FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Mobile Witness App

Witestify

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Wen Chao Chen

Matrikelnummer 1129468

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Wien, 19. September 2015	Wen Chao Chen	Horst Eidenberger

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Mobile Witness App

Witestify

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Wen Chao Chen
Registration Number 1129468

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Vienna, 19th September, 2015

Wen Chao Chen	Horst Eidenberger

Erklärung zur Verfassung der Arbeit

Wen Chao Chen
Dr. Natterergasse 2-4/5/23

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. September 2015

Wen Chao Chen

Danksagung

Zunächst möchte ich mich bei meinem Betreuer, Ao.Univ.Prof. Mag. Dr. Horst Eidenberger, bedanken für die Gelegenheit, an diesem interessanten Projekt zu arbeiten.

Mein Dank gilt auch den Kollegen und Freunden, die sich die Zeit nahmen, als Testpersonen zu agieren, und hilfreiche Rückmeldungen sowie Ratschläge lieferten.

Vor allem jedoch gilt der Dank meiner Familie, die mir stets zur Seite stand und mich im Laufe des Studiums unterstützten.

Acknowledgements

I wish to express my gratitude to my supervisor, Ao.Univ.Prof. Mag. Dr. Horst Eidenberger, for his guidance and support, as well as the opportunity to work on such an interesting project.

I would also like to thank my friends and colleagues, who assisted me in the testing stages, who provided me with informative feedback, and made suggestions to improve my work.

Finally, I am most grateful to my family for their encouragement, and their endless support in the course of my studies.

Kurzfassung

In dieser Arbeit wird eine Android App vorgestellt, welche die Funktion eines digitalen Zeugen einnehmen soll. Eine audiovisuelle Aufnahme, von welcher Ort und Zeit vermerkt werden, wird auf einen Online Datenspeicher übertragen und wird auf diese Art vor fremden Zugriff geschützt. Die Sicherstellung, dass die Daten nicht vom User manipuliert werden kann, sowie die Aufzeichnung von Zeit und Ort würde einen idealen digitalen Zeugen erschaffen.

Abstract

In this thesis, an Android app that provides the functionality of an electronic witness, is introduced. An audiovisual recording, from which the location and the time are recorded, is transferred to an online data space to prevent accessing. The guarantee that the data cannot be manipulated by the user, and recording of time and location would create the ideal electronical witness.

Contents

CHAPTER 1

Introduction

In recent years, image and video capturing have seen a shift from the handheld camera towards smartphones, because cameras built into state of the art smartphones are able to capture videos at a resolution of 1080p, or even above, hence, delivering a very high quality. Furthermore, since a smartphone is always within easy reach of its owner, videos can be captured anywhere, allowing live recordings of assaults. or other acts of violence, for instance. However, while state of art technology allows high quality video capturing, it is still prohibited to use recorded video footage as witness material in court. A common argument for rejecting video footage as evidence in court is the concern, that the submitted video footage might have been manipulated. A video captured with the standard camera application of the smartphone is usually stored on the local storage of the device, so that users can easily access them for editing purposes.

Unfortunately, easy accessibility also provides the opportunity to secretly edit a video before its submission to the court, thus, no longer guaranteeing the originality of the video data. Furthermore, the video footage does not provide important information of an occurred incident, such as the time or the location. As a matter of fact, video evidence are unreliable and therefore, can not be used as digital evidence.

The aim of this work is to develop an Android app which can act as an electronic witness, so that bearing witness in court with help of video footage can be reconsidered. The app presented in this thesis allows a 1-click activation of audiovisual recording. Following the video capture, in order to avoid the video evidence becoming a target of manipulation, the app transfers the captured videos to an online data space, where it cannot be accessed for any other purposes than video playback. Since it is important to know when, and where an incident occurred, and who was involved, the time and the geographic location of the video capture are recorded. Furthermore, video frames that include faces of people are extracted, and saved as separate images. By storing the additional information associated with the recorded video in a database, a certain degree of authenticity can be ensured. Following a client-server design approach, users get a *ticket* for every video recording they own. A ticket provides users with the additional information about a video, such as time and location, and gives them the option of previewing video keyframes, playing the video, or changing the video title.

The thesis is organized as follows: Chapter 2 starts with an analysis and comparison of existing approaches to implement the application requirements. System design and implementation details are discussed in Chapter 3. Finally, the thesis is concluded in Chapter 4

CHAPTER 2

Possible approaches

2.1 Video transfer

The transfer of the recorded data to an online data space is a key feature of the Android app. The two main methods to achieve a transfer are uploading the video as an entire file, or by delivering the video as a stream of segmented packets.

