conference on Human Computer Interaction (Interacción '14). ACM, New York, NY, USA, , Article 27 , 8 pages. doi: 10.1145/2662253.2662280

Azevedo J, Lopes Pereira R & Chainho P 2015, ‘An API proposal for integrating Sensor Data into Web Apps and WebRTC‘. In Proceedings of the 1st Workshop on All-Web Real-Time Systems (AWeS '15). ACM, New York, NY, USA, Article 8. ISBN: 978-1-4503-3477-8. doi: 10.1145/2749215.2749221

Cailean I 2013, *Why native apps remain unrivalled by web apps in user experience and discoverability*. Available from: <http://www.trademob.com/why-native-apps-remain-unrivalled-by-web-apps-in-user-experience-and-discoverability/> [6 June 2016]

Charkaoui S, Adraoui Z & Benlahmar EH 2014. ‘Cross-platform mobile development approaches’. In 2014 Third IEEE International Colloquium in Information Science and Technology (CIST), Tetouan, 2014, pp. 188-191.  
doi: 10.1109/CIST.2014.7016616

Charland A & LeRoux B 2011. ‘Mobile Application Development: Web vs. Native’. Queue 9, 4, Pages 20 (April 2011), 9 pages. doi: 10.1145/1966989.1968203

Dhungana D, Falkner A, Haselböck A & Schreiner H 2015. ‘Smart factory product lines: a configuration perspective on smart production ecosystems’. In *Proceedings of the 19th International Conference on Software Product Line* (SPLC '15). ACM, New York, NY, USA, 201-210. doi: 10.1145/2791060.2791066

Grégoire, JC 2015, ‘On Embedded Real Time Media Communications‘*.* In Proceedings of the 1st Workshop on All-Web Real-Time Systems (AWeS '15). ACM, New York, NY, USA, Article 7 , 4 pages. ISBN: 978-1-4503-3477-8. doi: 10.1145/2749215.2749224

Grigorik I, 2013, *High Performance Browser Networking*, 1st edn., O’Reilly Media, Sebastopol. ISBN: 978-1-449-34476-4.

Hart C, 2015, *Developing mobile WebRTC hybrid applications*. Available from: <https://webrtchacks.com/webrtc-hybrid-applications/>. [29 May 2016]

Heitkötter H, Hanschke S & Majchrzak TA 2012, ‘Comparing Cross-Platform Development Approaches for Mobile Applications’. Available from: <https://www.wi1.uni-muenster.de/pi/veroeff/heitkoetter/Comparing-Cross-Platform-Development-Approaches-for-Mobile-Applications.pdf> [6 June 2016]

Johansen RD, Pagani Britto TC, Cusin CA 2013, ‘CSS Browser Selector Plus: A JavaScript Library to Support Cross-browser Responsive Design‘. In *Proceedings of the 22nd International Conference on World Wide Web* (WWW '13 Companion). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 27-30. ISBN: 978-1-4503-2038-2.

Khan M, 2015. *WebRTC Signalling Concepts*. Available from: [https://www.webrtc-experiment.com/docs/WebRTC-Signalling-Concepts.html](https://www.webrtc-experiment.com/docs/WebRTC-Signaling-Concepts.html). [28 May 2016]

Leaver E, Iwase Y, Katsura K 2015, *A Study of WebRTC Security*. Available from: <http://webrtc-security.github.io/>. [6 June 2016]

Levent-Levi T, 2014. *What’s Behind Ericsson’s OpenWebRTC Project?* Available from: <https://bloggeek.me/ericssons-openwebrtc-project/>. [29 May 2016]

Levent-Levi T, 2016. *Microsoft, Apple and WebRTC in 2016.* Available from: <https://bloggeek.me/microsoft-apple-webrtc/> [4 June 2016]

Loreto, S & Romano SP, 2014, *Real-Time Communication with WebRTC*, 1st edn., O’Reilly Media, Sebastopol. ISBN: 978-1-449-37187-6.

