

# A Centralized System to Manage Digital Contents in Multiple Advertising Displays

Arménio Baptista, Alina Trifan, and António J. R. Neves

DETI/IEETA

University of Aveiro

3810-193 Aveiro, Portugal

{armenio, alina.trifan, an}@ua.pt

**Abstract**—Current technology makes possible to enhance the way people experience advertising or general information notices, through the use of digital displays. These devices can be connected among them and centrally managed in order to individually display different contents. This process, also known as digital signage, is used nowadays in public spaces, transportation systems and retail stores, just to name a few applications. In this paper, we present a solution for a digital signage system, based on a centralized agent, through which digital contents can be managed and displayed on several autonomous stations. These stations can be either static, such as monitors, or mobile, such as autonomous robots, and they can be used in any type of multimedia advertising. The core of the presented solution is a web server, that stores the uploaded contents and exposes a web dashboard for their management. Registered users can manage and distribute the contents through the connected stations (or agents). The only requirement is a network connection between the server and the agents. The system was used with the aim of automatizing the dissemination and advertising of local research works and we present results of its use in the support of several research events that took place within our academic campus.

**Keywords**—Digital signage; Advertising; Cloud-based Platform; Digital Contents Management; Multimedia.

## I. INTRODUCTION

This manuscript is an extended version of the original paper presented at the 14th International Conference on Autonomic and Autonomous Systems (ICAS 2018) [1]. This extended version provides an overview about related systems, a deeper presentation of the developed system and extended experimental results. The main focus of this work is a solution for controlling multimedia resources and displaying them using a unique platform. This platform supports the management and manipulation of the resources with the goal of producing a final content to better fit the monitors associated to a specific terminal of the system we propose. As such, our solution allows the control, maintenance, composition and division of the multimedia resources across the stations. It also takes into account user permissions in order to control the access to these resources for each user registered in the platform. In order to handle the multimedia files uploaded by the users, the solution proposes the use of the FFmpeg library [2], which has compatibility with all major image and video formats and codecs.

One of the keys to the success of the retail industry passes by advertising to the general public, mainly by using marketing

strategies [3]. The technological evolution, more and more, has an important role in the marketing and the advertising of products [4]. Roggeveen et al. present experimental results that show that sales in hypermarkets are enhanced when digital displays are used [5].

This paper is structured into six sections, this Introduction being the first of them. Section II presents related work, mostly commercial solutions that can be compared with the proposed solution. Section III overviews the architecture and the implementation details of the proposed system. In Section IV we cover the methodology used to connect the digital displays, or agents, to the web server. We present experimental results in Section V and we assess the importance of this work and discuss future work directions in Section VI.

## II. RELATED WORK

Up until recently, most digital advertising systems relied on a manual configuration to display multimedia contents. This translated into a person moving to the locations of the physical displays and copy each content on them. Technological advances allowed digital marketing companies to adopt new and more advanced and appealing ways of advertising. Due to this growth, several areas and terms emerged, such is the case of the term digital signage, which is the one that better describes current digital advertising systems.

The term digital signage is nowadays widely used and can be defined as the remote management of digital display, typically tied in with sales, advertising and marketing [6]. It is usually implemented as a network of digital displays that are centrally managed and individually addressable to display digital contents, namely text, animations or video messages for advertising, informing, entertaining and merchandising to targeted audiences.

There are several systems and products supporting this technology as we can see in most of the existent advertisement displays presented in a wide area of scenarios, such as: airports, food chains, outdoor sites, shopping malls, just to name a few (Figure 1).

An extended search for the most relevant digital signage systems revealed that these mostly come from companies specialized in building these solutions as out of the box products. Consequently, all of the encountered solutions are closed and the implementation behind them is not made public.



Figure 1. A digital signage example in Times Square, New York.

1) *Yodeck*: Yodeck [7] is the most similar system compared to our project (Figure 2). The Yodeck system consists on a software solution to manage a series of monitors using only one product: a Raspberry PI [8] called "Yodeck Playbox", which brings a MicroSD card containing proprietary Yodeck software. The user only needs to buy the Raspberry PI and connect it to a monitor, create an account, upload the multimedia contents to the platform and associate these contents to a virtual monitor.

Their solution divides the contents into 4 categories:

- Media - contents uploaded to the system: image files, video files, documents or web site addresses.
- Playlists - set of media with a predefined duration: an image slideshow or a video.
- Widgets - small widgets that display useful info: Really Simple Syndication (RSS) feed, a clock, etc.
- Shows - set of media, playlists and widgets that can be associated to a virtual monitor (i.e. a Raspberry PI).



Figure 2. Illustration of the Yodeck digital signage solution [9].

2) *Xarevision*: Xarevision [10] is a leading company in technologies for retail and operates the biggest in-store digital network in Portugal, reaching over 40% of the active population. It administers broad digital networks of centrally managed displays and also creates differentiated interactive media, like gesture recognition, georeferencing and individual addressing.

Their technologies are apply mainly in the retail industry and they offer the following three products:

- Queue management - this product offers management solutions to be applied in supermarkets. It can be found in almost all big surfaces and it is used to optimize waiting times and to increase the customer service. Generally, it is composed of a ticket dispenser and a customer calling screen.
- New media - includes features like gesture recognition, georeferencing and individual addressing. Moreover, it supports remote control and supervision of the distributed networks across the country.
- Digital signage and Corporate TV - last but not least, the digital signage solution, which can be related to the work we present in this manuscript. Their solution supports the client from providing the installation of hardware and network management, to supporting content creation and update, as well as continuously monitoring technical aspects during the system's lifetime.

3) *JCDecaux*: JCDecaux [11] is probably the most recognized advertising company in the world in outdoor sites (Figure 3). Despite there is no published information about the development and implementation of their terminals, it is interesting to refer it here because of the distributed advertising system they possess.