2.1.1 Video Upload

To upload entire media files, basic methods to consider are the Hypertext Transfer Protocol (HTTP) [1] and the File Transfer Protocol (FTP) [2]. With HTTP, the maximum possible size of an uploaded file has to be defined. Since the duration of recorded videos vary and the file size might also differ due to different video resolutions, this limitation is a reason not to rely on HTTP. In addition to that, HTTP is not able to resume an upload, once the process is interrupted by a lost connection. FTP, on the other hand, does not have a file size limitation and is able to resume uploads, but has issues with firewalls and Network Address Translations (NATs) [3]. The FTP protocol is stateful, and requires two connections. One control connection to maintain a session, and a secondary connection for data transmission. Since the control connection stays idle during a file transmission, a long transfer period may result in NATs dropping the control connection, thus, ending the session during file transfer. In contrast, HTTP is a stateless protocol which does not depend on sessions. There are also no issues with firewalls and NATs, since the port numbers HTTP uses are well-known, and widely used.

2.1.2 Traditional streaming

The traditional streaming approach uses the Real-time Transport Protocol (RTP) [4] in conjunction with the Real-time Streaming Protocol (RTSP) [5] in order to provide live streaming of media content. RTP is a protocol that allows data transmission in real-time, while RTSP provides control over a session and the delivery of real-time data, but is not capable of data transmission. In order to start streaming, a session between server and client must be established, since RTSP is a stateful protocol. RTSP requires a specialized server to keep track of a client's state and to respond to client requests. Once the session is established, the server continuously streams data packets to the client in real time until the session ends. An advantage of this method is that a media file does not have to be downloaded in order to be played. Playback is possible, as long as the client receives enough data packets to buffer successive video frames. However, video playback may be disrupted, if a sufficient amount of data packets is not received for video buffering.

2.1.3 Adaptive streaming

State-of-the-art approaches to streaming are able to adjust the video quality depending on the current available bandwidth of the client, which is achieved by generating a set of different video qualities from the original video file. In order to allow switching between different video qualities, all generated video files are partitioned into segments with equal duration of a few seconds. The result is having short segments of the video, encoded in various bit-rates. In contrast to the traditional streaming approach, which depends on one video source, thus, on video quality, this approach is more flexible. Following a change of bandwidth on the client side, switching to a lower or higher video quality during video playback by requesting video segments of different qualities avoids disruptions, and reduces buffering time. However, choosing a selection of bit-rates to encode in, as well as defining a certain duration for each video segment is important. Since everything is hosted on a server, the shorter a video segment is, the more files for one video quality have to be hosted. The protocol used to for data transmission is HTTP over TCP. Advantages of using HTTP include a reliable transmission as well as congestion control. HTTP also avoids complications with firewalls and NATs, and does not require any specialized server infrastructure. There are several HTTP-based streaming solutions existing.

HTTP Live Streaming (HLS)

HLS [6] is a protocol implemented by Apple that allows live streaming or streaming of multimedia data. HLS creates alternative files by encoding, and segments them into smaller files, formatted as MPEG-2 Transport Streams (M2TS). For media playback, a playlist file in .m3u8 format is used. The playlist file fulfills the role of an index file, referring to the playlist of different video qualities, while those playlists contain links to the segments in the respective video quality. In case of live streaming, the client also receives a m3u8 file format playlist, but the difference is that the playlist is continuously updated, as new segments are added to the playlist. To deploy HLS, a server supporting HLS is required.

IIS Smooth Streaming

Smooth Streaming [7] is a protocol introduced by Microsoft for streaming purposes. While Smooth Streaming also defines segments, namely chunks, the server only has to host one video file for each bit-rate the original video has been encoded in. Each chunk represents a MPEG-4 fragment that is stored within a contiguous MP4 file. New file extensions are introduced with Smooth Streaming in order to differentiate a regular MP4 file from a fragmented MP4 container format. With this specification, the server can grab a chunk upon client request, create a file containing the chunk data, and transfer it as a standalone file. A difference to HLS is that a chunk starts with a keyframe, which allows seamless switching between different qualities. A comparison between the MPEG-2 Transport Stream and the Fragment MP4 can be in a white paper written by Timothy Siglin [8]. Unlike most HTTP-based adaptive streaming solutions, Smooth Streaming effectively avoids the problem of having too many files by storing the encoded videos in full-length rather than in segments. However, in order to deploy Smooth Streaming, the proprietary encoding tool *Expression Encoder* is required.