Mullins C 2015, ‘Responsive, mobile app, mobile first: untangling the UX design web in practical experience’. In Proceedings of the 33rd Annual International Conference on the Design of Communication (SIGDOC '15). ACM, New York, NY, USA, , Article 22 , 6 pages. doi: 10.1145/2775441.2775478

Nurminen JK, Meyn AJR, Jalonen E, Raivio Y & Garcia Marrero R 2013, ‘P2P media streaming with HTML5 and WebRTC’. In Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on, Turin, 2013, pp. 63-64. doi: 10.1109/INFCOMW.2013.6970739

Ristic D, 2014. *WebRTC data channels*. Available from: <http://www.html5rocks.com/en/tutorials/webrtc/datachannels/>. [28 May 2016]

Singh K & Buford J 2016, ‘Developing WebRTC-based Team Apps with a Cross-Platform Mobile Framework’. 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, 2016, pp. 236-242. doi: 10.1109/CCNC.2016.7444762

Stifter M, 2016. *WebRTC – Development of a browser based real-time peer-to-peer remote support application*. Bachelor thesis, FH JOANNEUM Kapfenberg.

Voutilainen JP, Salonen J & Mikkonen T 2015. ‘On the Design of a Responsive User Interface for a Multi-device Web Service’. In Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on, Florence, 2015, pp. 60-63. doi: 10.1109/MobileSoft.2015.16

WebRTC Tutorial, 2014 (video file). Available from: <https://www.youtube.com/watch?v=5ci91dfKCyc>. [27 May 2016]

What’s next for WebRTC, 2015 (video file). Available from: <https://www.youtube.com/watch?v=HCE3S1E5UwY>. [6 June 2016]

Xanthopoulos S & Xinogalos S 2013. ‘A comparative analysis of cross-platform development approaches for mobile applications’. In Proceedings of the 6th Balkan Conference in Informatics (BCI '13). ACM, New York, NY, USA, 213-220. doi: 10.1145/2490257.2490292

Xinogalos S, Psannis KE & Sifaleras A 2012. ‘Recent advances delivered by HTML 5 in mobile cloud computing applications: a survey’. In Proceedings of the Fifth Balkan Conference in Informatics (BCI '12). ACM, New York, NY, USA, 199-204. doi: 10.1145/2371316.2371355

1. <http://webrtcstats.com/are-we-at-the-tipping-point-of-webrtc-adoption/> [4 June 2016] [↑](#footnote-ref-1)
2. <https://www.skype.com/de/business/> [↑](#footnote-ref-2)
3. <https://www.wireshark.org/> [↑](#footnote-ref-3)
4. <http://caniuse.com/#search=webrtc> [27 May 2016] [↑](#footnote-ref-4)
5. <http://www.statista.com/statistics/421152/wbe-browser-market-share-in-austria/> [27 May, 2016] [↑](#footnote-ref-5)
6. <https://blogs.windows.com/msedgedev/2015/09/18/ortc-api-is-now-available-in-microsoft-edge/> [27 May 2016] [↑](#footnote-ref-6)
7. <https://www.w3.org/TR/webrtc/> [↑](#footnote-ref-7)
8. <http://getbootstrap.com/> [↑](#footnote-ref-8)
9. <https://webrtc.org/native-code/> [↑](#footnote-ref-9)
10. <https://crosswalk-project.org/> [↑](#footnote-ref-10)
11. <http://www.openwebrtc.org/> [↑](#footnote-ref-11)
12. <https://developer.apple.com/app-store/review/guidelines/> [1 June 2016] [↑](#footnote-ref-12)
13. <https://cordova.apache.org/> [↑](#footnote-ref-13)
14. <http://phonegap.com/> [↑](#footnote-ref-14)
15. <https://www.xamarin.com/> [↑](#footnote-ref-15)
16. <http://www.appcelerator.com/titanium/titanium-sdk/> [↑](#footnote-ref-16)
17. <http://ionicframework.com/> [↑](#footnote-ref-17)
18. <https://angularjs.org/> [↑](#footnote-ref-18)
19. <http://sass-lang.com/> [↑](#footnote-ref-19)
20. <https://www.sencha.com/products/touch/#overview> [↑](#footnote-ref-20)
21. <https://jquerymobile.com/> [↑](#footnote-ref-21)
22. <http://mobileangularui.com/> [↑](#footnote-ref-22)
23. <http://www.telerik.com/kendo-ui> [↑](#footnote-ref-23)
24. <http://code.kik.com/app/2/index.html> [↑](#footnote-ref-24)
25. <http://passportjs.org/> [↑](#footnote-ref-25)
26. <http://oauth.net/> [↑](#footnote-ref-26)
27. <https://github.com/ging/licode/tree/master/erizo> [↑](#footnote-ref-27)
28. <https://janus.conf.meetecho.com/> [↑](#footnote-ref-28)