D6.6	
Version	1.0
Author	ZED
Dissemination	PU
Date	03/02/2017
Status	FINAL

D6.6: NUBOMEDIA Social Game Demonstrator

Project acronym:	NUBOMEDIA
Project title:	NUBOMEDIA: an elastic Platform as a Service (PaaS) cloud for interactive social multimedia
Project duration:	2014-02-01 to 2017-01-31
Project type:	STREP
Project reference:	610576
Project web page:	http://www.nubomedia.eu
Work package	WP6: Demonstration
WP leader	ZED
Deliverable nature:	Report
Lead editor:	Teo Redondo / Paula Collazos (ZED)
Planned delivery date	11/2016
Actual delivery date	02/2017
Keywords	NUBOMEDIA, Demonstrator

The research leading to these results has been funded by the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement nº 610576

FP7 ICT-2013.1.6. Connected and Social Media

Contributors:

Teo Redondo (ZED)

Internal Reviewer(s):

Luis López Fernández (URJC)

Version History

Version	Date	Authors	Sections Affected
0.1	14/10/2016	Teo Redondo / Paula Collazos	All (Structure of the Content)
0.2	21/11/2016	Teo Redondo / Paula Collazos	Fill document with demonstrator information
1.0	03/02/2017	Teo Redondo / Paula Collazos	Final version

Table of contents

I. Introduction

Once Upon a Time (*OUaT*) is an interactive storytelling app that uses real-time multimedia communication and augmented reality technologies to allow the users to create and star in their own interactive tale.

Users will have at their disposal a series of costumes in order to dress up like the characters in the fable through predefined sets of virtual accessories. Thanks to the VCA system (Video Content Analysis: face recognition and tracking), the accessories that will be overlapping the reader's image will move with him, matching his face. Additionally, by using AR (Augmented Reality) markers, users will be able to "handle" different objects (flags, swords, pets…), also related to the story. To a large extent, this product can be considered more as a "toy" than a game. Thanks to this, it has a bigger potential as a product since it allows a free use dimension, beyond the functional rules and conditions of a game.

We have developed a demonstrator for mobile devices that makes use of NUBOMEDIA functionalities. In our case this includes, specifically, augmented reality features by overlaying 2D objects on a detected face: hats, mustaches, glasses... as by markers.

The demonstrator consists of two pieces of software, a client part that will run on iOS and Android platforms and a server part that will run as a client service within the NUBOMEDIA platform. The client software provides the basic features and contents of the application: simultaneous multiuser reading of children's tales (including pictures, texts, and user interface) and dressing up by overlaying 2d elements on top of each user's video feed. Besides that, the client has multilingual support in the user interface (Spanish and English) and in the stories (Spanish, English and German).

We have also created a web microsite and marketing material for the product to be communicated to the final users and so that it may be uploaded along with the binary to Google Play (Android) and Apple Store (iOS) stores.

The development of the demonstrator has been interrupted due to the insolvency of the company ZED Worldwide and the reduction of the staff available to finalize the project.

II. Functional description

We have developed an interactive storytelling app that allows the user to tell a tale of which he will be the hero together with his listeners, with the following main features:

2.1. Read story

"Reading" a story consists of going through the pages of the story, which can only be done by the Narrator. Each page corresponds to each one of the text fragments that are shown at once on screen (the same illustration can be maintained for several pages).
To move page press the button 'Next page' or 'Previous page'.
Each illustration (and therefore its transition) is associated with one or more pages.

2.2. Wear costumes

All users can disguise themselves using the assets 2D catalog, organized by sets, corresponding to the story.
When you choose an accessory, it is automatically placed on your video image in the corresponding place on your face, and is thus placed even if the user moves.

<u>DISGUISE: COSTUMES MENU / ACCESSORIES MENU</u>
In the HUD of the current session screen there are two buttons to disguise:

- **Costumes menu**. It gives access to the menu that contains each costume (set of accessories, the own ones of a single character generally) available in the story. When you select a costume, the user "puts on" all of its accessories at once.
- **Accessories menu**. In this case, the menu makes available all the accessories of the story, without being grouped by costumes. The user gets each accessory, one by one.

Each costume and accessory can be freely used through these menus. All users participating in a session can use them, even if they do not have the role of narrator.

<u>REMOVE A COSTUME</u>
When the user has an active costume (at least one accessory on his image), a ‘Remove accessories’ button emerges under his video window (a red cross on the costume icon).
When you press it, all the accessories of your image are automatically removed.
This button only appears under the video window that shows the own user’s stream.

<u>SUGGESTIONS</u>
By default (associated with a particular page), the HUD notifies the **user Narrator** with a visual feedback at the ideal moment, in narrative terms, to put on a costume. This

feedback consists of a special brightness or contour in the 'Costumes' button, that once opened this menu will be replicated on the corresponding costume.

2.3. Show AR objects

During a session, with the camera active, there is the technology to read AR codes and thereby represent virtual objects in the video stream.
These virtual objects are, like the costumes, thematically associated with the narrative, although their use is not mediated by an interface, but users use them freely from sheets of paper with the codes printed on them (in the real world): they expose them to the camera and the object is displayed along with the video signal. They are, therefore, accessible to any user at any time (while there is an active session).

2.4. Manage video windows

The video windows are the containers of each of the video signals that participate in the session, one for each user. There is a video window for each participant, including the user.
Taking into account that the limit number of participants per session is 4 that will be the maximum number of video windows present on the screen.
The default layout of the video windows is **minimized**, forming a column to the left of the screen and leaving the bulk of it for illustration.

FEATURES

- The **Hide / Show Video Windows button** lets you hide or display the minimized video windows of the viewers, to make the background (illustration) completely clear. If this button is pressed when there is a maximized video window, it is also hidden; when the video windows are deployed again, they will all be minimized.
- A video window can be **maximized** by tapping on it. When this happens:
  - It moves from the left column to the center of the screen (larger size).
  - The 'Photo' and 'Remove Costume' buttons (if any) appear under the video window, and the 'Minimize' button on the upper right.
- A maximized video window can be **minimized again** by tapping the 'Minimize' button.
- [Only if you are Creator of the session]: In case all the participants slots (3 + user) are not occupied, the **'Invite friends' button**, which leads to the friends selection menu, is shown in the last unused slot.

2.5. Photograph

It is possible to take screenshots of the open video window as the main one on the active session screen. To do this, it is only necessary to press the **'Photo' button** under that video window and the resulting image will be stored in the 'Photo Gallery' section.

When a user creates a session he adopts the role of **CREATOR**, and has control over who participates in it. There are two possible ways to conduct a session:

TALE MODE

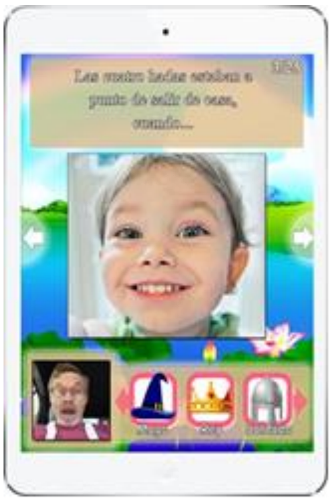
It is the main mode of *Once Upon a Time*. In it, a participant in the role of **NARRATOR** tells a story (directs the reading) to the rest of connected participants (**SPECTATORS**). All functionalities are available.

<u>Starting a new version</u>:

A user creates a new session and a unique code is generated. This code will persist or not in time (for future sessions). When another user introduces this code, he gets access to the story.