HTTP Dynamic Streaming (HDS)

HDS [9] is a streaming solution implemented by Adobe that supports streaming over Adobe's proprietary Real Time Messaging Protocol (RTMP) and HTTP. Similar to Smooth Streaming, HDS partitions a media file into chunks, and is able to save them as one, or multiple files using their .f4f file format. Unfortunately, deploying HDS can only be achieved by using standard Apache server software and caching infrastructures.

Dynamic Adaptive Streaming over HTTP (DASH)

DASH [10] is an standardized adaptive streaming solution developed by MPEG. Unlike HLS, Smooth Streaming, and HDS, DASH does not require specialized server infrastructure or proprietary tools for deployment. Similarly to Smooth Streaming, a media file is partitioned into chunks, with each chunk being encoded in a selection of different bit-rates. Period is the term the DASH protocol uses to name a chunk. DASH also uses a fragmented MP4 container format where each chunk start with a keyframe, and defines the media content as a sequence of consecutive chunks that do not overlap. DASH uses Media Presentation Description (MPD), a XML-based document, to define the periods, the video quality profiles, namely representation, for each period, and to store segment information, such as the URL to a certain segment source.

2.2 Location detection

Since an ideal electronic witness provides information about the location an incident occurred, a location detection solution for Android is required. There are two Application Program Interfaces (APIs) available that provide the option of accessing location information of an Android device, both implemented by Google.

2.2.1 Android Location API

The Android Location API [11] allows devices to receive information about the current geographic location, and to register listeners that respond to location changes. To start with, a location provider has to be declared, because the API is not able to determine on its own, which location providers are available, and provider should be used at which time, or place. The providers to choose from are either Global Positioning System (GPS), cell towers, or Wi-Fi networks.

2.2.2 Google Location Services API

The Google Location Services API [12] is a newer API that offers more options to improve accuracy and power consumption. In contrast to the Android Location API, this API is able to get the best available location, based on available location providers. For instance, because GPS are not suitable for indoor location, the Google Location Services API is able to get the location information from other available providers such as cell towers or Wi-Fi networks. In addition to that, the priority of a location request can be defined in order to reduce power consumption. A higher priority delivers precise location information, e.g via GPS, but also consumes more power, whereas a middle, or lower priority consumes less power, but only delivers an accuracy of approximately 100 meters, or 10 kilometers respectively. A thing to note is that all Google Play services are delivered through the Google Play Store, thus, a connection has to be established first. Therefore, Android devices without the Google Play Store, or Android devices that are located in countries, where the use of Google services are restricted, e.g China, unfortunately are not able to make use of this API.

2.3 Face detection

Once a video file is uploaded and hosted by the server, users of the app shall receive a ticket for each recorded video with a keyframe description that describes an important scene in the video. In particular, in a video that involves filming people, the faces of the people can be considered the most important content in the video. Thus, for the extraction of human faces, the App requires the implementation of face detection.

2.3.1 Android face detection

Android provides the Camera.FaceDetectionListener interface [13], which is part of the Android Camera API, in order to detect faces during a video capture.

2.3.2 Face API

The Face API [14] is a component of the Mobile Vision API, which was recently added with the rollout of Google Play services 7.8, at the time of writing. According to Google, it is faster, more accurate, and provides more information than the FaceDetector.Face

API. The Face API added face tracking mechanisms to detection faces in a video, which becomes important, if there is a need to differentiate between two or more faces. It is also to note that even though face tracking is possible, the Face API provides no functionality for face recognition. However, what the API provides are classifications of facial characteristics. The classifications available are *open eyes* and *smiling*, with more likely to be added in the future. A minor drawback of using the Face API is that the API is a fairly new addition, so as a result, there are some errors to be fixed.

2.3.3 OpenCV4Android

OpenCV [15] is a library that provides functionality for real-time computer vision. The library is written in C/C++, which is why the Android Java wrapper, OpenCV4Android, has been introduced for development on Android. Although OpenCV has a steep learning curve, it allows the implementation of functionalities for various application areas of computer vision. However, one thing to note is that OpenCV does not have support for hardware acceleration on the Android operation system, while the Standard Android API and the Face API do.