Their main products are the billboards that can be found mainly in bus stops and open public space or shopping centers. Our solution was inspired by these digital billboards and we aim to have a similar impact, with cheaper and open-source elements.

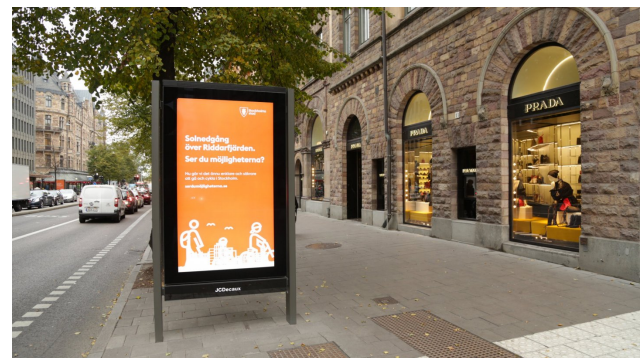


Figure 3. JCDecaux billboard [12].

4) *Enplug*: Enplug [13] is a digital signage software company that provides contents advertising. It supports social contents, like Youtube and Instagram feeds, RSS feeds, digital menus, Google calendars, graphics and videos.

5) *NoviSign*: NoviSign [14] is a digital signage software company with a cloud-based solution that supports Windows, Android and Chrome OS players. It has an online control dashboard, Studio, that allows the creation, edition and management of digital contents from any browser, as well as the update of the screens in real time. It supports a variety of widgets, like text, images, video, RSS, games, polls and social widgets. Also, it can be integrated with a camera for face

recognition, a Radio Frequency Identification (RFID) reader and barcode and sensors for triggering ads, events or slides.

6) *Mvix*: Mvix [15] offers digital signage systems with a portable physical module, similar to a Raspberry PI, which has a pre-loaded content management software capable of communicating with a cloud-based server. It offers a web-based software for creating, editing and updating digital signage templates. It includes content apps and widgets which enable users to display High Definition videos and images, live web pages, RSS feeds and media animations. This digital signage solution supports playlist management, comprehensive calendar-based scheduling and a multi-role user management.

7) *ScreenCloud*: ScreenCloud [16] is a digital signage solution that can turn almost any screen into a display of multimedia contents. It offers compatibility with a large variety of devices, such as: Google Chromebit, Amazon Fire TV, Google Chromebox, Android TV, Apple Mac Mini, Mi Box, etc. The scheduling, management and upload of contents is made using a web browser.

All the identified digital signage solutions were relevant for the development of our proposal, since they allow us to identify possible requirements and features to be included. Nevertheless, the lack of scientific open publications on such systems, together with the lack of architectural details of the previously described solutions, makes it difficult to provide straight-forward comparisons between the system we propose and the commercial ones.

### III. ARCHITECTURE

Our main goal was the design of a system capable to store, manipulate and manage multimedia contents uploaded by its users. This led to the identification of three major agents involved: the web server, the control dashboard and the monitors. The resulting architecture is presented in Figure 4.

This architecture allows the web server to control all the monitors (using HTTP requests to communicate) and to expose a web dashboard by means of which the users can manage them. As the web server is the unique point of communication, the synchronization of the updates made by the users and the monitors is immediate.

#### A. Resources division

In order to split and organize the multimedia contents, or resources, a solution of three components is proposed, as presented in Figure 5:

- **Contents:** base element of the multimedia resources, which can be: images, videos and presentation files uploaded by the users.
- **Timelines:** set of Contents with a predefined sequence, much similar to a video composed of different contents. If the Content is an image, the user has to define the duration of the image in the Timeline. If the Content is

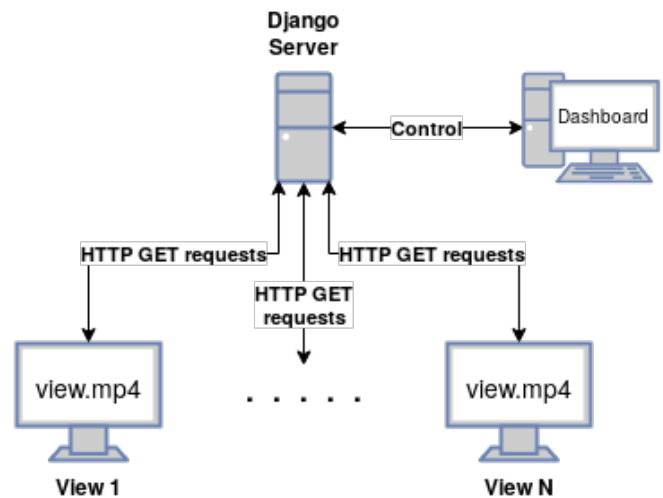


Figure 4. Proposed system architecture. The web server was built based on the Django [17] framework and is the central unit of the system. It exposes a web dashboard through which multimedia contents can be uploaded and managed. The server communicates with one or multiple terminals in order to distribute these contents.

a presentation file, the user has to define the duration of each slide or page of the presentation.

- **Views:** set of Timelines with a predefined sequence that is associated to a physical terminal. This represents the final product to be displayed on the monitor. The only way for a View to be created is through a connection to a terminal.

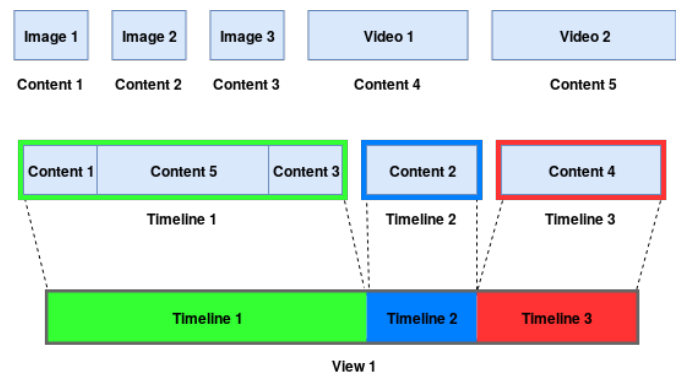


Figure 5. Resources composition. Individual contents can be joined into one or more timelines, which will then be associated to one or multiple views, or displays.

