Remote Collaborative Real-Time Multimedia Experience over the Future Internet

ROMEO

Grant Agreement Number: 287896

D5.4

Report on User Generated Content Provisioning

Document description	
Name of document	Report on User Generated Content Provisioning
Abstract	This document describes the ROMEO *User Generated Content*, including its role in the ROMEO system, its architecture, functionalities and user interfaces. The *User Generated Content* allows ROMEO users to create, upload, download, search and share content with collaborating users. To achieve high compatibility amongst devices and operating systems, the ROMEO *User Generated Content* also implements a professional transcoding service, triggered by a watch folder script that automatically submits new content to a transcoding-server, who creates a re-encoded copy of the content according to a pre-defined profile.
Document identifier	D5.4
Document class	Deliverable
Version	v1.3
Author(s)	H. Marques, Jonathan Rodriguez (IT) O. Altunbas, C. Ozkan, E. Cimen, A. Akman (TTA) H. Gökmen, G. Tanık, B. Demirtaş (ARCELIK) M. Schmalohr, M. Laabs, S. Petersen (IRT) K. Georgiev, N. Daskalov, V. Petrov, E. Angelov (MMS)
QAT team	V. de Silva (US), J. Lauterjung (R&S), F. Pascual (TID)
Date of creation	31-July-2013
Date of last modification	30-Jan-2014
Status	Final
Destination	European Commission
WP number	WP5
Dissemination Level	PUBLIC
Deliverable Nature	Report

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Purpose of the Document

The purpose of this document is to describe the work developed in WP5, specifically in Task 5.5, regarding the provisioning of the ROMEO *User Generated Content* (UGC) component.

1.2 Scope of the Work

ROMEO considers the case in which users can create and make available content to collaborating users. This service is enabled through the use of the UGC component which includes the design of client (fixed, portable and mobile) and server modules as well as two graphical user interfaces (GUI). The first GUI allows authenticated users to upload, download, search/list and delete content whereas the second GUI enables professional content transcoding service. This, create and sharing capability, can be performed while ROMEO users receive the 3D professional content, which is expected to enrich the whole immersive experience that ROMEO will bring.

1.3 Objectives and Achievements

The main objective of Deliverable 5.4 is to report the provisioning of the UGC component that allows ROMEO users to create/edit their own content to share with the collaborators. The main achievements are the definition of the UGC architecture and the design of the client (fixed, portable and mobile) and server modules, developed in WP6, that implement the UGC functionalities. In addition to typical file operations, the UGC also includes an automatic, profile based, transcoding service. By enabling this functionality, ROMEO minimises compatibility issues associated with different user-submitted codecs, containers and playout systems. This also provides collaborating users a higher quality of experience (QoE).

1.4 Structure of the Document

This document is structured as follows: Section 2 describes the role of the UGC in ROMEO; Section 3 provides the details of the UGC, including its architecture, functionalities and user interfaces and; Section 4 presents the conclusions.

2 THE ROLE OF USER GENERATED CONTENT FUNCTIONALITY IN ROMEO

The ROMEO project aims at providing 3D multi-view content to both mobile and fixed users. While providing such rich and high quality content, it also aims at allowing them to enjoy the content jointly by supporting communication and social interaction between the users. With this respect, ROMEO's *User Generated Content* (UGC) sharing functionality is a major capability which enriches the user interaction and empowers the concept of socializing around TV content.

Within the scope of the UGC sharing functionality, ROMEO project will allow its users to share personal contents with a set of users which is to be defined by the content owner. With such a capability, users' joint TV experience will be improved since it will be possible to support the main content with additional custom contents while being in contact with the selected users. For this purpose, the ROMEO project implements a client-server based UGC sharing system which works in close coordination with the Audio Visual Communication Overlay component of the project, which also aims at improving ROMEO user interaction during the live streaming.

The UGC architecture was designed and developed to be independent from the ROMEO P2P overlay architecture. ROMEO users have a direct communication path to the UGC server (for uploading/downloading content) regardless of their position on the P2P overlay and access technology. The reasoning behind this decision is to minimize the impact of the UGC data flow - which is delay tolerant - on the P2P content delivery - which is bandwidth and delay sensitive. This design also facilitates the implementation of a central UGC authorization mechanism. The data delivery paths for the different ROMEO services are depicted on the overall ROMEO System Architecture described on deliverable 2.3. [1], Appendix C, section 1, figure 59.

3 USER GENERATED CONTENT DETAILS

3.1 General Architecture

The UGC architecture consists of a UGC client, a UGC Server, a File Server and a Transcoding Server, as depicted in Figure 1. The core UGC functionality is implemented as a REST Server/Client Architecture. The UGC Server runs an Apache Web Server with ZEND PHP Framework and the UGC client runs a Java based HTTP client. The file server is where the users' uploaded content is actually stored while the Transcoding Server is responsible for automatic content transcoding via a watch folder shell script.

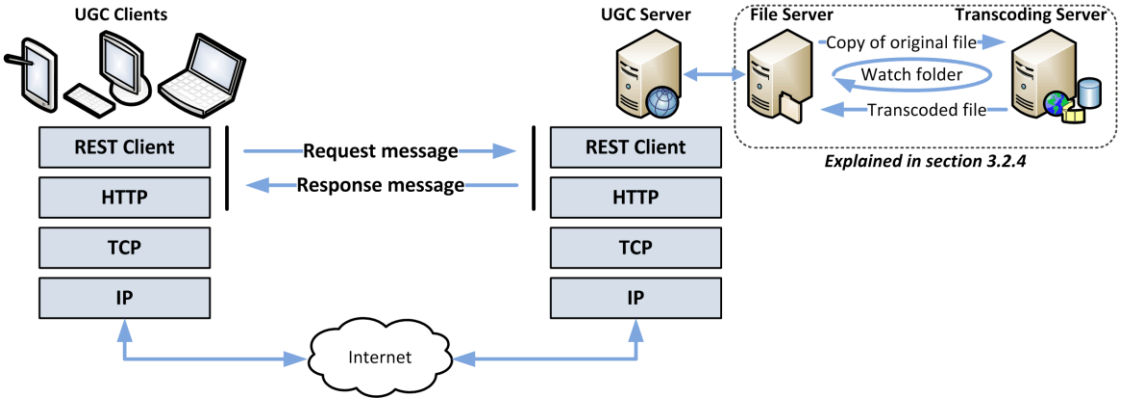

Figure 1 - Block diagram of the User Generated Content ROMEO component

The UGC mainly interacts with the "UI & Control" module at the client side and "Authentication, Registration and Security" module at the server, as can be seen in Figure *2*. Details on the interaction between the File and Transcoding Servers are given in section 3.2.4.