CHAPTER 3

Implementation

3.1 Functional requirements

- Audiovisual recording with minimal latency
- Recording the location and the local time at recording time
- Face detection at recording time
- Transfer of the recorded data to an online data space, where the content cannot be accessed by anybody but the system administrator.
- A user gets a ticket with the video entry and a keyframe descriptions of the recorded data for each video that he, or she, has recorded, and uploaded.

3.2 Non-functional requirements

- Support for different platform versions of Android
- Support for different screen sizes
- Providing an user interface that is intuitive to use

3.3 System design

Given the requirements, three classes - User, Video, and Keyframe - can be extracted, as illustrated in Figure 3.1. The User class makes sure that a certain video can only be viewed by the person who recorded it. The easiest solution is achieve that requirement is to introduce user management by providing user account creation, and using common credentials, such as username and password, to sign in. The Video class represents an electronic witness, and provides a reference to the resource hosted on the server, as well as information about the date, the time, and the location of recording. Additionally, each video is associated with 3 video frames that summarizes the video content best. Thus, the Keyframe class provides a reference to the video keyframe, which simply is an image.

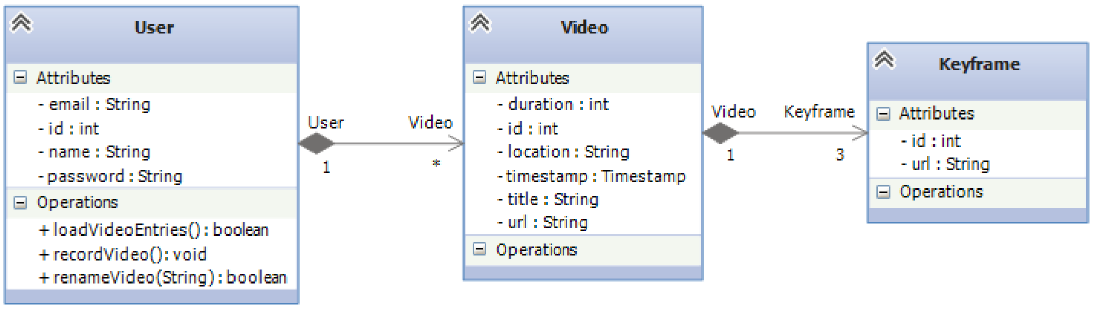

Figure 3.1: Class diagram of Witestify.

In order to meet the functional requirements, an end user of the Android app, first of all, must be able to create an account, and to sign in. Once signed in, the app should load all videos the user has recorded into a list, and display them as item cards. An item card displays all the relevant information such as date, time, and location of the recording, and also includes a keyframe description. Every item of the list provides the options the view all keyframes, to change the video title, or to play the recorded video. By default, the videos are sorted by most recently recorded videos, but, alternatively, the user can select another sort criteria. Aside from the default sorting function, available criterias are the video title, and video duration. The most important use case is capturing a video which is immediately uploaded to the host server afterwards. The server's main tasks are handling client requests, creating, reading, and updating database entries.

The use cases, and their dependencies, are summarized in Figure 3.2, while Figure 3.3 outlines the lifecycle of the app. When the button to start recording is clicked, the app will check whether the network services have been enabled, so that location information can be accessed. Some Android devices may require enabling location services in addition to network services. In that case, the app will open a dialog window and prompt the user to do so. The camera preview surface to start recording is only displayed, if location information can be accessed. Since an electronic witness must contain information about when an incident occurred, it is most reasonable to measure date and time at the start of recording, because the video duration provides information of when the recording has ended. After recording has ended, the user receives an notification displaying the upload progress at the same time the upload of the video and the keyframes is initiated. The upload procedure is intended to run in the background, so that the user can start another recording session, or use other apps meanwhile. Once uploading is done, a new video entry is added to the database, and the videos are re-loaded with the default sorting algorithm, to show the new electronic witness on the top of the list.