<u>Telling a story</u>:

1. The user accesses the 'ReadingScreen'. This screen provides the user with the story's reference text, a video window for each spectator connected to his narration and an interface that will allow him to choose virtual costumes. The user will also be able to see its own image in live video stream.
2. For their part, the connected spectators will access the 'Spectator Screen' that contains a main video window showing the narrator and an interface to choose virtual costumes. As in the previous case, they are also able to see a secondary video window showing their own image in live video stream.
3. While reading, the narrator is able to turn pages at any time by doing tap on the screen.
4. In certain moments during the story, the interface suggests costumes to go along with the story. At this point, the narrator as well as the spectators can choose from a set of available accessories. The most suitable ones for that specific moment will be suggested to each participant. Once a set is selected, it will be placed automatically on top of the user's image as determined by the facial recognition system based on three key parameters: eyes position, nose position, mouth position. The narrator will also suggest objects, but they will require the use of previously printed AR markers, instead of selecting an interface option.
5. The story moves forward as the narrator turns the pages.
6. The session ends when the narrator closes it. Any spectator can leave at any time.


Las cuatro hadas estaban a punto de salir de casa, cuando...

Once Upon a Time, tale mode

FREE MODE

A multimedia video session is established between two or more users without any reference text. All of them are free to use the available assets in order to “dress” their own image during the session.

<u>Starting a new version</u>:

A user creates a new session and invites other users from his or her ‘Friends’ list to join. As they accept this invitation, they will be added to the new session.

<u>Free mode session</u>:

This mode simply consists in setting up a multimedia video session between two or more connected users. During this session, there will be audio and video transmission and the participants will able to disguise freely using the virtual accessories and objects, being visible to the others.

In this case, the virtual accessories are available individually (not by costume sets) and will be placed one at a time. Virtual objects will work as in ‘Tale’ mode: participants will use at will the AR markers that they previously printed.

The session will end when only one user remains.

III. Development description

OUaT is a collaborative project where technologies developed by different parties are converging. The integration of each technological part to generate a consistent and demonstrative product will require coordination with the rest of partners in order to implement technologies with different origin.

The mentioned technologies converging in this product are specifically:

- **VCA (Video Content Analysis)**
  - Face detection and tracking.
  - Nose, mouth and eyes detection.
- **AR (Augmented Reality)**
  - Overlay 2D sprites on detected positions of the video stream.
- **Communication capabilities**
  - Bi-directional video/audio communication + real-time video communications N to N video feed.

The AR 2D Assets developed are costumes and items:

- Assets are predefined sets of virtual accessories: the costumes.
- According to the different tales and their related situations.
- AR items represent the most iconic elements in each tale, such as an apple and a potful.
- They are used through coded templates displayed to the device video camera.

We have developed a fully operative app for iOS tablets. The following describes the developments made:

3.1. HTML5 + Javascript + Apache Cordova

The initial development of the client focused on a single source code that would be later deployed on each of the target platforms (Android and iOS). Taking advantage of our experience and knowledge, we created a series of prototypes using HTML5, CSS3 and Javascript under Apache Cordova through the generation of hybrid applications (as they make use of the device hardware such as the camera).

Several application packaging tests were performed, either using only Cordova or Cordova and CrossWalk (XWalk). The latter embeds a chromium engine in the

packaging making the application increase by about 25Mb and allowing greater portability to other devices.
The tests performed were not encouraging, as they only work on two Android devices and not on iOS. These tests were even performed using the browsers of the devices (web access), getting the best results from devices that had a Firefox browser installed (almost 100%), whereas on those using Chrome only 20% worked (see Excel file attached to the document).

3.2. Native iOS development (iPad devices)

Given the low viability of NUBOMEDIA technology in its HTML + Javascript version in mobile environments, different options of native development technologies are evaluated given the little previous experience in this type of development (once the use of multiplatform engines with which we are more familiar Cocos 2Dx and Unity 3D is discarded by the potential complexity that will add both in the development of the application and in the use of NUBOMEDIA libraries).
We finally begin with the development for iOS using Swift language under XCode (with the corresponding learning curve) since the available libraries for NUBOMEDIA are at that time only available for that platform.
It is decided to only consider as a demonstrator objective devices with large screen, since the functional characteristics of the product require sufficient resolution and size to be able to display the different elements: images and texts of the story and multiple connection windows in the room. This is why the development of a specific version for iPad is initiated and it is only for these terminals for the ones where the interfaces distribution and design is solved.
The development begins by materializing the functional basis of the application, which is the reader of stories (illustrations and text contents) with multilingual support (both in the final contents as also throughout the navigation menus and user interfaces), user interface, navigation and graphic effects such as animations and transitions of interface elements. Tests are performed on devices of high (HD) and low (SD) resolution.
Subsequently, data persistence is provided for the application, thus allowing the registration of users and the definition and storage of friends' lists with a functional solution that reinforces explicit consent of the users involved in every friend-relation (since a potential target of the application are children, these considerations and features are key for potential real life deployment). This is done using a separate development that maintains user data structures, friends, friendship requests, as well as definitions of overlay elements definitions and QR objects for the purpose of augmented reality features. Communication with the persistence backend is developed via Rest-API.
Finally, the libraries of NUBOMEDIA made by Telecom Italia are added. After solving a series of problems with their integration in the Swift base code and solving several issues an error-free compilation is reached and the implementation of the NUBOMEDIA hosted service is initiated in parallel with the construction of the service itself.
In the last phase of the development we decided to try to make the application visually adapt to smaller devices (iPhone) and we therefore started the development for the scaling and distribution of interfaces and graphic elements correctly.

3.3. Native Android development

Development begins in Java for Android devices. A "splash screen" and a connection with the service are finally obtained, and thus showing the communication with users of the room in is solved after many issues (although in a very precarious way).
Given the relative low knowledge of native development under Android and the issues experimented we explore other development alternatives in order to expand the range of devices in which the application is visible. We finally opted for Qt5 / QML and we begin to develop a prototype product using that technology as it allows a wider multiplatform reach (Windows, Linux, OSX, Android and iOS).
We decide to start with the implementation of the product core: story reader with multilingual support, user interface, navigation and graphic effects type transitions and animations of interface elements. Our intention being the validation of the feasibility of using Qt5 as the basic development technology for the prototype before introducing the integration of NUBOMEDIA libraries.
The final result of these efforts is a functional story-reading application that is interrupted before performing the final integration of NUBOMEDIA libraries and functionalities.

3.4. Service development

The platform service documentation described that it would be possible to host applications made in JavaScript (NodeJS) and Java. JavaScript was to be our first option given our knowledge of it and we even perform a few tests with it. However, in the final expression of this functionality, only Java was finally supported by NUBOMEDIA platform and we therefore had to adapt to it.
To facilitate the development of the service, we had to use Spring-Boot as a technology within the Java Spring framework that allows the execution of web applications.
The project is developed under Eclipse and Maven is included in the technology mix to enable the software package management needed for NUBOMEDIA's core and filters dependencies and for later packing the final service application in a jar file.
For the development of the service, a Kurento Media Server (KMS) had to be installed locally on a Linux machine (Ubuntu 14.04) in order to have a reasonable stable test and development environment.
The service communicates with both the KMS and a backend service that allows data persistence.
According to the documentation the service must be deployed in the NUBOMEDIA PaaS through a Git repository that can be public or private, adding three files: one of certificates jks, another the generated jar and finally the Dockerfile in which it is specified the image to use, where to locate the key file and the execution of the jar.
The final result of our efforts in this activity was a fully operational service, correctly deployed in the NUBOMEDIA PaaS following all development requirements and procedures. The service was tested in many different settings and debugged as needed.