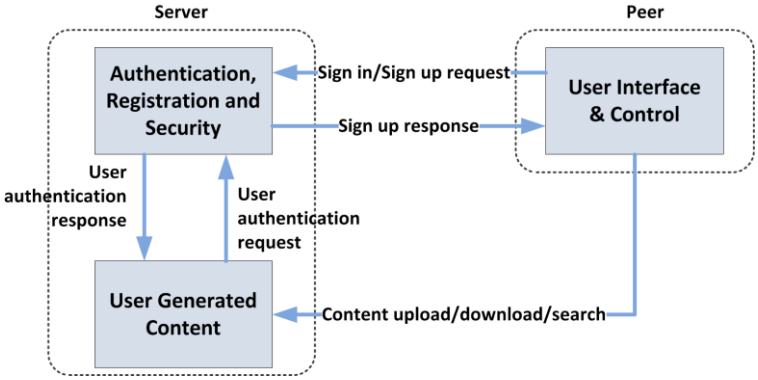

Figure 2 - UGC interactions

To provide the functionalities defined in ROMEO deliverable 2.2 [2], the UGC server makes use of the Zend Framework, RESTful web services and a MySQL database. The Zend Framework [4] is an open source, object oriented web application framework for high-performing PHP applications. The Zend Framework offers a robust, high performance Model-View-Controller (MVC) implementation, a database abstraction that is simple to use and implements HTML for rendering, validation, and

filtering so that developers can consolidate all of these operations using one easy-to-use object oriented interface. Representational State Transfer (REST) [5] was used because the UGC design required the use of a client-server architecture that separates the user interface from the data storage. REST-style architectures conventionally consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. The HTTP protocol was used to implement the REST pattern. Using the HTTP protocol to implement the REST pattern means that every resource can be represented as a unique Uniform Resource Identifier (URI) and accessed through a set of HTTP verbs.

The REST stateless constraint requires application state to be maintained exclusively by the client. The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. Each message contains all the information necessary for the service to perform its function, and isn't reliant on a previous exchange.

A RESTful web service, also called a RESTful web Application Programming Interface (API), is a web API implemented using HTTP and REST principles. It is a collection of resources, with four defined aspects:

- the base URI for the web API - such as http://localhost/file/list;
- the Internet media type of the data supported by the web API - this is often JavaScript Object Notation (JSON) but can be any other valid Internet media type provided as a valid hypertext standard;
- the set of operations supported by the web API using HTTP methods (e.g., GET, PUT, POST, or DELETE);
- the API must be hypertext driven.

In the proposed UGC design, the uniform interface requires all interactions between clients and services to be mediated through HTTP methods (GET, PUT, DELETE and POST).

3.2 Provided functionalities

The UGC ROMEO component provides the following functionalities:

- Content upload
- Content download
- Content list/search
- Content delete
- Content transcoding

The API design is based on the abstract class *Zend_Rest_Controller* which guides implementation of action controllers. Each controller has 5 methods by default:

- void deleteAction()
- void getAction()
- void indexAction()
- void postAction()
- void putAction()

These methods/functions were redesigned to upload/download and do other file operations as explained in next subsections. The UGC server API functional overview is shown in Table 1.

Table 1 - UGC Server API Overview

Resource	GET	POST	PUT	DELETE
File	Download File	Upload File	(not required for file operations)	Delete File

3.2.1 Content upload / download

3.2.1.1 Download file

The download file operation allows authenticated users to download authorized files. For this operation, the user has to send a GET request with the required parameters as shown in Table 2. Whenever the server receives the GET request, the *getAction()* method is executed to trigger the download.

Table 2 - Required Fields for the download file API

Field Name	Description
name	Name of the user downloading the content
file_id	ID of the file returned from a search query
transcoding_type	Type of the device specific file transcoding
signature	Signed with the user private key provided from a license file

According to a ROMEO design choice, the requests sent from the users are handled by a file controller in the server. At first there is a null check and validation of all required fields. If there is a missing field the response will be “STATUS”; “NOT_OK” with “WRONG_API_FORMAT” error.

The next procedure is to check the authentication of the user. For this purpose, at first, the user name is checked, and if it doesn’t exist in the “users” table, the response will be “STATUS”; “NOT OK” with “USER_NOT_EXISTING” error. Secondly, the user authorization is checked. If the user authentication fails, due to mismatch password, the response will be “STATUS”; “NOT OK” with “AUTHENTICATION_FAILED” error.

Afterwards the requested file is checked for user access rights. If the user has no access rights to read or write, the response will be “STATUS”; “NOT OK” with “USER_NOT_AUTHORIZED” error.

At the server, the uploaded files are represented with a unique *file_id*, given by the system, stored in the database. This unique *file_id* matches the real file the user has uploaded before. In order to retrieve the file from the system, the file path information has to be gathered from the database. The *file_id* received from the user is used to query the database and get the path information of the requested file. The result of the query may become null, then the response will be “STATUS”; “NOT OK” with “NO_FILE_IN_DATABASE” error. If the result is not null, the existence of the file is

checked. If the file does not exist on the retrieved file path, the response will be “STATUS”; “NOT OK” with “FILE_NOT_EXIST” error.

Up to this point, if everything is OK, then the server prepares HTTP response headers for the file transfer and sends the file to the user. An example of the API call is:

```
http://localhost/api/file/1?name=neil&file_id=file51346bfe97f63&transcoding_type=0&
signature=3d7e4e9945f76c1d05e1ab5ed19d59a61e3514d6ef2004123cde7e99913a2
284
```

Table 3, provides a summary of the return values provided by the download file API running at the UGC server.

Table 3 - Return values for the download file API

Message	**Description**
Contents	Contents of the downloaded file if successful
STATUS	OK for success, NOT_OK if an error exists.
ERROR	Reason for the error (exists only if status is NOT_OK)
WRONG_API_FORMAT	Check if all required fields are filled.
AUTHENTICATION_FAILED	The signature from user does not match the signed content in server
USER_NOT_EXIST	No such user exists in the database
USER_NOT_AUTHORIZED	User does not possess the rights to download the file
NO_FILE_IN_DATABASE	A file with the given id does not exist anymore
FILE_NOT_EXIST	File is not available in the server
INVALID_TRANSCODING_TYPE	Transcoding type is not known by the server

3.2.1.2 **Upload file**

The upload file operation allows authenticated users to upload content to the server. This operation uses the HTTP POST method with the parameters shown in Table 4. Whenever the server receives the POST request, the *postAction()* method is executed to trigger the content upload. By default the UGC server allows to upload a maximum file size of 200MB, defined in *php.ini* configuration file.

Table 4 - Required fields for the upload file API

Field Name	**Description**
name	Name of the user uploading the file
user_names	Names of the users entitled to download the file
file_name	Name of the file to be uploaded including the file extension
signature	Signed with the user private key provided from a license file

When the POST request is received by the UGC server, the handler checks the *file_id*, which is server side generated id for a given *file_name*. If it exists, the file upload API treats this as a file update procedure. If it does not exist, the API considers this a new file. Whatever the case there is a null check and validation of all required fields. If there is a missing field, the response will be “STATUS”; “NOT_OK” with “WRONG_API_FORMAT” error. Then the user authorization is checked. If user

is not authenticated the response will be “STATUS”; “NOT_OK” with “AUTHENTICATION_FAILED” error.

After the successful authorization the code checks file upload error list. If there is an error the response will be “STATUS”; “NOT_OK” with “FILE_UPLOAD_ERROR_” + Upload Error Code. If no errors are encountered the uploaded file is moved to the pre-defined upload path and additional information is added to the database. If there is an abnormal condition while accessing to the database the response will be “STATUS”; “NOT OK” with “FILE_DATABASE_ACCESS_FAILED” error. Then a *file_id* is generated from the exact *file_name* and it is inserted to the database to map who has access rights to the file.

An example of this API call is:

```
http://localhost/api/file?name=neil&user_names=neil,yuri&file_name=doc1&signature
=5ed1a45d65c98abbba0dc6f530219831a764d06f46e4a1efb98f93a4f9276af4
```

Table 5, provides a summary of the return values provided by the upload file API running at the UGC server.