This division of resources gives users the freedom to create and dispose multimedia contents into any chosen order. It also allows them to reuse the Timelines in different Views without the need of recreating them again.

## B. Permissions

A control policy was identified as a requirement in order to protect the privacy of the multimedia contents among users. This policy has to support the blocking or disabling of modifications of resources that do not belong to a given logged-in user. In order to restrict the improper access of the users to multimedia resources, an authentication system with different permission levels was implemented. The Django framework provides a user authentication system, which facilitated the implementation.

The permissions levels of the proposed system are illustrated in Figure 6.

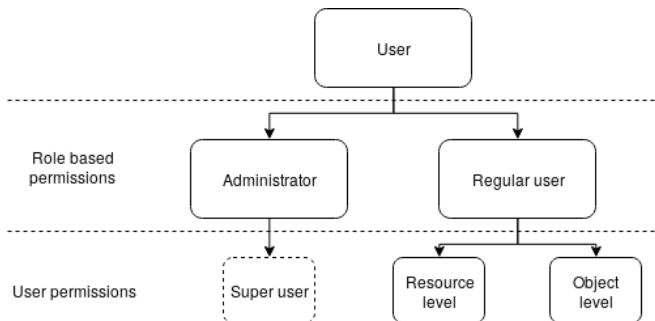


Figure 6. Users permissions.

The role based permissions contemplate two roles: administrator and regular user. The administrator has access and control over any resource in the system. Moreover, she can access a section of the dashboard that allows managing regular users in terms of permissions and CRUD operations (Create, Read, Update, Delete). The system has two types of user roles:

- **Administrator:** user responsible to manage the users access to the dashboard and their permissions for each resource. Additionally, the administrator can add, edit and delete any resource, having no restrictions for her actions.
- **Regular User:** user with permissions predefined by the administrator and that can only access the resources predefined by those permissions.

For regular users, we included a sub-level of permissions, with two distinct facets: the resource and object levels. The resource level refers to the different resources presented in Section III-A. The object level refers to an object in particular, a specific resource of the type Content, Timeline or View. The administrator manages these permissions through the web dashboard.

1) *Resource level:* The resource level has three different types of permissions, associated to each of the contents type: Contents, Timelines and Views. When creating a user, the administrator fills in a form with basic information associated to the user, as illustrated in Figure 7. Note that this form has three checkbox fields, each one for a resource (Content, Timeline and View). These checkboxes are the permissions at

the resource level of the user being created. These will define which resources the user can create and edit.

Only the allowed resources will be displayed in the left navigation bar of the interface of the corresponding user. That is, a user with no permissions over Contents, will only have view access of Timelines and Views the left navigation bar.

Figure 7. User creation web page.

2) *Object level:* The object level refers to which object from the resources the user has access. By default, the user only has access to the objects he has created. However, the administrator can extend the access of a user to a certain object, based on a Access Control List. This list of permissions is associated to an object and contains the permissions of each user for that given object.

In Figure 14, there is a padlock for each object of the table. This padlock redirects to a page where the administrator edits the permissions over the respective object, as in Figure 8. This page lists all users with resource level permission over the resource of the object, so the administrator can grant access to that particular object.

Figure 8. Modification of object level permissions of a Content.

This permission is particularly useful if some resource needs to be updated with contents that belong to a different user.

The users with Timelines and Views permissions can view all the underlying resources in order to create a Timeline or View, that is:

- To create a View, the user can use any of the underlying Timelines.
- To create a Timeline, the user can use any of the underlying Contents.

These blocking permissions allow, not only that users can be associated to a specific task, but they also provide cooperation among users in order to reach a final multimedia product. As an example of specific tasks we can think of giving Content and Timeline permissions (Resource level) to a designer responsible for providing multimedia contents. Alternatively,

View permissions can be granted to a user responsible for the monitors inside of a specific building.

### C. Files format support

One of the essential requirements of the system is the capacity to upload, store and transform the files to be displayed later. The compatibility of the system with the formats of the files needed to be considered. This meant not only validating the uploaded file, but also ensuring compatibility with the library chosen to operate over these files.

We consider the upload of three types of files: image, video and presentation files. Having studied the existent commercial solutions, we understood that these types grant a large range over the needs of the users. They enable the creation of slideshows with predefined duration and they allow video concatenation, which in the end leads to one final video product.

1) *Image formats support*: For images, we opted for two of the most used image formats, that are used at worldwide scale and cover a wide range of contents:

- JPEG [18] (.jpeg) - JPEG is a lossy based image format, but also supports lossless compression.
- PNG [19] (.png) - PNG is a lossless based image format.

2) *Video formats support*: For videos, the system supports two video formats, both of them being multimedia container formats that can contain both audio and video data:

- AVI [20] (.avi) - AVI is a multimedia container format created by Microsoft.
- MP4 [21] (.mp4) - MP4 is also a multimedia container and one of the most used video formats.

All the video codecs supported by FFmpeg can be used (ex. H.264 [22], H.265 [23], just to name a few).

3) *Presentation formats support*: In the case of the presentation formats, we decided to support PDF and PowerPoint files (.ppt and .pptx extensions) due to their large usage. The developed system supports files with multiple pages (or slides). The upload of the files is made with the File Upload capability of Django and saved in a media directory of the server.

### D. Web server - control dashboard

The core of the presented solution is a web server that can store the uploaded contents and exposes a web dashboard for their management. During the following sections we present the main processes that the web server executes and the workflows it uses to reach the final product.