3.5. Testing and debugging

NUBOMEDIA (FaceOverlayFilter) filters are tested, noting that the image is not correctly overlaid in different devices (lower resolution devices worked correctly but HD resolution devices presented overlay aberrations -see document Annex II-). We are

told that we must use the filters provided by Visual Tools that are going to be the official ones instead of the one contributed by NUBOMEDIA that is a demo.
After installing the filters provided by Visual Tools in the local KMS, tests are performed with several filters (NuboXXXXDetector). It is observed that it is not possible to overlay any image, but instead it is only possible to perform detection and tracking of the components of the face.
We investigate how to solve this problem by doing many other tests and modifications and find a viable approach by linking the filter provided by Visual Tools (NuboFaceDetector) with the filter provided by NUBOMEDIA (FaceOverlayFilter) and performing calculations for the positioning of the different elements to be overlayed based on the area detected by the NuboFaceDetector filter.
A series of functional tests and video captures are made in final devices to demonstrate both the progress made and issues met.
A series of documents is made for the request of the new filter to VTT (see document Annex III). Coinciding with the description of the above requirements, Visual Tools exit of the consortium is communicated to us, and the filter is finally developed by members of the URJC I team. When the first version of the filter is finally received, it is integrated in the demonstrator and a new set of tests are performed showing new issues (the KMS failed after connecting a second user to the same room in which there was a previous user with an AR disguise on). These new problems where reported through the development list and our service source code is then uploaded to the Github repository for examination by the URJC I team.

We received a series of patches to be installed on the local KMS server, which solved the problem of connectivity and the use of filters by several users in the same room.
After performing several new tests we verified that the functionality had improved substantially, but now different problems showed up in new testing situations. We asked information to better understand if several filters could be linked to add several overlaying elements in a video feed (to show, for example, glasses, a hat and a mustache superimposed on the same face). The final proposed solution is to avoid this type of functionality, since memory consumption will be excessive.

As a deliverable of the final result of the demonstrator, we created a new video showing most of the demonstrator functionality and a high level QA report (see Annex IV).

3.6. Content, design and development

A complete process of design of the application is carried out starting on the pre-production stage of the project.
First steps on design involve the development of several concept documents which define the basic structure and foundations the product will have. After being discussed and tuned up, the main idea is finally shaped, resulting on a main premise upon which the whole product will be built: “an interactive storytelling app with real-time multimedia communications that will allow users (eminently parents and their kids) to *gather* and participate around a tale being told, even if they are physically distant”.
Once this is set, the functional design document is started, as well as the tales are selected so the Art team can begin the conceptualization process.
Design and Art work in parallel to define and compose the tales’ literature, their visual style, the usability of the user’s interface and the whole structure the app will have.

At a given point, development can start to prototype real functionality for those contents and designs and, as it takes form, Design and Art can iterate the design and make low level adjustments, until the app come to its final design.

Art design

Art design follows a process divided in several stages.
First of all, after the tales are chosen, the artist experiments with different artistic styles and techniques, looking for one which allows a fair balance between agile production and aesthetical appeal. Watercolor on paper subsequently scanned and digitally edited turns out to be the best option
In parallel, Art works with Design on the tales' scenes breakdown, extracting the key narrative moments that will be needed to effectively tell the story.
After these two stages are completed, Art starts creating the illustrations for each tale. This stage involves a first sketch made with watercolors on paper. Art and Design keep coordinately working on the final result of the scenes, looking after the narration purposes.
When an illustration / scene gets to its final paper version, it is scanned and passes to the final digital treatment stage on Photoshop. This has to be done for each illustration / visual scene. Digital pieces are compiled and the image binary is finally exported on JPG format.
2D assets, defined by Design, are generated directly on a digital format, following the instructions regarding properties and formats given by the Programming team.

Design of the interface of the application

After being defined on the functional document, the Design team elaborates a flow diagram that previews the whole navigation and screens set the application will include. A mock up for every screen is also made, so, in the rough, the application can be visualized before starting its real development. Although being a simple demonstrator focused only on central functionalities, its low level definition finally results on over 10 sections which branch out and include a variety of contextual options. Each mode ('Tale' and 'Free') require a different lobby and HUD, and different cases must be considered when managing the "join session" and "invite" functionalities.
The Art team then takes this first sketch and works on the final layouts and visual styles, adjusting elements on the basis of usability criteria and aesthetic appeal.
The design of the application interface is developed by the artists using Photoshop. Binary images are exported in different resolutions on JPG format.

Story design

The tales that will be included on the app are chosen from the most traditional and celebrated ones passed through decades on the Western culture. It is decided that three of them will be used on the demonstrator, being them:

- The Three Little Pigs
- Little Red Riding Hood
- Goldilocks and the Three Bears

Each one of them will be limited to a certain number of "narrative moments" or illustrations. Each narrative moment may include one, two or three text sections (which are called "pages") subject to a maximum number of characters. This way, one illustration can contain several short text pages.

After the tales and their structure and conditions are defined, the Design team writes an original version closely based on the most know versions of the chosen tales and breaks it down into the aforementioned “narrative moments”. These are sketched on a rough storyboard and shared and discussed with the Art team.
The next stage consists on the elaboration of a low level technical script of each tale which is divided into scenes and pages, and includes a detailed description of texts, elements needed on the illustration, 2D assets that will go with each page and every other detail needed to guide the definitive production of every scene.
While the Art team is elaborating the graphic materials, Design works with the Development team to consolidate the literary and functional elements that will be used to generate the build.
In their final form, the stories have the following structure:

- A description file (description.json), which indicates the identifier of the story, as well as the cover and the languages it manages.
- A file to relate the illustrations to the texts (story.json).
- A directory for images that will be divided into high and low resolution.
- A directory with languages. Three languages are presented per story: Spanish, English and German.

3.7. Marketing and Production release materials

Design of "marketing material"

Given that the Apple Store is our target channel for the distribution of the app, the Art and Design teams proceed to identify, define and create the required materials for publishing on Apple’s digital store. Therefore, these materials are generated:

- 1 x App icon (JPG image, 1024x1024).
- 5 x Screenshots for iPhone devices (JPG images, 2208x1242).
- 5x Screenshots for iPad devices (JPG images, 2732x2048).
- Application final title: “LiveTales”.
- Application description texts.
- Application keywords and categorization.

The described materials are elaborated and collected, ready to be uploaded upon publishing.

Product web design

A website, in the format of a microsite, is designed, developed and published, providing the final product, LiveTales, with a window to manage its corporate online presence and message.
The website includes is structured on different sections which describe every aspect of the product from a user centered perspective, and includes:

- Original graphic compositions.
- Product features description.
- Download link.
- Support link (“mail to” function).

The site’s URL is: www.playw.com/livetales

IV. Repositories of the content of the demonstrator

The repositories of all content generated for the demonstrator are housed in Github publicly.

Functional design of the demonstrator

Name	nubomedia-design
URL	https://github.com/pyromobile/nubomedia-design.git
Last commit	52147476654cd75b20b41d5d7f678248d33181ec

Demonstrator's client source code repository

Name	nubomedia-ouatclient-src
URL	https://github.com/pyromobile/nubomedia-ouatclient-src.git
Last commit	3e41f7d3fc16875709f9029801912d85998e0a7b

- ***Html_js prototype source code***

Name	html_js
URL	https://github.com/pyromobile/nubomedia-ouatclient-src/html_js
Description	HTML5 & JavaScript prototypes to be packaged using Apache Cordova.

- ***Native iOS (iPad) demonstrator client source code***

Name	ios/LiveTales
URL	https://github.com/pyromobile/nubomedia-ouatclient-src/ios/LiveTales
Description	Native swift2 source code (IPad version of the demonstrator).