Table 5 – Return values for the upload file API

Message	**Description**
STATUS	NOT_OK if failed, OK if successful
FILE_ID	ID of the created file
FILE_NAME	Name of the file created same as the sent
ERROR	Reason for the error (exists only if status is NOT_OK)
WRONG_API_FORMAT	Check if all required fields are filled
AUTHENTICATION_FAILED	The signature from user does not match the signed content in server
USER_NOT_EXIST	No such user exists in the database
FILE_UPLOAD_ERROR	An error occurred in the file upload process
FILE_DATABASE_ACCESS_ FAILED	Cannot access the file database
USER_RIGHTS_DATABASE_ ACCESS_FAILED	Could not update the user rights database

An example of the returned JSON string when status is NOT_OK:

```
{       "resources": {
                "STATUS": "NOT_OK",
                "FILE_ID": "",
                "FILE_NAME": "myfirstvideo",
                "ERROR": "AUTHENTICATION_FAILED"        }
}
```

3.2.2 **Deletion of content**

The delete file operation allows authenticated users to delete content. This operation uses the HTTP DELETE method with the parameters shown in Table 6.

Table 6 – Required fields for the delete file API

Field Name	Description
name	Name of the user requesting the deletion of the file
file_id	ID of the file to be deleted
signature	Signed with the user private key provided from a license file

When the UGC server receives a DELETE request, the handler checks the 'file_id' parameter to update the server resource state value. After the null check of required parameters, it checks the user authentication. If the provided ‘name’ and ‘signature’ do not match with UGC server users’ table, the response will be “STATUS”; “NOT_OK” with “UNAUTHORIZED_USER” error. If the authentication succeeds the handler checks if the user is the owner of the file. If it is not, the response will be “STATUS”; “NOT_OK” with “NOT_OWNER” error. If the user is the owner then the handler proceeds to delete the file. If the deletion is successful, the response will be "STATUS": "OK".

An example of this API call is:

```
http://localhost/api/file?file_id=file513771f3476ba&name=neil&signature=af059248e3e178a1a35f2f2e0536b1a82eeaa34bcf25ae453d4f5f9b60d8e00f
```

Table 7, provides a summary of the return values provided by the delete file API running at the UGC server.

Table 7 – Return values for the delete file API

Message	Description
STATUS	NOT_OK if failed, OK if successful
ERROR	Reason for the error (exists only if status is NOT_OK)
WRONG_API_FORMAT	Check if all required fields are filled
AUTHENTICATION_FAILED	The signature from user does not match the signed content in server
NOT_FILE_OWNER	The provided user is not the creator of the document
FILE_NOT_AVAILABLE	Such a file name does not exist
NO_DATABASE_ENTRY	Such an entry does not exist in the database
NOT_DELETED	Existing file could not be deleted
FILE_DATABASE_ACCESS_FAILED	Cannot access the file database
USER_RIGHTS_DATABASE_ACCESS_FAILED	Could not delete the user rights database

An example of the returned JSON string when status is OK:

```
{    "resources": {
        "STATUS": "OK"
}}
```

3.2.3 List /Search content

The list/search file operation allows users to list/search uploaded content at the content repository. This operation uses the HTTP GET method with the parameters shown in Table 8.

Table 8 – Required fields for the search file API

Field Name	Description
name	Name of the user requesting the search
signature	Signed with the user private key provided from a license file

An example of this API call is:

```
http://localhost/api/file?name=neil&signature=3d7e4e9945f76c1d05e1ab5ed19d59a6
1e3514d6ef2004123cde7e99913a2283
```

Table 9, provides a summary of the return values provided by the list/search file API running at the UGC server.

Table 9 – Return values for the list/search file API

Message	Description
STATUS	status of the call OK → if successful NOT_OK → if failed
FILES	list of available files
FILE_NAME	Name of the shared file
FILE_ID	Server-generated unique id of the shared file
FILE_OWNER	User name of the owner of the file
FILE_SIZE	Size of the file
UPLOAD_DATE	File upload date time
ERROR	Reason for the error (exists only if status is NOT_OK)
WRONG_API_FORMAT	Check if all required fields are filled
AUTHENTICATION_FAILED	The signature from user does not match the signed content in server.
USER_NOT_EXIST	No such user exists in the database

An example of the returned JSON string when status is OK:

```
{    "resources": {
        "STATUS": "OK",
        "FILES": [{
            "file_name": "_file_name",
            "file_id": "_file_id",
            "file_owner": "_file_owner",
            "file_size": "_file_size_in_kbytes",
            "upload_date": "_upload_date"
        }]
}}
```

3.2.4 Video transcoding

The ROMEO UGC also supports the capability to transcode the uploaded videos before they are made available to other ROMEO users. The server for this process is called transcoding-server. It is conceptualised as a 3-tier architecture and connected to the upload-folder from the UGC server, as depicted in Figure 3. It offers a command-line and the possibility to watch the upload folder permanently for new files to encode, but also offers a web-interface to create and edit encoding-profiles and starting encoding-jobs.

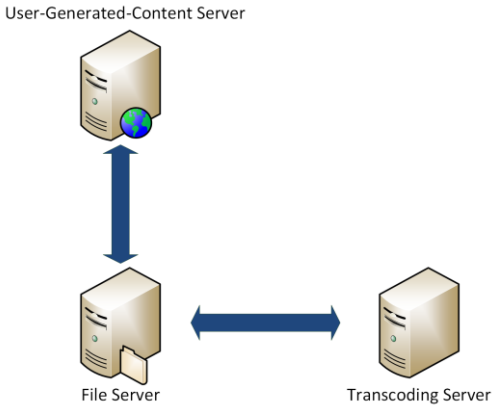

Figure 3 - The environment of the transcoding-server

3.2.4.1 Transcoding-server architecture

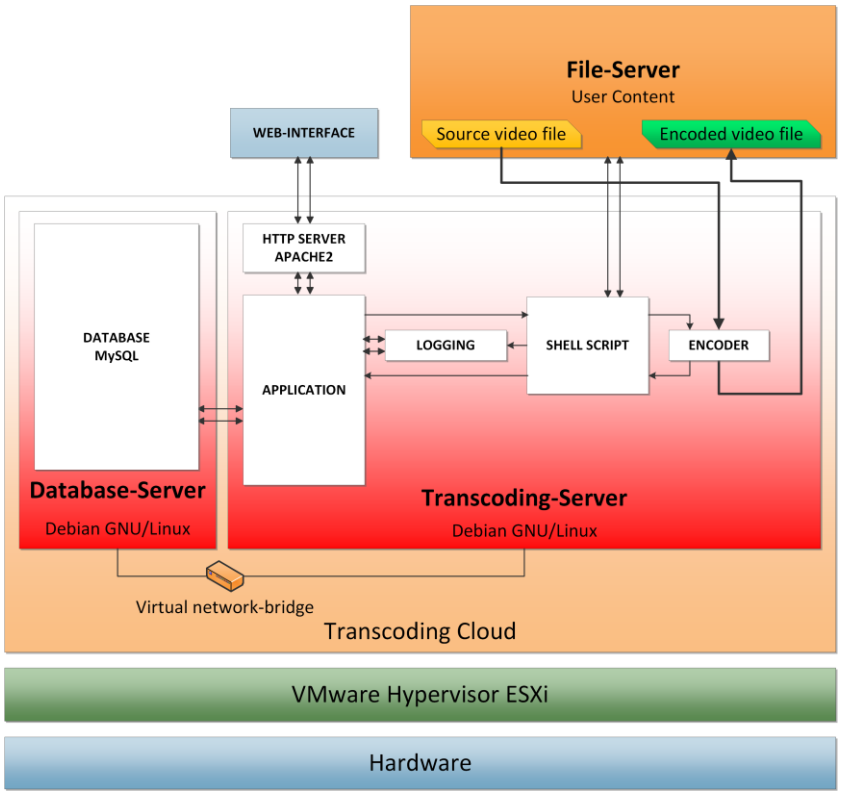

Figure 4 - The virtualisation-environment and software-architecture

The hardware for the transcoding-server is a Server with two Intel Xeon 8-Core CPUs and 32 GB RAM. There is a VMware Hypervisor ESXi-Server running as virtualisation operating system. In this environment there are two Debian GNU/Linux machines installed which contain the applications for the transcoding-process. These machines are the cores of the transcoding-cloud. The virtualisation-environment and software-architecture can be seen in Figure 4.