1) *Files upload and validation*: When a file is uploaded with the Django File Uploader, the web server saves the file and initiates its validation. It starts by validating the extension of the file among the supported ones. If the extension is a valid one, the web server further validates the uploaded file according to its format. Our main approaches for these validations are:

- Image format - for image validation, the `img_hdr` (Table I) Python library is used.
- Video format - for video validation, the `MoviePy` [24] library is used. It is a Python library with video processing capabilities. It allows cuts, concatenations, title insertions, video compositing, video processing and creation of custom effects.
- PDF format - for PDF validation, the `PyPDF2` (Table I) Python library is used.
- PPT format - for PPT validation, the `catppt` [25] Linux command is used.
- PPTX format - for PPTX validation, the `python-pptx` (Table I) Python library is used.

If the file is valid, the web server returns the acknowledgement to the user. Otherwise, the file is deleted and an error message is returned to the user.

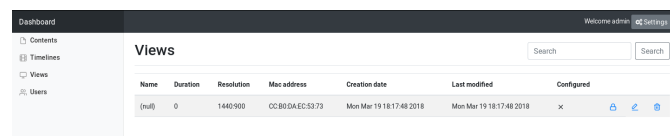
2) *PDF transformations*: In order to create a video from a PDF file and due to the requirements of FFmpeg library, which works with image or video files, the web server converts every slide of the PDF into an image file. These images are saved in a directory on the server and all the names are saved in a text file. FFmpeg reads the text file and creates the video with the duration specified by the user for each slide. In the end, all images are deleted and the video file is saved.

3) *PowerPoint transformations*: The conversion process of PowerPoint files to video is an extension of the PDF transformation. The PowerPoint file is converted to PDF with the command line LibreOffice [35] converter. Then, the PDF transformation is performed.

4) *Timelines creation*: The creation of a Timeline is quite intuitive to the user. The process starts by accessing the Timeline tab and clicking the Add button. To create a Timeline, the user needs to input a name for the Timeline and to associate Contents to it.

After the submission, the web server initiates the creation of a preview of the Timeline. The preview has a predefined 400×300px video resolution and it will be available in the edition form of the Timeline, once the process is completed.

5) *Views creation*: The process of creating a View is made automatically with the connection of a Raspberry PI. This step can be customized for different platforms but it is currently optimized for Raspbian. The View will be displayed in the dashboard with an empty name and with a false configured flag, as in Figure 9. In other words, the View has to be configured in order to display contents and to be distinguished from others.



Name	Duration	Resolution	Mac address	Creation date	Last modified	Configured
(null)	0	1440x900	CC:80:DA:EC:53:73	Mon Mar 19 18:17:48 2018	Mon Mar 19 18:17:48 2018	<input type="checkbox"/>

Figure 9. Dashboard Views web page with an unconfigured View.

TABLE I. Most important Python libraries used in this work.

Python Library	Description
imghdr [26]	Determines the type of image contained in a file or byte stream. It is used to validate when a file with an image extension is uploaded (.jpeg or .png).
PyPDF2 [27]	Designed specially to operate over a PDF file. It is used to validate when a file with PDF extension is uploaded.
pdf2image [28]	Reads a PDF file and converts it into a Python Imaging Library (PIL) object. It is used to convert the PDF files to image, to be later converted in a slideshow video.
python-pptx [29]	A handler for Microsoft PowerPoint files (.pptx), that allows to create and modify the presentations. However, in our solution, it is only used to validate the uploaded files.
ffmpy [30]	A command line wrapper to FFmpeg [2]. It uses the Python subprocess [31] module to execute the commands.
screeninfo [32]	A module to fetch the location and the resolution of the physical screens connected. In our solution, it is used to fetch the resolution of the screens connected to the Raspberry PI devices.
netifaces [33]	A module to fetch all the interfaces connected to the local device. It is used to find a suitable MAC address of the Raspberry PI.
requests [34]	A module designed to send HTTP requests. In our system, it is useful to send the HTTP GET and HTTP POST requests from the Raspberry PI to the server.

Upon the configuration of a View, if it has any Timelines associated, the server initiates the process to create the MP4 file associated to the View. This file is compressed using the H.264 standard [22] and encoded with YUV420 at 25 frames per second. To better fit the resolution associated to the monitor, all the Contents are adapted to this resolution. That is, when a View is configured with Timelines associated, the system goes through all the Contents associated to these Timelines and makes the changes needed to fit the screen resolution, Content by Content.

To create, manipulate and merge these files, we chose the FFmpeg library since it has compatibility with all major video and image formats. The FFmpeg library adapts the Contents using mostly padding and resize transformations. When iterating over the frames of the Contents, the FFmpeg library resizes the frames which have different sizes and applies padding to keep the Contents aspect ratio. This process is explained in detail in Section III-E.

### E. Multimedia contents transformation

In Figure 10 the transformations of FFmpeg are illustrated. The figure shows five frames of a Timeline with three Contents (an image, a video and another image) with these resolutions:

- Image 1: 1000×665 px
- Video: 1280×720 px
- Image 2: 6000×1977 px

Before explaining the FFmpeg transformations to these 5 frames, it is important to refer 3 effects: letterboxing, pillarboxing and windowboxing [36]. Letterboxing consists in the transformation of frames with widescreen aspect ratio (16:9) to a standard-width video ratio (4:3) while preserving the frames original aspect ratio. This transformation consists of a padding transformation both on top and bottom of the frames. On the contrary, the pillarboxing effect consists in the transformation of a standard-width video format into a widescreen aspect ratio by applying padding into the frames both on left and right. Windowboxing consists of the combination of both effects: letterboxing and pillarboxing. This is noticeable when the frames of a video are centered in the screen with a padding

effect all around them. This happens when the resolution of the screen is bigger than the frames and no resize transformation is used.