- ***Native iOS (smartphones) prototype source code***

Name	ios/LiveTales-smartphones
URL	https://github.com/pyromobile/nubomedia-ouatclient-src/ios/LiveTales-smartphones
Description	Iphone adaptation branch of the previous Ipad source code (wip).

- ***Android demonstrator prototype source code***

Name	android
URL	https://github.com/pyromobile/nubomedia-ouatclient-src/android
Description	Native Android code for a basic prototype development using Nubomedia's Android libraries.

- *QT5 prototype source code*

Name	Qt5
URL	https://github.com/pyromobile/nubomedia-ouatclient-src/qt5
Description	Qt5/QML prototype source code (wip).

QA review results and demonstrator functionality videos

Name	nubomedia-ouat-qa
URL	https://github.com/pyromobile/nubomedia-ouat-qa.git
Last commit	280ba689fa9f97692d8f4c9b284a241ab522ef16

Art and assets repository

Name	nubomedia-contents
URL	https://github.com/pyromobile/nubomedia-contents.git
Last commit	e851ec4a19df840f4203f672364aa912dd37fc38

Note: The sources of the graphic files are not included because of their high weight.

- *Demonstrator Client Sources: Stories*

Name	tales
URL	https://github.com/pyromobile/nubomedia-contents/tales
Description	Child stories illustrations, and final stories packaging (art + synch texts .json files)

- *Client sources of the demonstrator: AR assets and items*

Name	assets
URL	https://github.com/pyromobile/nubomedia-contents/assets
Description	Graphic 2d assets for AR demonstrator functionalities.

- *Customer sources of the demonstrator: GUI*

Name	user-interface
URL	https://github.com/pyromobile/nubomedia-contents/user-interface
Description	Demonstrator's user interface graphic elements.

Demonstrator Service source code

Name	nubomedia-ouatservice-src.git
URL	https://github.com/pyromobile/nubomedia-ouatservice-src.git
Last commit	57e7d2c5f93e8b59d46ee881d9e690dbd04a5fc7

Demonstrator's service binaries ready for NUBOMEDIA PaaS deployment

Name	nubomedia-ouatservice-dev
URL	https://github.com/pyromobile/nubomedia-ouatservice-dev.git
Last commit	4cd556d3cb815ed0048538416e2cc463fc25b1b3

Name	nubomedia-ouatservice-pre
URL	https://github.com/pyromobile/nubomedia-ouatservice-pre.git
Last commit	dd405621eb1cac4f50ebf63fc1eaa0f210dd13b2

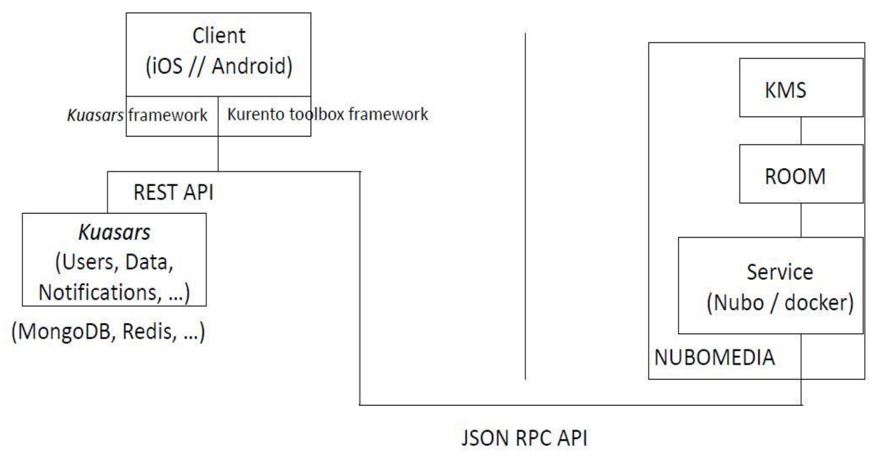
Name	nubomedia-ouatservice
URL	https://github.com/pyromobile/nubomedia-ouatservice.git
Last commit	c8bcd716dc080834d4f03135f48a5b4108ea43f7

Marketing and public communication materials

Name	nubomedia-marketing
URL	https://github.com/pyromobile/nubomedia-marketing.git
Last commit	3db7cb41e481508b19d706f3e27ae144f2624fec

Annex I - Architecture and technical installation of the demonstrator

The schema of the architecture implemented for our demonstrator is the following:



The demonstrator consists of two parts, a client part developed in native code for iOS platforms, and another part server (service) hosted in the PaaS of NUBOMEDIA that is responsible for performing certain operations between the client and the KMS.

In addition, the client and the service communicate with an external data persistence service called Kuasars.

- <u>CLIENT</u>

The packaging process is performed by the XCode development environment to create a binary file and uploaded to the Apple Store for validation and publication.

The installation process on a device is the usual of any application published in the Apple Store.

- <u>SERVICE</u>

For the packaging and deployment process, the steps described in the NUBOMEDIA documentation are followed (http://nubomedia.readthedocs.io/en/latest/paas/paas-introduction/).

Package preparation

1. The Java code is packaged with Maven, to generate a jar file with the following nomenclature in its name: **ouatnbmservice-x.x.x-SNAPSHOT.jar**
2. A Docker file is generated, in which you specify:
   - how the image is to be mounted
   - what directory structure will have and where the jar file will be located
   - which ports are exposed to other containers
   - which commands will be executed when the image is installed.

The content of the *Dockerfile* is like this:

```
FROM nubomedia/apps-baseimage:v1

MAINTAINER Nubomedia

RUN mkdir /tmp/ouatnbmservice

ADD ouatnbmservice-0.0.1-SNAPSHOT.jar /tmp/ouatnbmservice/
ADD desarrollo2.jks /

EXPOSE 443 8888

ENV DISPLAY=:0

ENTRYPOINT java -Denv=dev -jar /tmp/ouatnbmservice/ouatnbmservice-0.0.1-SNAPSHOT.jar
```

3. They will be uploaded to a git repository (in our case located in GitHub) the following files:
   - desarrollo2.jks
   - Dockerfile
   - ouatnbmservice-x.x.x-SNAPSHOT.jar

Deployment in the Nubomedia Paas
To perform the deployment on the Nubomedia PaaS, a user account is required on the platform.
Once on the platform, the steps to follow would be:

- Create an application.
- In the form that appears, we specify the name of the application, the number of instances that will be wanted, the type of instance and the data source of the repository.

Data persistence. Kuasars.
In turn, the demonstrator has persistent user data, which are stored on the Kuasars platform.
You must have access to the platform to create an application.
You will need to create the following entities:

Entity	Description

Users	Entity that contains the user's public information such as Nick.
Friends	Entity that contains friendly relations between users.
Requests	Entity that contains the requests for friendship. This entity is expirable.
Overprints	It contains the information of the objects to be overprinted and passed to the KMS by the service upon request of the client.
QRObjects	It contains the information of the augmented reality objects that are going to be shown through QR codes and that is passed to the KMS by the service upon request of the client.

Annex II – Anomaly report for Nubomedia FaceOverlayFilter

Problem detected.

We have observed that the filter used for painting overlaying images (***FaceOverlayFilter***) moves the reference frame of the face recognition when used on a HD device.

Devices used.

Model	iPad Air 1
S.O. version	9.3.1
Resolution	2048x1536 (retina)
UIView renderer	w:440 – h:587

Model	iPad Mini 1
S.O. version	9.3.3
Resolution	1024x768 (no retina)
UIView renderer	w:440 – h:587

Documentation.