3.2.4.2 **Software-architecture of the transcoding-server**

Figure 5 shows the architecture of the transcoding-process. The shell-script gets the uploaded files either via a web-interface or from the watch folder on the File-Server. The transcoding-application manages the transcoding-process and creates a queue so that video-files can be treated sequentially. Transcoding includes the applicable pre-processing, de-coding and re-encoding of each video-file. After the final encoding process, the files will be uploaded back to the file-server.

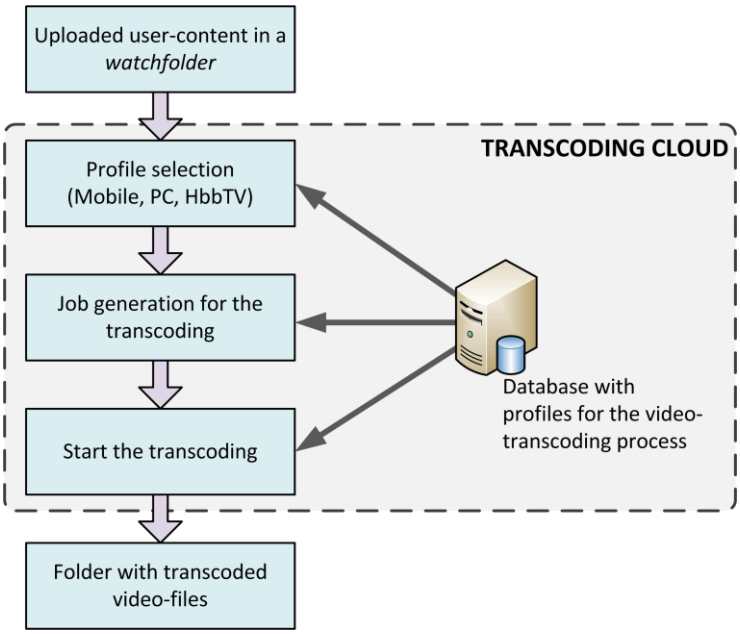

Figure 5 – The flow of the transcoding process

3.2.4.3 **The actual transcoding-process**

The communication between the main-application and the transcoding-engines is managed by a dynamically generated shell-script. It handles the demultiplexing of the original user-video-file, the video- and audio-encoding and the multiplexing by the MP4Box tool[1].

The transcoding begins with demultiplexing the source video file into a video and an audio file. The files are transcoded separately by different engines. After that the MP4Box tool multiplexes video and audio and generates a new file. This process is illustrated in Figure 6.

[1] MP4Box is a multimedia packager open source tool, that allows manipulation on multimedia files. It provides a considerable number of functionalities: conversion, splitting, hinting, dumping and others. Available at: http://gpac.wp.mines-telecom.fr/mp4box/mp4box-documentation/

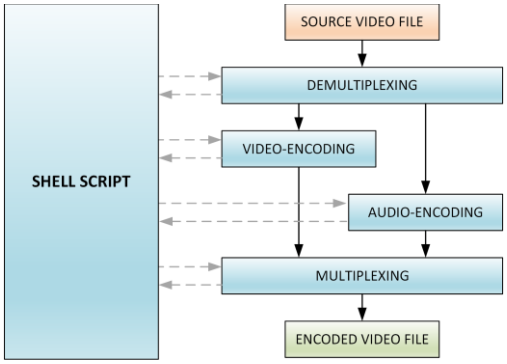

Figure 6 - The actual transcoding-process

3.2.4.4 **The web-interface**

The web-interface for the transcoding-server, depicted in Figure 7, uses asynchronous JavaScript and XML (AJAX) and a responsive web design. It allows encoding video files in 3 steps and to create or edit profiles. The screenshots provided in Figure 8 to Figure 10 show these three steps.