Using FFmpeg with the arguments:

- "scale" and "force\_original\_aspect\_ratio"
- "pad"

makes it is possible to apply the intended transformations to the frames.

The "scale" parameter allows to specify the scale resolution to apply to the frames, while using the "force\_original\_aspect\_ratio" to maintain the original aspect ratio of the images. This transformation will upscale the frames, if the resolution of the screen is bigger than the frames, and downscale, in the opposite situation. The "pad" parameter applies padding to the frames after the scale transformation. When the frames of the Contents have a different aspect ratio than the one of the screen, the letterboxing and pillarboxing are perceptible.

Back to Figure 10, the example uses FFmpeg to fit a monitor with a 1366×768 resolution which has a 1.78:1 aspect ratio. In Image 1 from Figure 10, as the resolution of the image is smaller than the screen, FFmpeg resizes the frames using an upscaling transformation to fit the screen. Although, as the aspect ratio of the image (1.5:1) is smaller than the screen (1.78:1), FFmpeg also applies a padding effect, resulting in a pillarboxing effect.

In the Video from Figure 10, FFmpeg resizes the frames using an upscaling transformation. In this case, the aspect ratio of the frames (1.78:1) and the screen (1.78:1) are equal, so FFmpeg doesn't apply the padding effect and the frames fit perfectly the screen.

Image 2 from Figure 10 is the opposite of Image 1. The resolution of the frames is much bigger than the screen and FFmpeg resizes the frames, but using a downscale transformation. As for the padding effect, the frames of the image (3:1) have a much bigger aspect ratio than the screen (1.78:1), so FFmpeg applies padding to the frames, resulting in a letterboxing effect.



Figure 10. Expansion of a Timeline with five frames from three Contents (Image 1 (fox), Video (tree frames) and Image 2 (mountain landscape)).

With these FFmpeg arguments, the result will never reach a windowboxing effect, because the "scale" parameter will always try to resize the frames to fit the screen and the "pad" parameter will compensate the difference between the aspect ratios, either with letterboxing or pillarboxing effects. Using the FFmpeg upscaling transformation over the frames of the Contents may have influence in the final quality of the frames if the resolution of the image is much smaller than the screen, which forces the users to upload Contents with more appropriate resolutions.

#### IV. AGENTS - MONITORS

The need to display the contents in the monitors and the need to have a terminal capable to communicate with the web server motivated our choice of a Raspberry PI as part of the distributed system. This equipment has enough processing capabilities to display the contents with a low need for space area and low energy dependency. It also has a 802.11b/g/n/ac networking interface to communicate over Wi-Fi in order to download the contents from the web server.

To test the system, we used the Raspberry PI Model 3B+, which needs a dedicated power plug to connect to the power supply. Otherwise the Raspberry PI shuts down on boot. We also tested the system using a Raspberry PI Model 3B and, on the contrary of Model 3B+, it only needs a Micro-USB connector as power supply (present in almost every Liquid Crystal Display or different types of monitors).

##### A. Life cycle

Regarding the development of the Raspberry PI system, the communication with the web server is made over HTTP requests and, as most of the case studies were in the Eduroam network (which does not allow peer-to-peer communication), all the communication requests must come from the Raspberry PI to the web server and not vice-versa. This led us to create a life cycle of the system, using a Python script to control the display and the communication. When the Raspberry PI starts, the script runs on boot and goes through a series of steps in order to register on the server (the first time it connects) and download the video.

This life cycle, as illustrated in Figure 11, is divided into four major states.

1) *State 1 - send\_init\_info()*: In this state, the script tries to fetch the data to be sent to the web server, namely:

- MAC address - using the netifaces [33] Python library this information allows the server to distinguish between the different terminals, as the MAC address is a unique identifier of each of the terminals. Moreover, it allows the user to know which View is associated to a terminal.
- Screen resolution - using the screeninfo [32] Python library, the resolution is used to adapt the Contents associated to a given View to its screen resolution.

To fetch the MAC address, the script tries to fetch the address associated to the Ethernet interface and if it fails, tries to fetch the address associated to the the default wireless lan interface (wlan0). If both cases fail, it tries to fetch the address associated to another wireless interface present in the list of interfaces.

To fetch the screen resolution, screeninfo [32] fetches the size and location of every physical screen connected. The script chooses the screen resolution of the default monitor. If it fails, the script waits for a period of time and tries to fetch it again when a monitor is successfully connected. After this, it sends the information to the web server with a HTTP POST Request. The server, when receiving this information, checks if the MAC address already exists and returns the path in the server to the video corresponding to that View.

2) *State 2 - get\_video()*: The second state after fetching the path to the corresponding video is responsible for downloading the video from the web server with a HTTP GET request and store it in the file system of the Raspberry PI. If it fails (if the View was not configured yet or the server is down), the terminal waits thirty seconds by default, and tries to download it again. This loop is repeated until the video is successfully downloaded.

3) *State 3 - play\_video()*: The third state is only responsible to launch the process of the video player in order to play the video on the monitor. To play the video, the OMX player is used, which is a command line player specifically made for the Raspberry PI GPU [37]. The player is launched with the following command line flags:

- -g - used to generate the log file.
- -b - used to set background to color black.
- --no-osd - used to hide status information on the screen.
- --loop - used to repeat the video indefinitely.