According to the library documentation, (http://kurento-ios.readthedocs.io/en/latest/dev_guide.html) the default values are as follows:

```
/*
Default media constraints:

Audio codec: Opus audio codec (higher quality)
Audio bandiwidth limit: none
Video codec: Software (VP8)
Video renderer: OpenGLES 2.0
Video bandwidth limit: none
Video format: 640 x 480 @ 30fps
*/
```

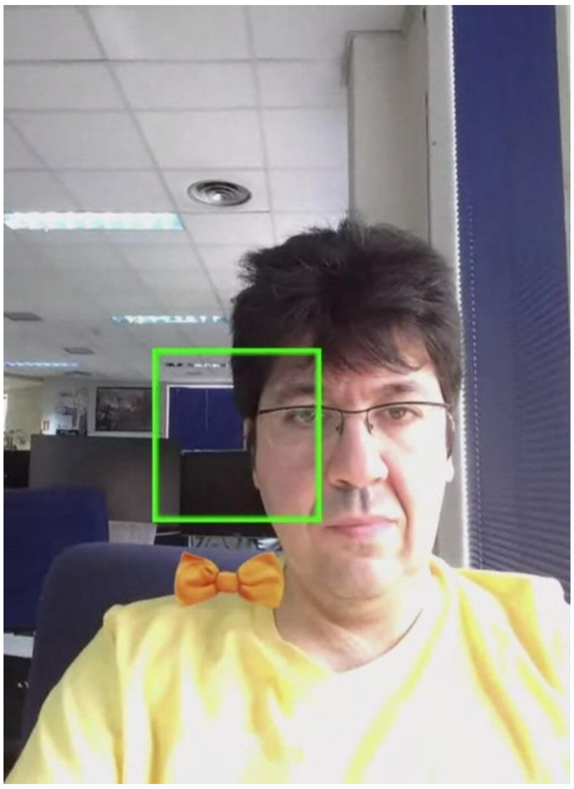
The “video format” is established according to the corresponding values (see UIView renderer for indicated values):

```
self.mediaConfig = NBMMediaConfiguration.defaultConfiguration()
self.mediaConfig!.cameraPosition = NBMCameraPosition.Front
self.mediaConfig!.rendererType = .OpenGLES
self.mediaConfig!.receiverVideoFormat.dimensions.height                  =
Int32(self.video1View?bounds.height)
self.mediaConfig!.receiverVideoFormat.dimensions.width                   =
Int32(self.video1View?bounds.width)
```

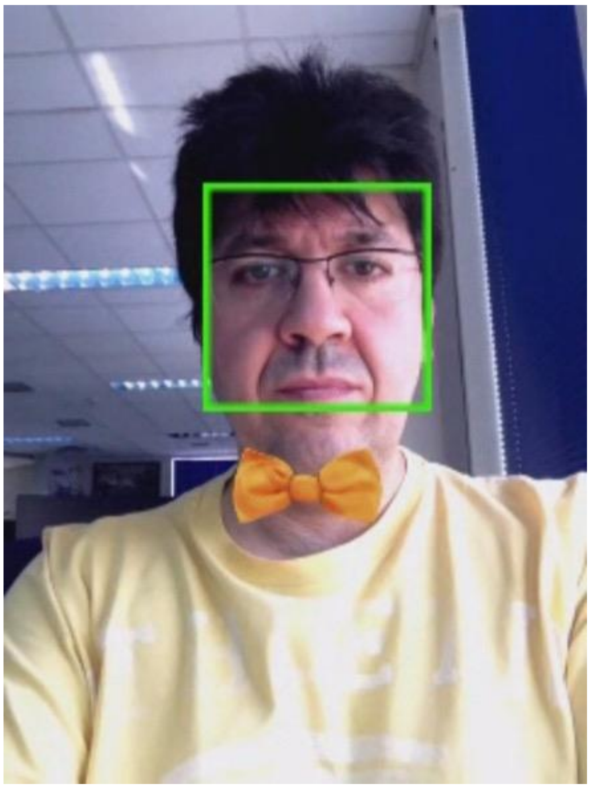
The changes made don’t seem to have any impact.

Simple mode.

Result for iPad Air 1 (HD):

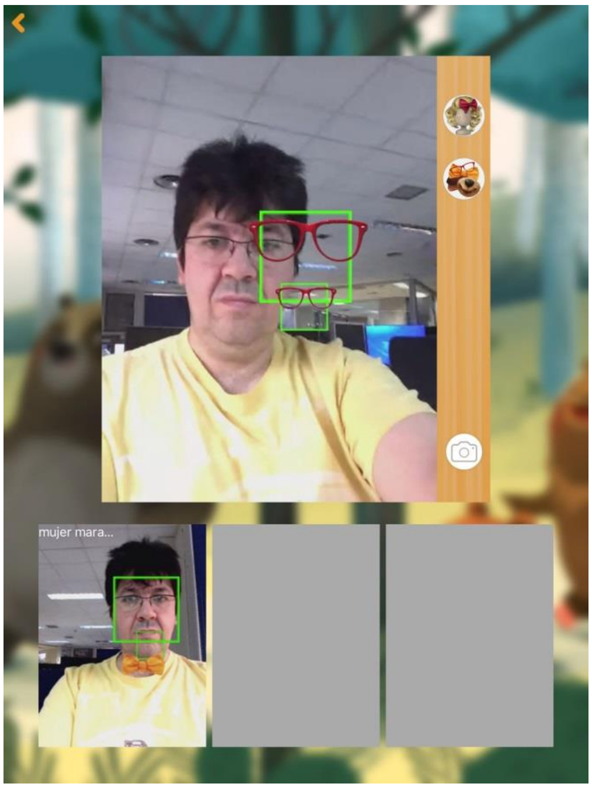


Result for iPad Mini 1 (SD):

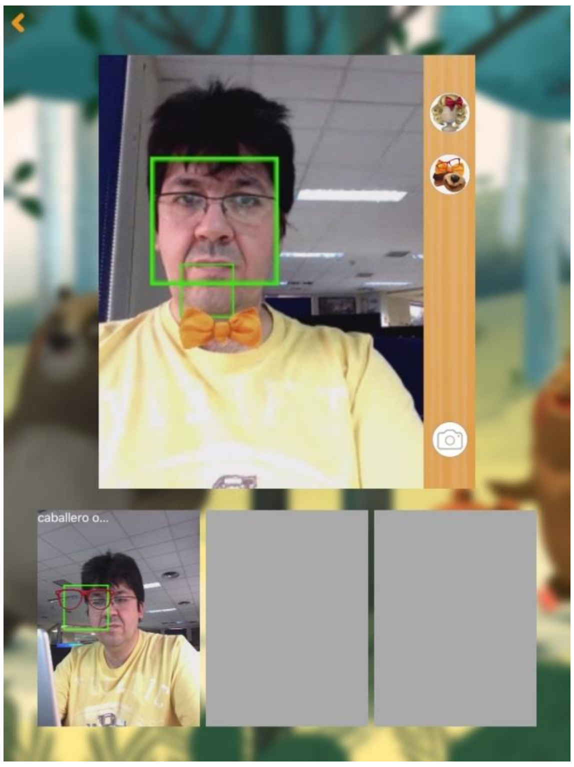


Conference mode.