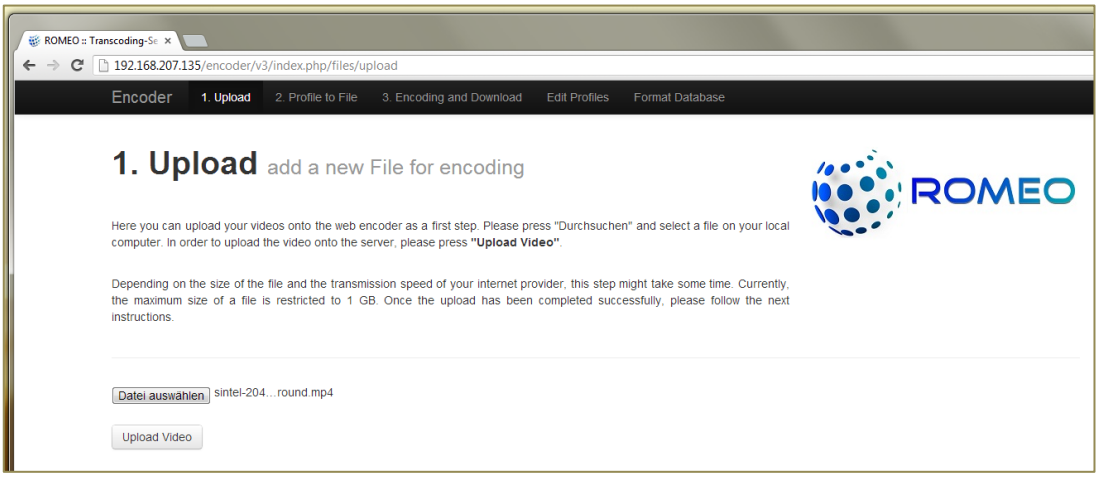

Figure 7 - Transcoding-server web interface

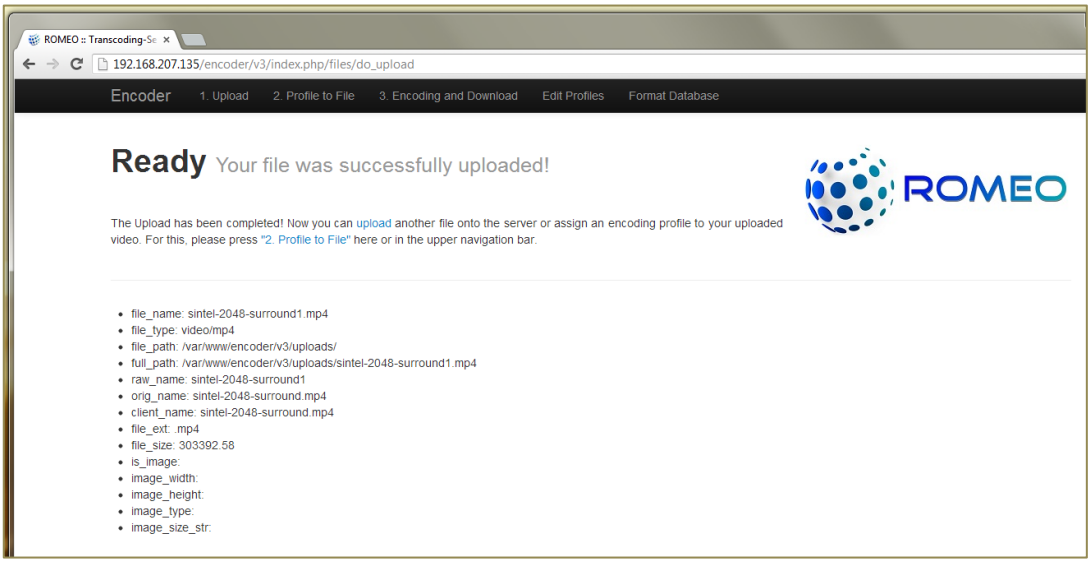

Figure 8 - Transcoding-server web interface: first step –file upload

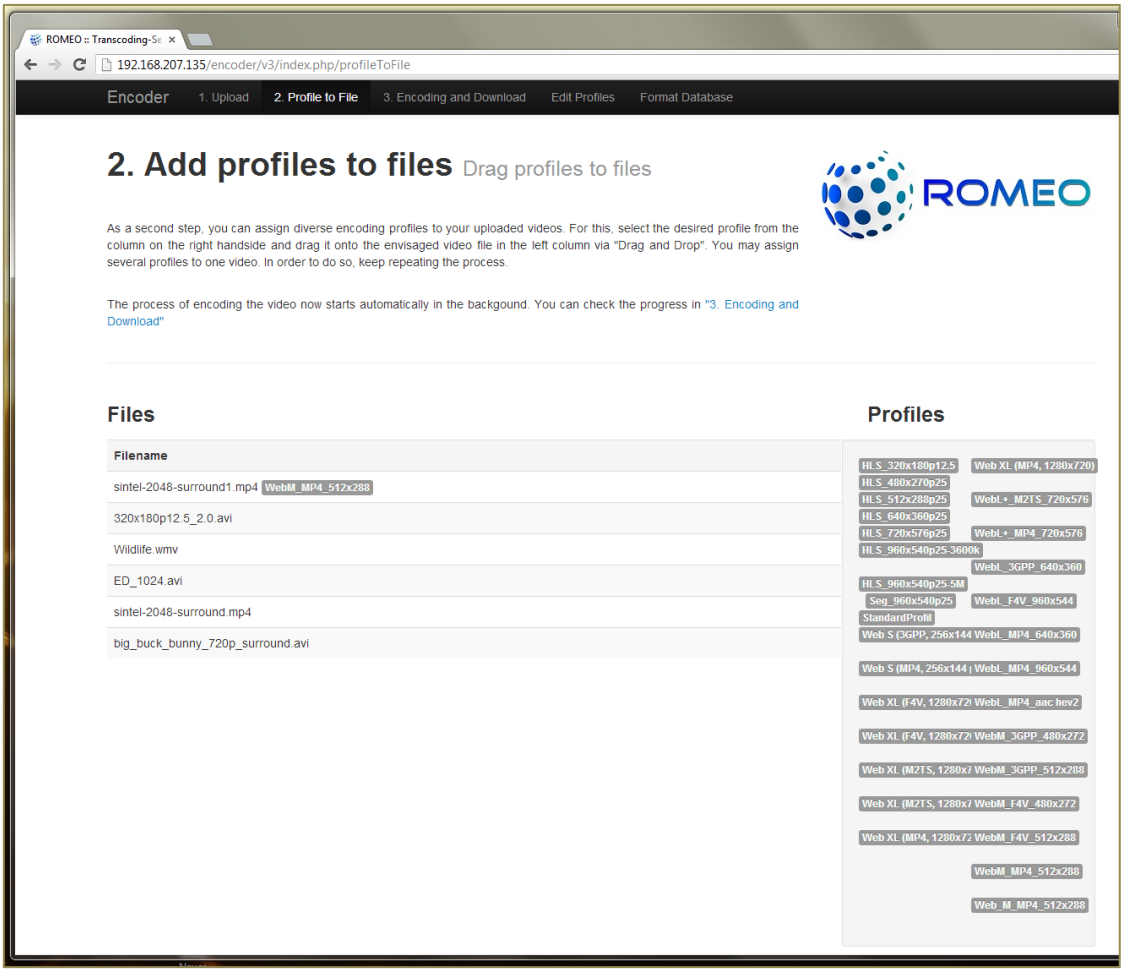

Figure 9 - Transcoding-server web interface: second step – add profiles to files by drag & drop

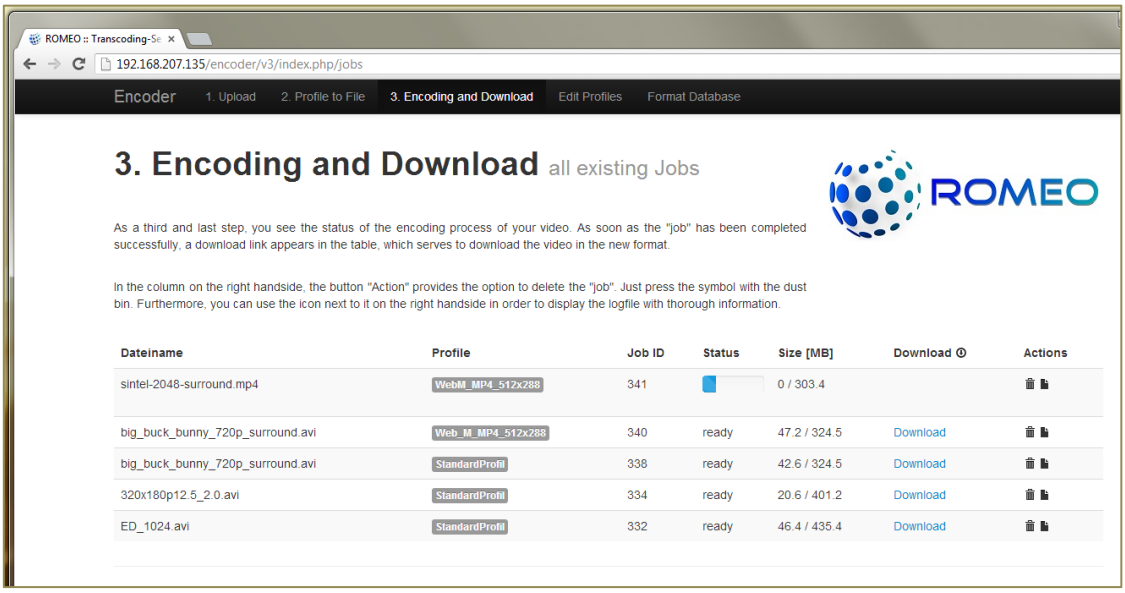

Figure 10 - Transcoding-server web interface: third step – encoding

3.2.4.5 **The encoding profiles**

The transcoding-server contains a format database based on the overview “Format CS” of the AV-standards (Figure 11), many formats are available for different play-out qualities. It is possible to generate various encoding formats (profiles) for the encoding-process through the web-interface (Figure 12) or to edit/modify a previously created profile (Figure 13).