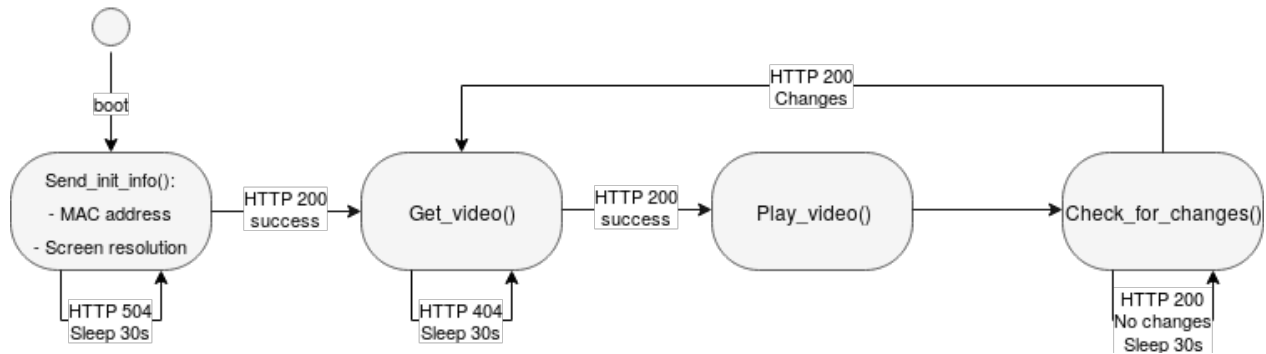


Figure 11. The different states of the monitor life cycle.

4) *State 4 - check\_for\_changes():* The fourth and final state is repeated infinitely and is responsible for polling the web server, each sixty seconds by default, to check for any changes in the View. If any changes are detected, the script jumps back to State 2 to download the video and continue the cycle, as in Figure 11. If at any moment the Internet connection is lost, it continues to poll the server until a response is obtained. While this happens, the previously downloaded video continues to be played. This means that the video is always displayed, even with no Internet connection, after a successful download.

In Figure 12, the terminal output of a Raspberry PI, from the connection with the server until the video is successfully downloaded, is presented. As this was just a test, the execution of the program stops when the video is downloaded. In a normal situation, the execution continues with State 4.

```
File Edit View Search Terminal Help
pi@rpi:~/monitor$ ls
requirements.txt run.py startup.sh
pi@rpi:~/monitor$ python3 run.py
14:58:03.876 - Monitor Logger - INFO - Initializing monitor
14:58:03.880 - Monitor Logger - INFO - Sending initial info to server
14:58:05.298 - Monitor Logger - INFO - Video unavailable. Retrying in 30s...
14:58:37.294 - Monitor Logger - INFO - Video unavailable. Retrying in 30s...
14:59:13.344 - Monitor Logger - INFO - Video download complete
14:59:13.345 - Monitor Logger - INFO - Running
pi@rpi:~/monitor$ ls
Monitor.log requirements.txt run.py startup.sh view.mp4
pi@rpi:~/monitor$
```

Figure 12. Raspberry PI terminal output. It is visible, from the output, that the video was successfully downloaded.

### B. Web server and Raspberry PI communication

The communication between the web server and the Raspberry PI is made with HTTP GET and HTTP POST Requests from the Raspberry PI to the web server. Since the system was designed to be deployed in the Eduroam network and as it does not allow peer-to-peer communication, all the requests must come from the agents to the server.

The web server exposes three specific URL for the terminals to make the requests:

- `login/` - used to authenticate.

- `monitor/new_monitor/` - used to register in the system or/and fetch the URL from the corresponding video.
- `monitor/check_for_changes/` - used to check for changes in the View.

Every request to the web server requires the terminal to log into the system in order to get the Cross Site Request Forgery (CSRF) token, provided by Django, to protect against Cross Site Request Forgeries attacks. A CSRF hole is when a malicious site can cause a visitor's browser to make a request to the server that causes a change on the server. This can happen as the request comes with the user's cookies [38]. According to the Django documentation this type of attack occurs when a malicious website contains a link, a form button or some JavaScript that is intended to perform some action on your website, using the credentials of a logged-in user who visits the malicious site in their browser [39].

The CSRF middleware and template tag of Django provides automatic mechanisms to protect against this type of attacks.

## V. RESULTS

The system presented in this paper is completely operational and ready to manage the upload of multimedia contents and display them. It implies the existence of a computer to host the server (with proper image and video processing capabilities), monitors to display the contents, single board to connect to the monitor (Raspberry PI) and, of course, network connection between the server and the terminals.

### A. Dashboard

The dashboard was designed with emphasis on the requirements of Resources and Permissions in order to facilitate the interaction of the users with the system. The dashboard gives control over four main resources: Contents, Timelines, Views and Users, having a navigation bar with these resources, as illustrated in Figure 13.

As noticeable in Figure 13, the dashboard has a top navigation bar, which has a dropdown button so the user can edit



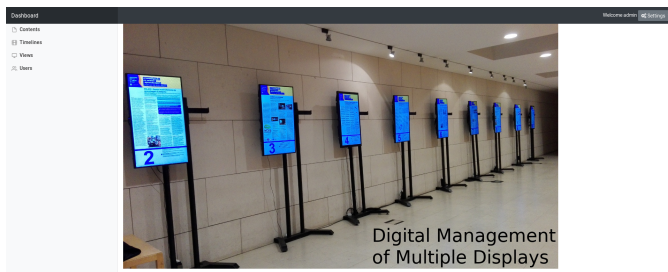


Figure 13. Dashboard home example.

her personal informations or logout. The left navigation bar contains the four possible navigations. This left navigation bar only shows the possible navigations to which the user has access. Only the users with permissions over the resources can access and control the respective resource. Only the administrator can edit these permissions and have access to the Users page.

When accessing a resource from the left navigation bar, one of the pages presented in Figure 14 (except User) is displayed. This page allows the visualization over the existing Contents, Timelines or Views to which the user has access. It also allows to create (except for Views as explained in Section III-D), edit or delete a resource. If the user is an administrator she can edit the permissions (object level) of the users to that resource.