Result for device iPad Air 1 (HD):



Result for iPad Mini 1 (SD):

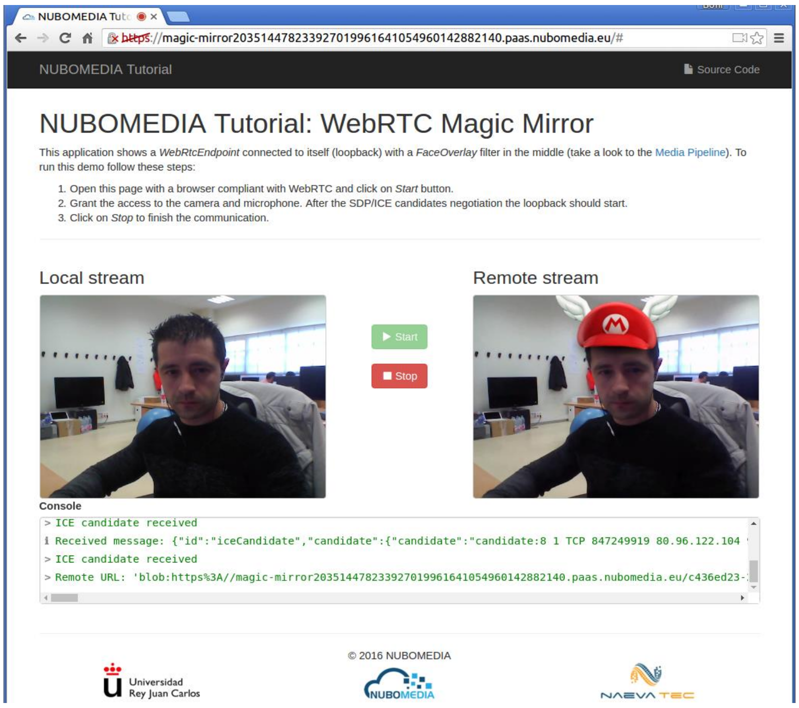


Annex III – Proposals for the development of a new filter that meets the requirements of the demonstrator:

First proposal.

Problem detected.

We are currently using the ***FaceOverlayFilter*** for overlaid images in the videostream provided by KMS.


NUBOMEDIA Tuto

~~https~~://magic-mirror20351447823392701996164105496014288214O.paas.nubomedia.eu/#

NUBOMEDIA Tutorial

Source Code

NUBOMEDIA Tutorial: WebRTC Magic Mirror

This application shows a *WebRtcEndpoint* connected to itself (loopback) with a *FaceOverlay* filter in the middle (take a look to the Media Pipeline). To run this demo follow these steps:

1. Open this page with a browser compliant with WebRTC and click on *Start* button.
2. Grant the access to the camera and microphone. After the SDP/ICE candidates negotiation the loopback should start.
3. Click on *Stop* to finish the communication.

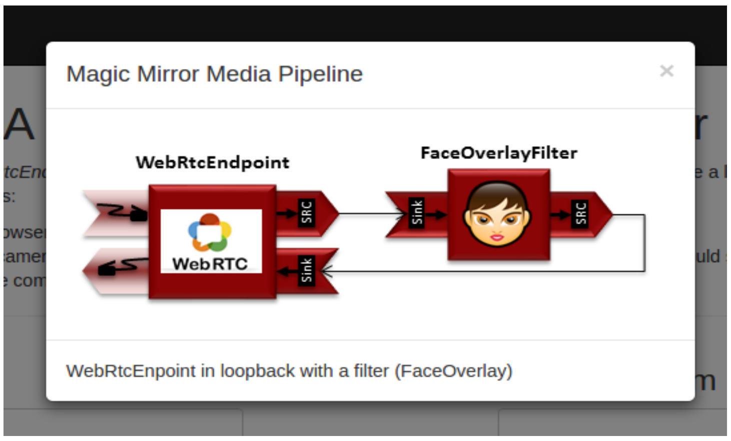
Local stream

Remote stream

Console

```
> ICE candidate received
ℹ Received message: {"id":"iceCandidate","candidate":{"candidate":"candidate:8 1 TCP 847249919 80.96.122.104
> ICE candidate received
> Remote URL: 'blob:https%3A//magic-mirror20351447823392701996164105496014288214O.paas.nubomedia.eu/c436ed23-
```

© 2016 NUBOMEDIA


Magic Mirror Media Pipeline
WebRtcEndpoint
FaceOverlayFilter
SRC
Sink
WebRTC
Sink
SRC
WebRtcEnpoint in loopback with a filter (FaceOverlay)

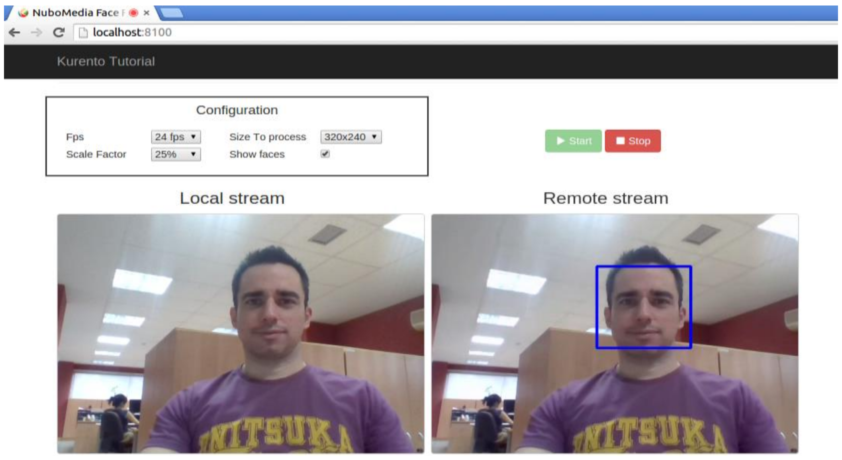
Uso:

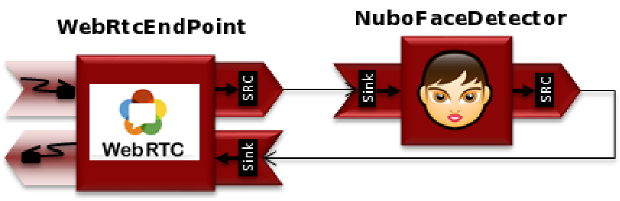
```
FaceOverlayFilter faceOverlayFilter = new FaceOverlayFilter.Builder(
roomManager.getPipeline(pid)).build();
faceOverlayFilter.setOverlayedImage(this.hatUrl, this.offsetXPercent,
this.offsetYPercent, this.widthPercent, this.heightPercent);
roomManager.addMediaElement( pid, faceOverlayFilter );
```

Proposal.

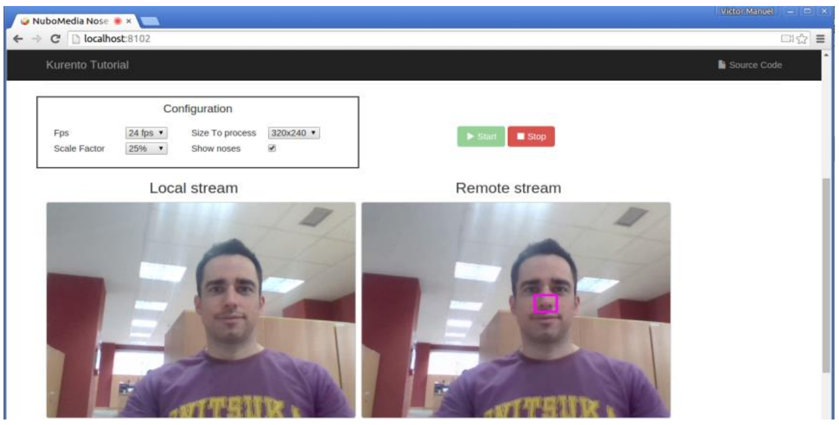
We would like to use the following filters created by Virtual Tools (http://nubomedia-vca.readthedocs.io/en/latest/index.html):