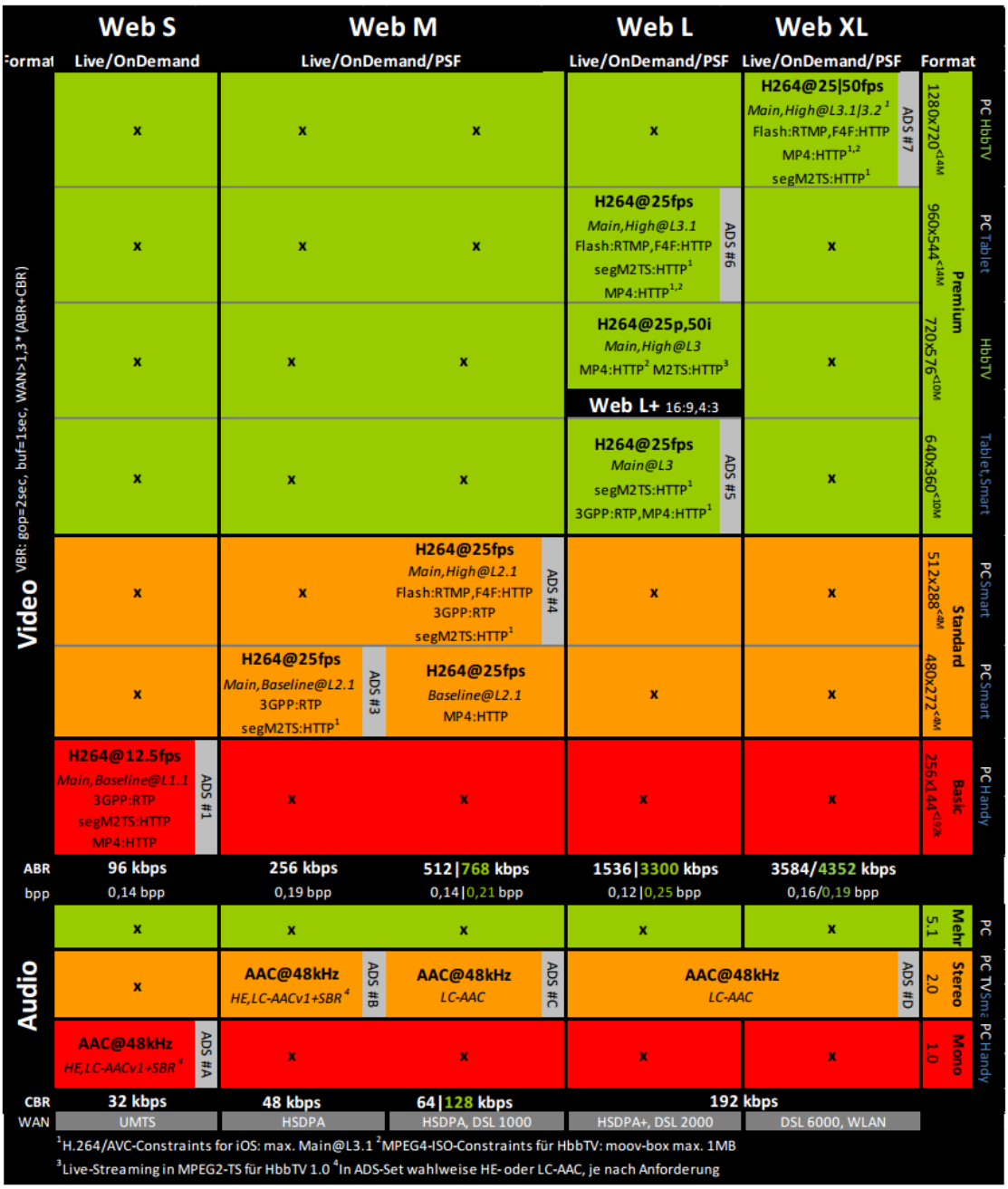

Figure 11 - The “Format CS” AV-standards

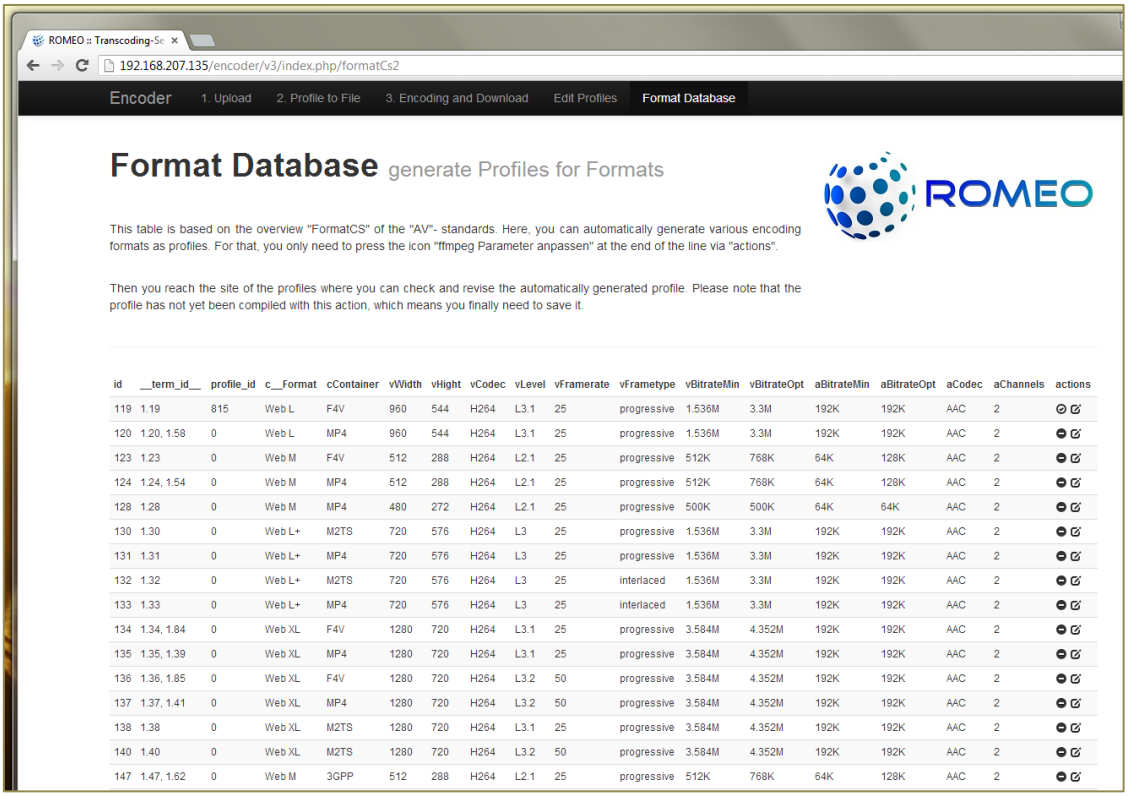

Figure 12 – Transcoding-server web-interface for the format database (based on the AV-standards)