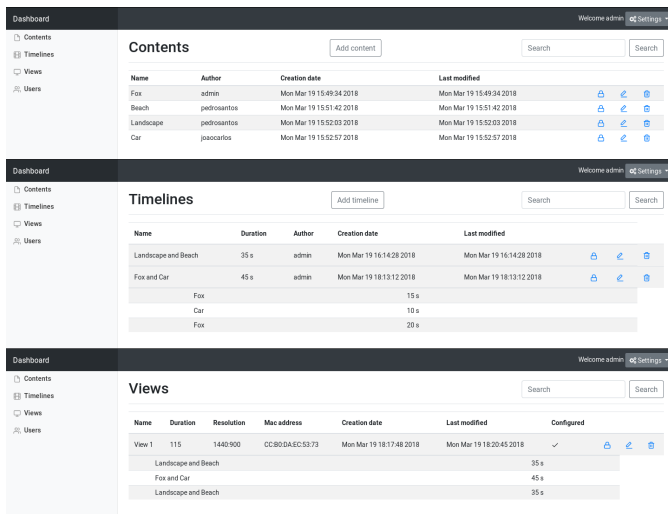


Figure 14. Visualization of Contents, Timelines and Views.

The dashboard was built using three known technologies: HTML, CSS and JavaScript. This triad allows to create interfaces using HTML for structure, CSS for styling and JavaScript to control the flow of some elements.

When editing a resource, a form is displayed so the user can edit the data associated to the resource. In the case of Contents, the form contains two fields: the name of the Content and a file upload button. However, when editing a Timeline or a View

(Figure 15), a table with the Contents or Timelines associated to the resource, respectively, is displayed. This table allows the addition of objects from the dropdown button. Moreover, the objects from the table can be dragged and dropped into the intended order. When editing a Timeline, the table also has a duration input field for each image or presentation Content. By filling in this field the user can specify the intended duration for each image or slide that will be part of the Timeline. In addition, a preview window displaying the video reproduced from the last Timeline submission is available.

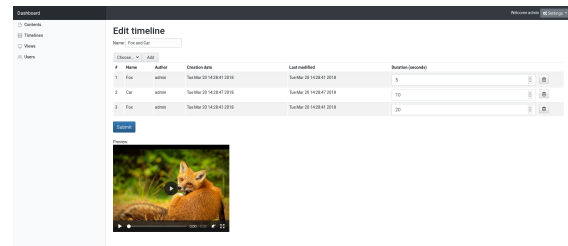


Figure 15. Edition of a Timeline.

### B. Final result

To test the system, we used a server at the University of Aveiro to run the web server. This makes the server available to all the terminals inside the university network (Eduroam), making the system portable and usable in any department. The access to the dashboard is also possible in any local with Internet connection (using a Virtual Private Network). The server has the HTTP Port (number 80) open in order to enable requests from other hosts inside the Eduroam network.

The system was tested in two environments: Institute of Electronics and Informatics Engineering of Aveiro (IEETA) building and Students@DETI [40] event. Both environments require the dissemination of multimedia contents. The following sections explain the procedure and details about each one.

1) *IEETA monitors*: Since research promoting monitors were placed across IEETA, the display of multimedia contents was based on a USB flash drive to store the contents. This would be manually connected to the monitor. Moreover, the process of updating contents consisted in removing the USB flash drive and manually updating them by transferring new ones from a computer. The need to automatize and speed up this process enabled the development of the solution we propose.

In Figure 16, we can see a real Full HD monitor (1920×1080 px of resolution and a 1.78:1 aspect ratio) with a Raspberry PI single board connected to the developed system. The final product, the video stream for that particular View is displayed.

The building has three monitors distributed across its area. We used the three monitors simultaneously. They were connected to Raspberry PI and the following Contents were used.

- 7 PDFs with 15 seconds of display.



Figure 16. Final result of a monitor in IEETA.

- 3 PowerPoint files with 5 seconds for each slide.
- 2 videos with MP4 format.

All of these Contents were composed in one Timeline, resulting in a video with 1 hour and 51 minutes. The system was updating automatically the video if any changes were made. The only delays in this process come from the difference between the time of creation of the video on the server and the time of polling and the download time.

2) *Students@DETI*: Every year, the Electronics, Telematics and Informatics Department of University of Aveiro hosts a research events that promotes projects of lecturers and students of the department. This year, we deployed our system in three spots of the event in order to test and advertise it. The system displayed multimedia contents uploaded by the organizers of the event with contents relevant to the event, mainly PDF, PowerPoint and video files.



Figure 17. Demonstration and exhibition of the system at Students@DETI, using a vertical monitor.

Two of the three spots were using a projector to display the multimedia contents into a wall, one at the entrance of the event and the other one in a hall. These two spots were displaying three PDF Contents with 15 seconds for each slide converted to a Timeline. The third spot was a smaller

monitor in an exhibition room. The monitor was displaying posters from the different students projects showcased at the event. As the posters had a vertical orientation, we changed the orientation of the Raspberry PI and the server made the changes automatically to better fit the monitor. In Figure 17 the monitor was displaying a Timeline with 15 PDF posters with 10 seconds for each one.

## VI. CONCLUSION

This project evolved as a necessity that we have identified within our local research community. We were motivated to develop an affordable solution, based on open-source technologies in order to overcome the lack of open available solutions. The main goal of this paper was to propose a solution to manage multiple multimedia contents in order to display them in multiple monitors. The system we have presented, as a whole, is operational and fully functional within our research unit. It manages the upload of multimedia contents and their display. It implies the existence of a computer to host the server (with proper image and video processing capability), monitors to display the contents, single board elements to connect to the monitor (like a Raspberry PI) and, of course, network connection between the server and the terminals.