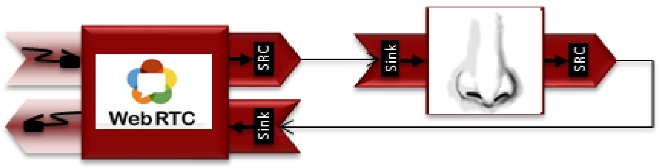
1. Face detector



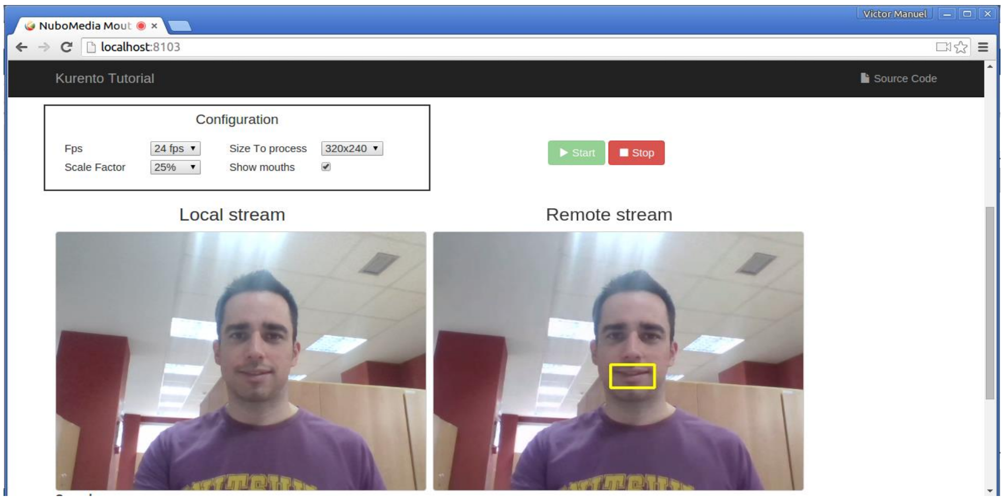


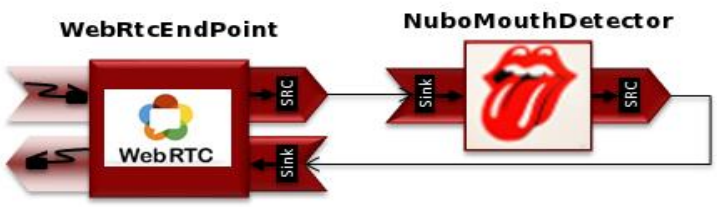
2. Nose detector



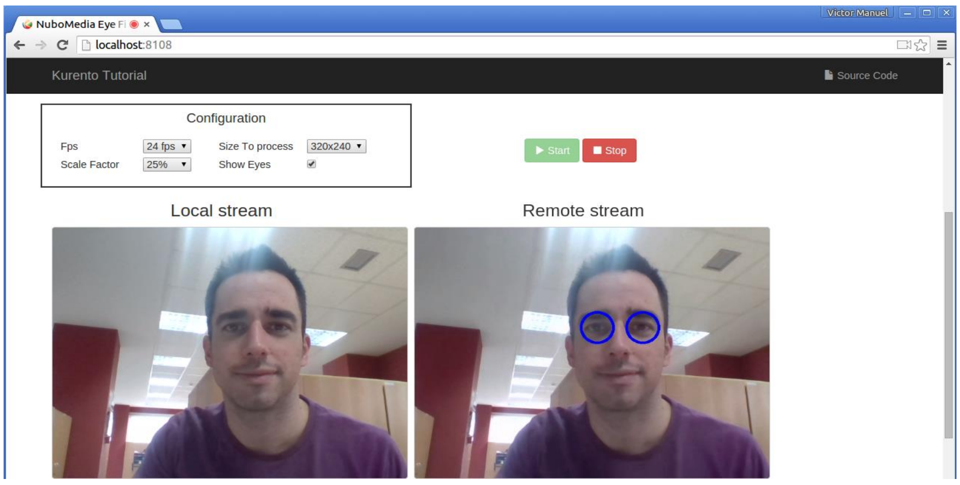


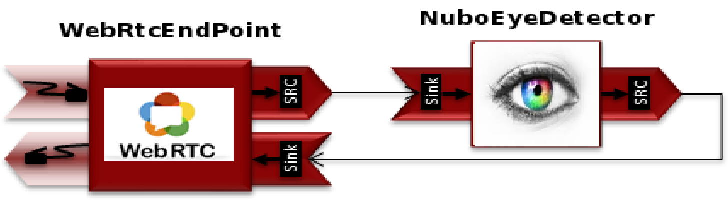
3. Mouth detector



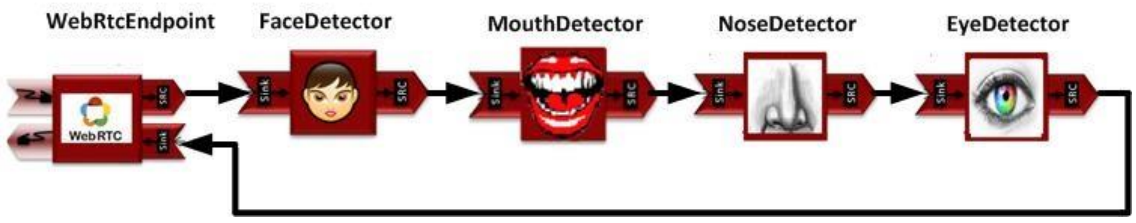


4. Eye detector

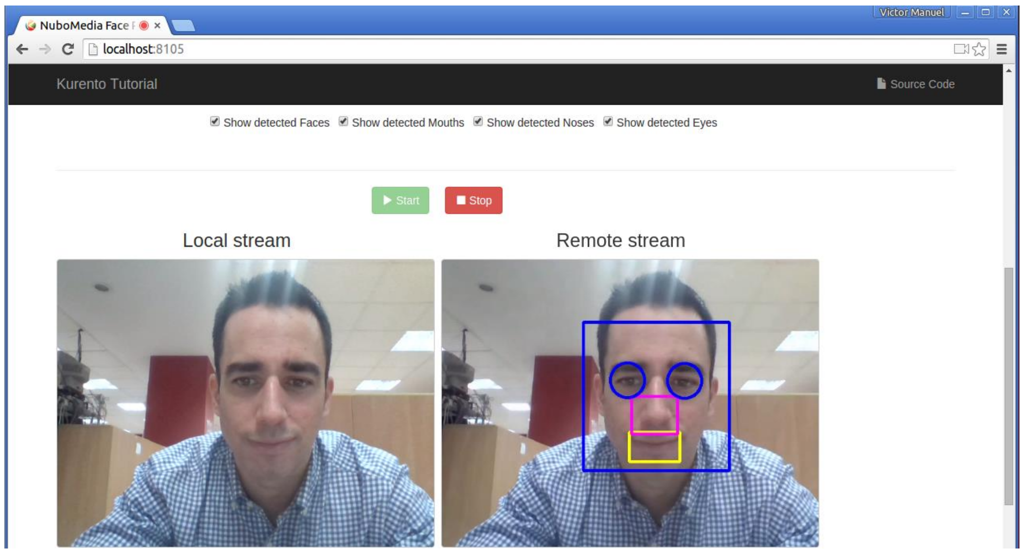




According to the documentation provided by Visual Tools, metadata can be transferred from one filter to another (http://nubomedia-vca.readthedocs.io/en/latest/face_profile.html) as you can see in the image below:



Result:



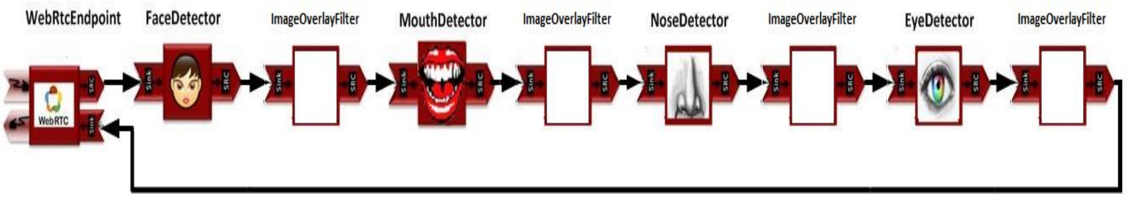
☑ Show detected Faces ☑ Show detected Mouths ☑ Show detected Noses ☑ Show detected Eyes

Local stream

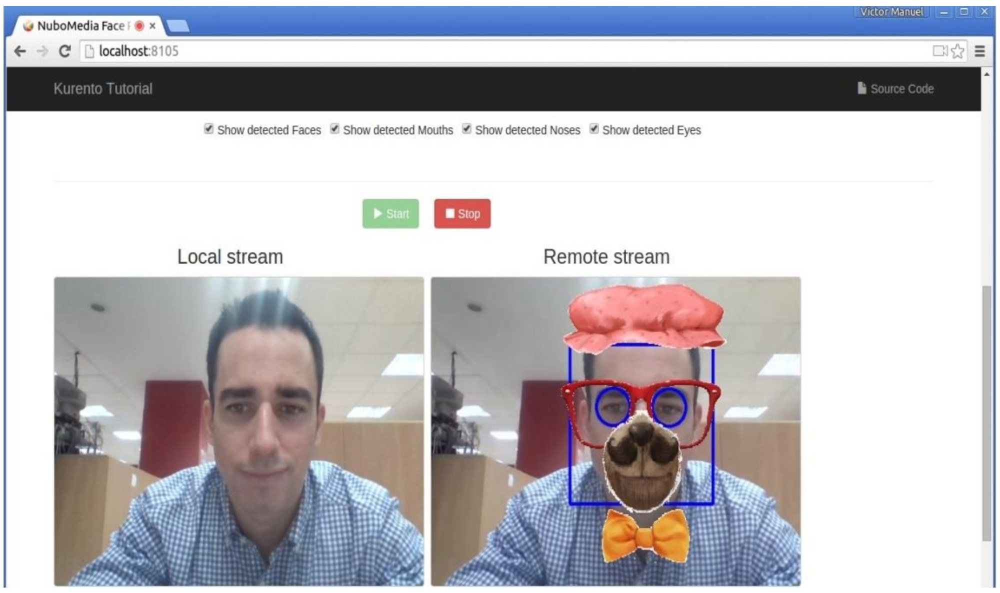
Remote stream

Proposal.

The objective is a result similar to the one shown in the image:



For compositions of the type displayed below…



…ImageOverlayFilter collects the metadata of the previous filter and places the indicated image according to the given position in the received metadata. An X-Y offset can be done by indicating the "height" and "width" of the image.

Parameter	Type	Description
imagePath	String	URL of the image to be overlaid.
ratioX	Float	X offset in relation to the upper left coordinate. The values can be positive or negative.
ratioY	Float	Y offset in relation to the upper left coordinate. The values can be positive or negative.
ratioWidth	Float	Image width. Only positive (0.0 to 1.0).
ratioHeight	Float	Image height. Only positive (0.0 to 1.0).

This filter is not supposed to detect anything. Its purpose is to use the metadata collected by the predecessor filter, place the indicated image and make the necessary adjustments according to the parameters and pass the videostream onto another filter or a WebRTCEndPoint.

Note:

This filter is meant to work in a way similar to ***"KmsDetectFaces"*** and ***"KmsShowFaces"*** (http://doc-kurento.readthedocs.io/en/stable/tutorials/java/tutorial-metadata.html), of which the former detects the face and the latter draws the rectangle according to the data provided by the former.

Second proposal

Current situation

The “***FaceOverlayFilter”*** has been used to overlay images on the videostream.

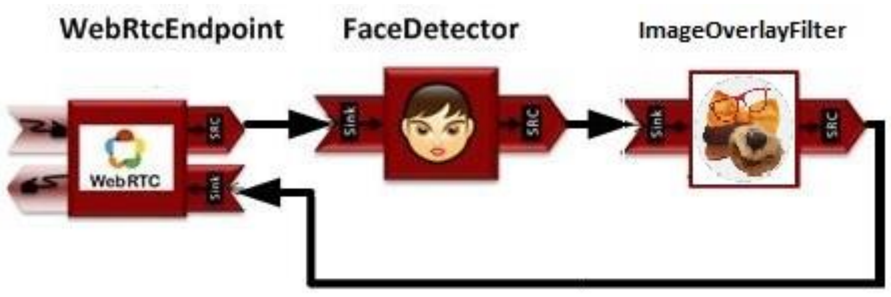
The filter does not work properly on high definition devices.

Having examined the filters created by Visual Tools for the NUBOMEDIA platform, it turns out that the detection of the different facial features is working properly for both high and low definition devices.

The filters created by Visual Tools are only for the detection of facial features, as they don’t have an API to overlay images on videostreaming.

The “***FaceDetector”*** filter will be used for this sprint.

We would need something to draw images using this filter (such as an **“ImageOverlayFilter”**) or similar.



Annex IV – QA report.

The demonstrator functionalities can be divided into the following categories:

· Interface and application flow.
· Interface and content localization.
· Login and connectivity functions.
· Friend management functions.
· Play Tales.
· Social features.
· Augmented Reality features
· Screenshots and screenshots management.

A variety of tests have been performed, such as interface, OS integration, performance, compatibility with tablets and current Apple versions. Contents and localization implementation have been checked, too.

As the results indicates, the development of the application progress properly and its functionalities show reasonable results, which means that a normal use is possible without any major handicaps. There are, however, still some imperfections related to the multi user video distribution as well as in the AR features. For the application to be launched, we recommend that these imperfections get fixed and the overall performance of the video features improves. Please, see below some notes for each functionality of the app, and some of the bugs that are recommended to be addressed before publishing:

Interface and application flow
The entire application can be used and there are no inaccessible sections or subsections. We reported the following faults:

- Error detected: The applications may shut down when exiting the children's tales.
- Incomplete functionality: The sound and font size options within the tales have not been enabled although they are on the UI.
-Usability: when moving between different sections, the screen mode switches from "landscape" to "portrait" which we consider an inconvenience for the user.

Localization
The app language is automatically detected, but can be changed during the tales.
- Error: The functionality has been implemented, but not all interface texts have been translated into all languages.

Login and connectivity functions
Easy sign in (no personal details required). No faults have been observed and a full use of the functionalities is possible.
- Usability: there is no option to reset your password in case you forget it.
Friend management
Functionality tests have been run and no significant faults have been detected.
Free mode
Tests for one to four users have been performed, and although the performance of the video and other VR features was poor, the application works.

-Error: Sometimes, when a user quits from the share play, the application shuts down for some of the members.
Tales mode
Tests for one to four users have been performed, and although the performance of the video and other VR features was poor, the application works.
-Error: Sometimes, when a user quits from the share play, the application shuts down for some of the members.
- Error: When the “father” finishes the tale, the share play won’t shut down for the “children”.

AR Functionalities (within a tale or free mode)
The functionality works properly despite its rather poor performance and difficult use, particular in share plays with 4 users.

Screenshot management.
No problems were reported.