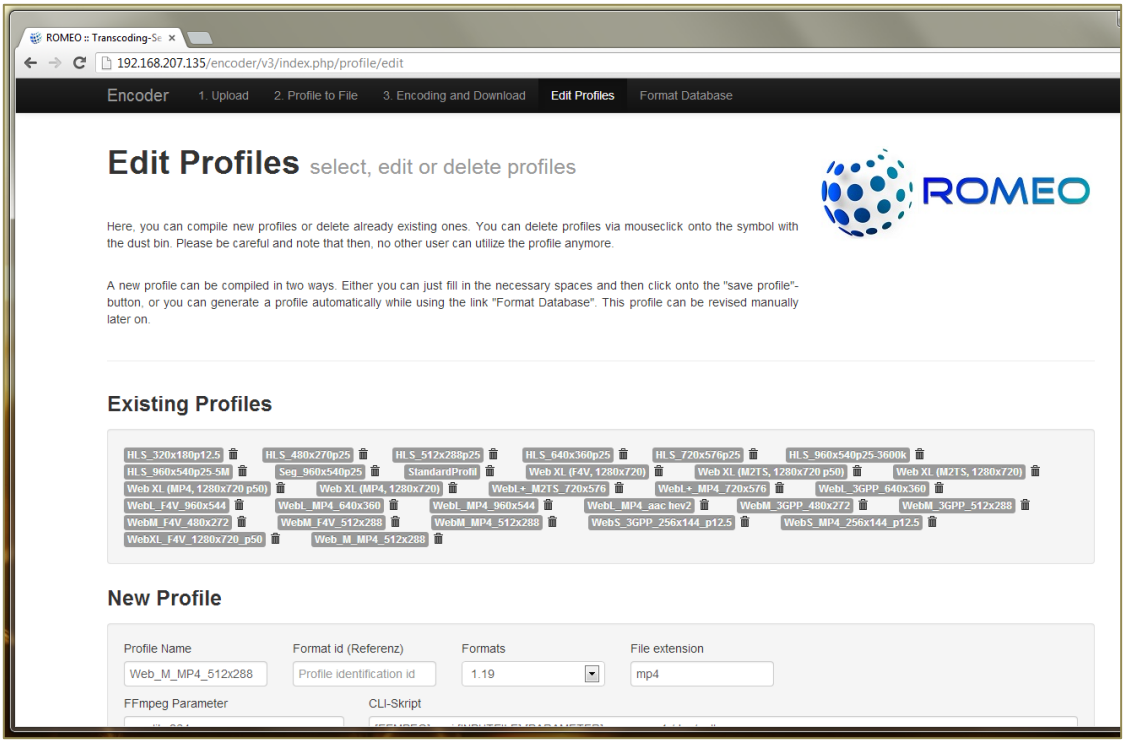

Figure 13 – Transcoding-server web-interface for editing/modifying encoding-profiles which can use the format-database

3.3 Content capture at mobile terminals

At the mobile devices the UGC is captured by the built in stereo camera. As depicted in Figure 14, the camera consists of two identical raw image sensors able to capture stereo frames. The sensor raw output is processed by the *Image Processing Pipeline* to YUV frames (YUV is a colour space typically used as part of a colour image pipeline. It encodes a colour image or video taking human perception into account, allowing reduced bandwidth for chrominance components, thereby typically enabling transmission errors or compression artifacts to be more efficiently masked by the human perception than using a "direct" RGB-representation). Depending on the shooting mode, the frames are encoded in H264/AVC format for video or JPEG format for still image capture. The content is then stored in the internal device memory. At the same time, frames are displayed on the auto-stereoscopic display. Image processing on the captured raw frames includes: Defect Pixel Correction, Lens Vignetting Correction, Lens Geometrical Distortion Correction, Chromatic Aberration Correction, Green Imbalance Correction, Raw Noise Filtering, Color Filter Array Interpolation, Color Correction, Gamma Correction, Luma and Chroma Noise Filtering, Temporal and Spatial Video Noise Filtering, Stereo Image and Frame Alignment, Video Stabilization.

The captured content also uses automatic control algorithms: Auto Exposure, Auto White Balance and Auto Focus, based on statistic data collected from both image sensors. The stereo camera is controlled by the GUI of the camera application.

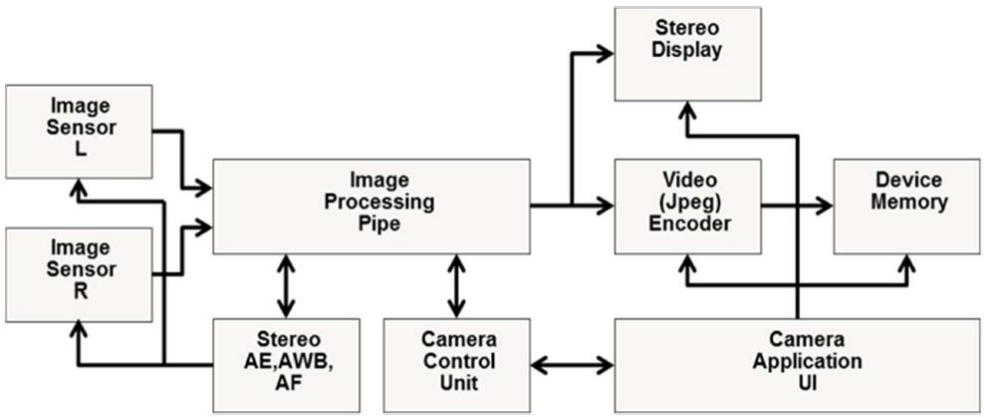

Figure 14 – Block diagram for the UGC capture at mobile devices

3.4 User Interface Capabilities

3.4.1 Fixed and portable terminals

The UGC graphical user interface (GUI) for the fixed and portable terminals coexists with other ROMEO components, DVB reception, Synchronization, Audio and Video Renderers and Audio and Video Overlay. Figure 15 depicts the UGC GUI tabs corresponding to the interfaces with these components.

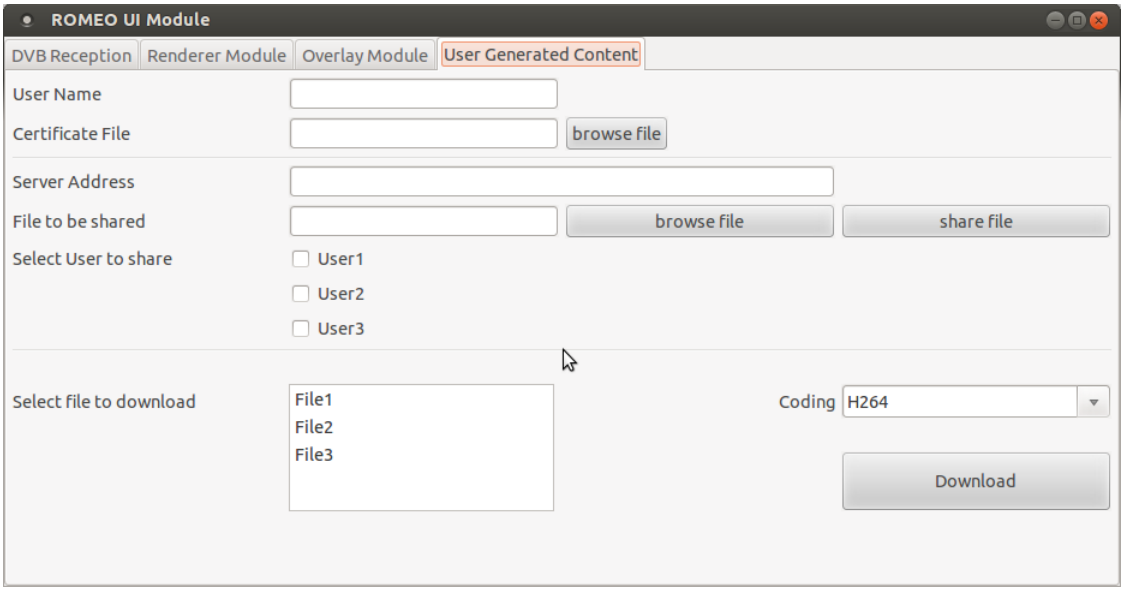

Figure 15 - UGC user interface (fixed and portable terminals)

The GUI uses a login mechanism for user authentication purposes. This mechanism includes username and password fields at the beginning of the HTTP payload, for each transaction with the server.

For the file upload procedure the GUI lists the users participating in the overlay communication, see ROMEO deliverable 5.3 [3]. A subset of the participants can be selected, together with a local file, to be uploaded to the UGC server. After uploading the file, all selected participants are able to download it.