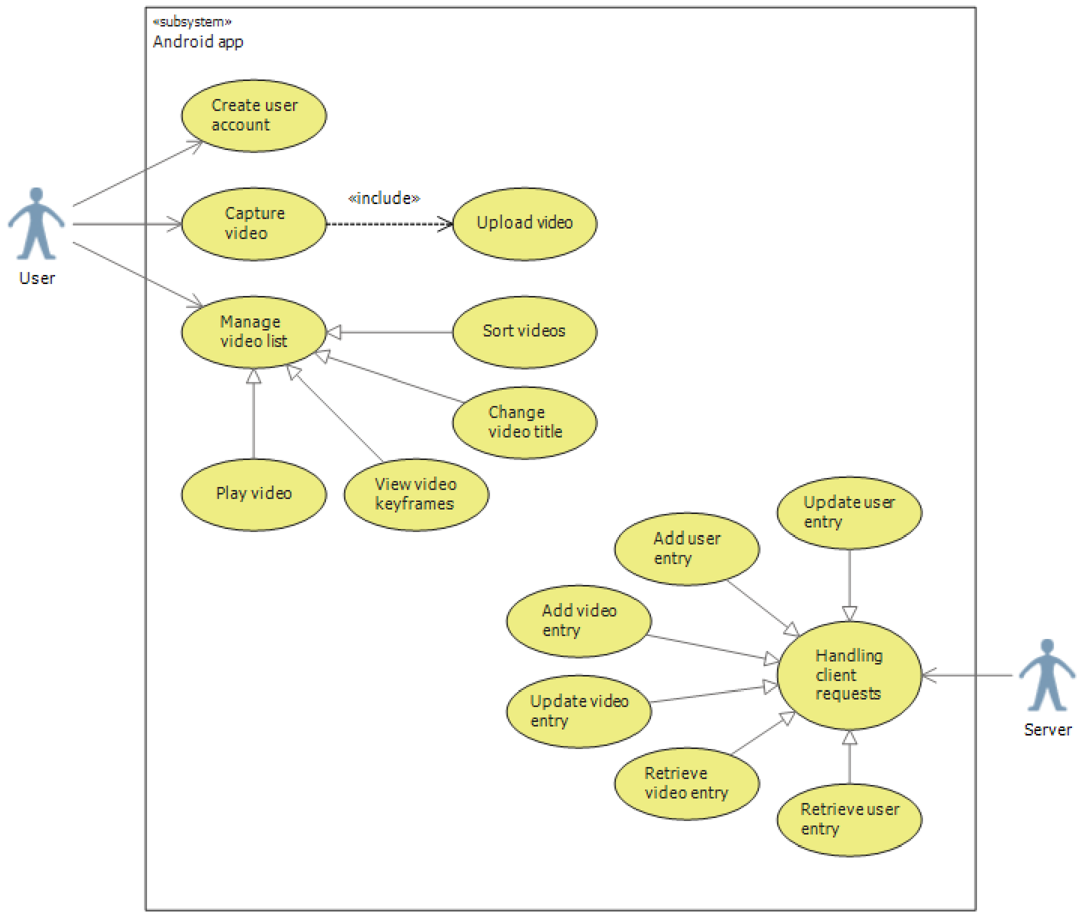

Figure 3.2: Use case diagram of Witestify.

3.3.1 Video capture

The video recorder in the Android app is implemented with the help of the Camera API. Using the OpenCV library is another way to approach implementing a video recorder, but the Camera API is functionally adequate for the needs of the Android app. Another reason to use the Camera API is that it also provides an callback interface for detected faces.

3.3.2 Face detection

Face detection (Section 2.3.1 is natively supported with the callback interfaces of the Camera API. The other options to choose from are the Face API or the OpenCV library. While the Face API is a little faster at detecting faces, the new API has revealed to have issues that have to be fixed. In that regard, Android face detection is more stable. The reasoning behind not choosing OpenCV is that the face detection speed depends highly

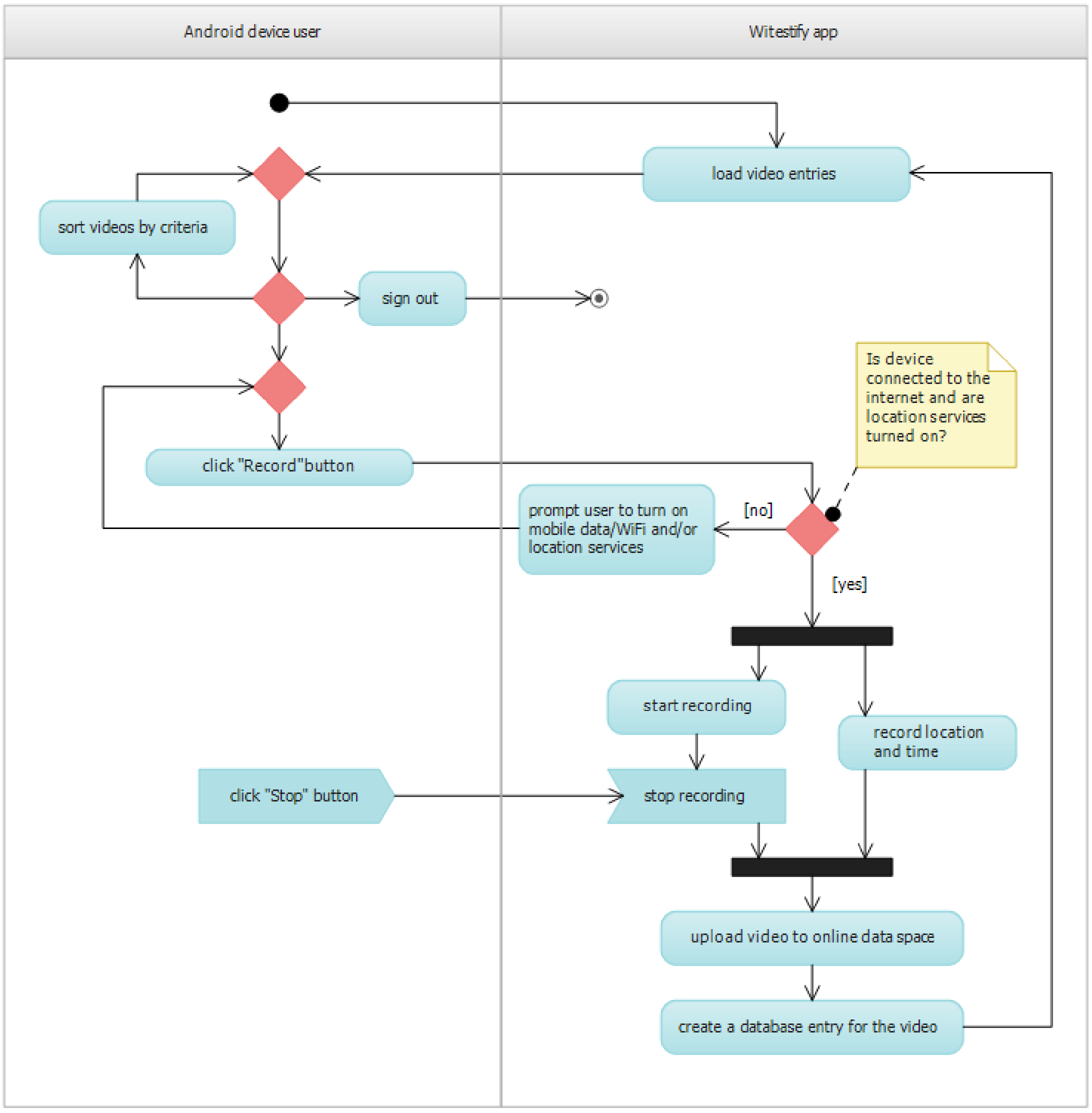
Android device user
Witestify app
load video entries
sort videos by criteria
sign out
Is device connected to the internet and are location services turned on?
click "Record"button
prompt user to turn on mobile data/WiFi and/or location services
[no]
[yes]
start recording
record location and time
click "Stop" button
stop recording
upload video to online data space
create a database entry for the video

Figure 3.3: Activity diagram of Witestify.

on the running Android device's CPU, as hardware acceleration is not supported.

3.3.3 Location detection

For location detection purposes, the Google Location Services API (Section 2.2.2) is chosen over the Android Location API (Section 2.2.1). Even though the Google Location Services API comes with the drawback of requiring a connection to the Google Play Store, the advantages outweigh the drawback. The Google Location Services API is able to determine the location of an Android device and switches location providers automatically depending on an indoor, or outdoor position, delivering the best, and most accurate location information possible. Being able to balance between location accuracy and power-consumption through priority settings is also an important factor, because video recording also takes a heavy toll on the Android device's battery life.

3.3.4 Video transfer

The best solution for video transfer would be streaming the live camera images to a host server using adaptive streaming (Section 2.1.3). However, HLS, Smooth Streaming, and HDS all require either specialized server infrastructure, or proprietary tools, which leaves DASH as the only option. Unfortunately, DASH is designed for streaming from server to client rather than from client to server. A video would have to be encoded in different bit-rates, partitioned into the segment files, and MPD files have to be available in order to begin streaming with DASH. Which in other words means that streaming to the server while recording is not possible, because the video file would have to exist in full-length for segment files to be created.