For file download, the GUI constantly polls the server, using the list/search file operation described in section 3.2.3, for the files that are available to the user. The user can select which coding is requested, and then download the selected file from the server.

3.4.2 **Mobile Terminals**

The UGC GUI for the mobile terminal, depicted in Figure 16, exists as a standalone module developed in Qt[2] and WebKit[3] technologies. The GUI uses a login mechanism for user authentication purposes. This mechanism includes *username* and *password* fields at the HTTP payload, for each transaction with the server (Figure 16 a)). For the file upload procedure the GUI lists the users participating in the overlay communication. A subset of the participants can be selected, together with a local file, to be uploaded to the UGC server (Figure 16 b)). After uploading the file, all selected participants are able to download it. For file download, the user interface constantly polls the server for the files that are available to the user. The user can select which coding is requested, and then download the selected file from the server (Figure 16

[2] Qt is a a CSS & JavaScript like language, cross-platform application and UI framework for developers using C++ or QML.url: http://qt-project.org/

[3] WebKit is an open source web browser engine. url: http://www.webkit.org/

c)). In the “Make snapshot” tab, the camera’s preview is shown. After taking the picture, a snapshot preview is shown in the field *Snapshot* (Figure 16 d) and e)). The user can use fast upload of the image to the UGC server (with settings described in the “Share file” tab). After taking picture, it is added to the gallery in the “Share file” tab and it may be uploaded to the UGC server.

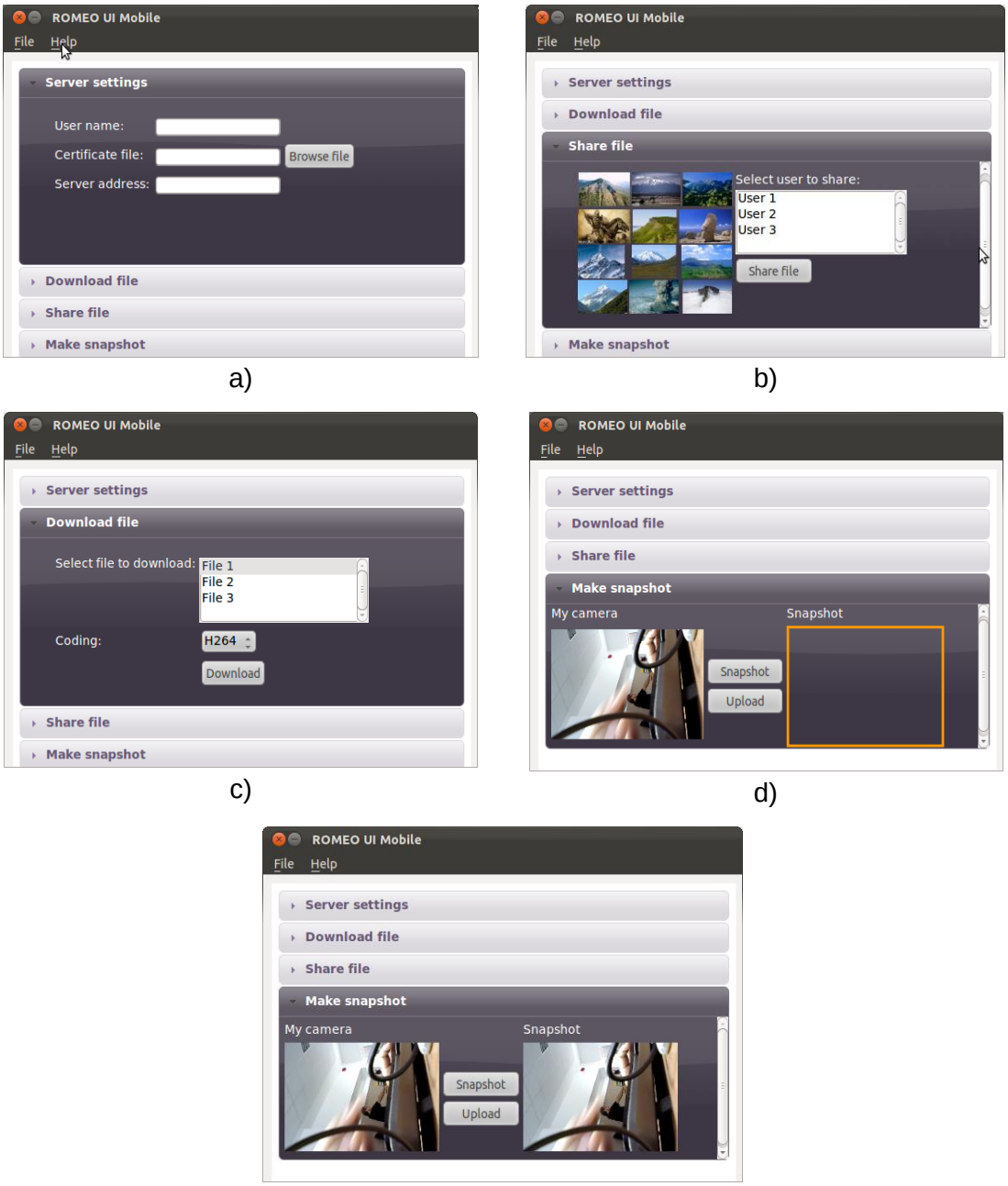

Figure 16 – The UGC GUI for mobile terminals

4 CONCLUSIONS

This deliverable describes the results of the research work developed on the ROMEO *User Generated Content* component. As described in section 3, to separate the user interface from the data storage, a client-server approach has been chosen. Peers (clients) run the software that allows ROMEO authenticated users to create, upload, share, search, download and delete content, whereas the servers, in a total of three, have each a specific function: the UGC server acts as the front-end server for the UGC component; the file server stores all UGC files and; the transcoding-server constantly watches the file server upload folder waiting for new files to encode. The UGC component has also specified two UGC user interfaces, one for fixed and portable devices and one for mobile devices, and a transcoding- server web interface that allows to create and edit encoding-profiles and also to start encoding-jobs.

For troubleshooting reasons, server modules have implemented return values that allow peers to debug errors they may encounter when performing UGC operations.

Additional work on the development of the UGC component is taking place in WP6, under the scope of *Task 6.8: Content Registration and P2P Security.*

5 REFERENCES

[1] ROMEO Deliverable 2.3, “Interim reference system architecture report”; 2012

[2] ROMEO Deliverable 2.2, “Definition of the initial reference end-to-end system architecture and the key system components”; 2011.

[3] ROMEO Deliverable 5.3, “Report on real-time Audio-Visual communication overlay for remote users”; 2013.

[4] Zend Framework http://framework.zend.com/about/

[5] Fielding, R. T. and Taylor, R. N. 2002. Principled design of the modern Web architecture. ACM Trans. Internet Technol.2, 2 (May. 2002), 115-150. DOI= http://doi.acm.org/10.1145/514183.514185.

APPENDIX A: GLOSSARY OF ABBREVIATIONS

A	
A/V	Audio-Visual
API	Application Programming Interface
ARC	Arcelik A.S.
G	
GUI	Graphical User Interface
I	
IT	Instituto de Telecomunicações
IRT	Institut fuer Rundfunktechnik Gmbh
J	
JSON	Javascript Object Notation
M	
MMS	MM Solutions AD
MVC	Model-View-Controller
O	
OS	Operating System
Q	
QoE	Quality of Experience
R	
REST	REpresentational State Transfer
T	
TTA	Turk Telekomunikasyon AS
U	
URI	Uniform Resource Identifier