Since the Android SDK does not natively provide any functions to partition a video source file into DASH segments, the decision towards uploading the video and the keyframes either via HTTP or FTP had been made. In the end, FTP was chosen over HTTp, because data transfer is resumed in case of a failed upload attempt, and file size limit is not as clearly defined as it is the case with HTTP.

3.4 Libraries

The application includes two 3rd-party libraries. ExoPlayer [16], the first library, provides a media player which also supports DASH and Smooth Streaming playback. The ExoPlayer is used to play the recorded videos that have been uploaded to the host server.

The other library used by the application is Volley [17], a networking library. Volley allows sending and cancelling network requests, provides network requests scheduling and request prioritization. Volley works in asynchronous fashion and is able to run requests concurrently. In general, the application uses Volley to send client requests to the server, as well as to receive the server responses again. The requests include user sign up, user sign in, retrieving and updating video entries from the database, and loading keyframe images.

CHAPTER 4
Conclusions

This thesis introduced Witestify, an Android app that allows its user to create electronic witnesses. Possible approaches to meet the application requirements were analyzed, and compared with each other. Finally, an system overview was given, and implementation details were discussed.

Although all application requirements are met, there is room for improvement regarding the implementation. The video recorder could be upgraded by adding additional functionality such as zoom, or white balance. Regarding video transfer, the solution of using FTP for data transmission should be changed to another, more secure transferring technique. If possible, the alternative transferring method should allow to stream the camera preview live from the Android device to server, so that a recorded video does not have to be temporarily stored on the device, creating an opportunity for manipulation. Furthermore, face detection can be improved by replacing it with the Face API, once the issues are fixed, and the API gets recommended for developing purposes.

List of Figures

Bibliography

[1] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999. `http://www.rfc-editor.org/rfc/rfc2616.txt`.

[2] J. Postel and J. Reynolds. File Transfer Protocol. STD 9, RFC Editor, October 1985. `http://www.rfc-editor.org/rfc/rfc959.txt`.

[3] Pyda Srisuresh and Matt Holdrege. Ip Network Address Translator (NAT) Terminology and Considerations. RFC 2663, RFC Editor, August 1999. `http://www.rfc-editor.org/rfc/rfc2663.txt`.

[4] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, RFC Editor, January 1996.

[5] Henning Schulzrinne, Anup Rao, and Robert Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, RFC Editor, April 1998. `http://www.rfc-editor.org/rfc/rfc2326.txt`.

[6] Roger Pantos and William May. HTTP Live Streaming. Internet-Draft draft-pantos-http-live-streaming-16, IETF Secretariat, April 2015. `http://www.ietf.org/internet-drafts/draft-pantos-http-live-streaming-16.txt`.

[7] Microsoft Corporation. Smooth Streaming Protocol. `http://msdn.microsoft.com/en-us/library/ff469518.aspx`, 2010. Accessed September 19, 2015.

[8] Timothy Siglin. Unifying Global Video Strategies: MP4 File Fragmentation For Broadcast, Mobile and Web Delivery. *A Transitions in Technology White Paper*, 2011.

[9] Adobe Systems Inc. HTTP Dynamic Streaming. `http://www.adobe.com/products/hds-dynamic-streaming.html`, 2011. Accessed September 19, 2015.

[10] ISO/IEC. *ISO/IEC 23009-1:2014. Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats.* ISO/IEC, 2014.

[11] Google. Android Location API. `http://developer.android.com/reference/android/location/package-summary.html`. Accessed September 19, 2015.

[12] Google. Google Location Services API. `http://developers.google.com/android/reference/com/google/android/gms/location/package-summary`. Accessed September 19, 2015.

[13] Google. Camera.FaceDetectionListener. `http://developer.android.com/reference/android/hardware/Camera.FaceDetectionListener.html`. Accessed September 19, 2015.

[14] Google. Face API. `http://developers.google.com/android/reference/com/google/android/gms/vision/face/package-summary`. Accessed September 19, 2015.

[15] Itseez. OpenCV for Android. `http://opencv.org/platforms/android.html`. Accessed September 19, 2015.

[16] Google. ExoPlayer. `http://developer.android.com/guide/topics/media/exoplayer.html`. Accessed September 19, 2015.

[17] Google. Volley. `http://developer.android.com/training/volley/index.html`. Accessed September 19, 2015.