A global overview over the results highlights different features that we intend to improve, as future work:

- The aspect of the dashboard that needs to be more appealing and intuitive to the user. Moreover, it could display for example, snapshots of the Contents.
- A more advanced Timeline editor in order to give the user a greater control over the sequence of Contents.
- The capability to fetch information in real time about the monitor in order to show the status in the dashboard. Moreover, we envision making it possible to run certain commands over the monitor (such as shut down or turn on).

As the system evolves we intend to expand the supported formats of Contents, including for example audio files. Another interesting feature that we plan to develop is the support for the interaction between the target audience and the system, either by a physical contact or even by voice control. This interaction would allow the user to pause and skip contents, for example.

## ACKNOWLEDGMENT

This work was partially funded by FEDER (Programa Operacional Factores de Competitividade - COMPETE), by National Funds through the FCT - Foundation for Science and Technology in the context of the project UID/CEC/00127/2013 and by the Integrated Programme of SR&TD "SOCA" (Ref. CENTRO-01-0145-FEDER-000010), co-funded by Centro 2020 program, Portugal 2020, European Union, through the European Regional Development Fund.

## REFERENCES

- [1] A. Baptista, A. Trifan, and A. Neves, "Digital management of multiple advertising displays," in *Proc. of 14th International Conference on Autonomic and Autonomous Systems, ICAS 2018*, 2018, pp. 36–41.
- [2] "FFmpeg official website." <https://www.ffmpeg.org/>, accessed: 2018-05-29.
- [3] A. Di Rienzo, F. Garzotto, P. Cremonesi, C. Frà, and M. Valla, "Towards a smart retail environment," in *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM, 2015, pp. 779–782.
- [4] N. I. Bruce, B. Murthi, and R. C. Rao, "A dynamic model for digital advertising: The effects of creative format, message content, and targeting on engagement," *Journal of Marketing Research*, vol. 54, no. 2, pp. 202–218, 2017.
- [5] A. L. Roggeveen, J. Nordfält, and D. Grewal, "Do digital displays enhance sales? role of retail format and message content," *Journal of Retailing*, vol. 92, no. 1, pp. 122–131, 2016.
- [6] J. Schaeffler, *Digital Signage: Software, Networks, Advertising, and Displays: A Primer for Understanding the Business*. Taylor & Francis, 2012. [Online]. Available: <https://books.google.pt/books?id=9ZUrt7-ixQC>
- [7] "Yodeck official website." <https://www.yodeck.com/>, accessed: 2018-05-30.
- [8] "Raspberry PI official website." <https://www.raspberrypi.org/>, accessed: 2018-05-22.
- [9] "Yodeck digital signage solution." <https://www.yodeck.com/wp-content/uploads/2015/12/yodeck-04-2.png>, accessed: 2018-06-08.
- [10] "Xarevision," <https://www.xarevision.pt/>, accessed: 2018-07-30.
- [11] "JCDecaux official website." <http://www.jcdecaux.com/>, accessed: 2018-05-30.
- [12] "JCDecaux billboard." <http://www.jcdecaux.com>, accessed: 2018-05-30.
- [13] "Enplug," <https://www.enplug.com/>, accessed: 2018-06-30.
- [14] "NoviSign," <https://www.novisign.com/>, accessed: 2018-06-30.
- [15] "Mvix," <http://www.mvixusa.com/systems/>, accessed: 2018-06-30.
- [16] "ScreenCloud," <https://screen.cloud/>, accessed: 2018-06-30.
- [17] "Django official website." <https://djangoproject.com>, accessed: 2018-05-22.
- [18] G. K. Wallace, "The jpeg still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.
- [19] C. Wilbur, "Png: The definitive guide," *Journal of Computing in Higher Education*, vol. 12, no. 2, pp. 94–97, 2001.
- [20] "Audio Video Interleave (AVI)," [https://msdn.microsoft.com/en-us/library/windows/desktop/dd318187\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/dd318187(v=vs.85).aspx), accessed: 2018-05-28.
- [21] "MPEG-4," <https://mpeg.chiariglione.org/standards/mpeg-4>, accessed: 2018-05-28.
- [22] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.
- [23] V. Sze, M. Budagavi, and G. J. Sullivan, "High efficiency video coding (hevc)," *Integrated Circuit and Systems, Algorithms and Architectures*. Springer, vol. 39, p. 40, 2014.
- [24] "MoviePy," <https://zulko.github.io/moviepy/>, accessed: 2018-05-22.
- [25] "catppt command," <https://linux.die.net/man/1/catppt>, accessed: 2018-07-01.
- [26] "Python imgHDR," <https://docs.python.org/3.4/librar/imghdr.html>, accessed: 2018-05-28.
- [27] "Python PyPDF2," <http://mstamy2.github.io/PyPDF2/>, accessed: 2018-05-28.
- [28] "Python pdf2image," <https://github.com/Belval/pdf2image>, accessed: 2018-05-29.
- [29] "Python python-pptx," <https://github.com/scanny/python-pptx>, accessed: 2018-05-29.
- [30] "Python ffmpeg," <https://github.com/Ch00k/ffmpeg>, accessed: 2018-05-29.
- [31] "Python subprocess," <https://docs.python.org/3/library/subprocess.html>, accessed: 2018-05-29.
- [32] "Python screeninfo," <https://github.com/r-r/screeninfo>, accessed: 2018-05-29.
- [33] "Python netifaces," <https://github.com/al45tair/netifaces>, accessed: 2018-05-29.
- [34] "Python requests," <https://github.com/requests/requests>, accessed: 2018-05-29.
- [35] "LibreOffice," <https://www.libreoffice.org/>, accessed: 2018-07-01.
- [36] C. Poynton, *Digital video and HD: Algorithms and Interfaces*. Elsevier, 2012.
- [37] "Omxplayer Github page." <https://github.com/popcornmix/omxplayer>, accessed: 2018-05-31.
- [38] "Cross-site request forgery," <https://www.squarefree.com/securitytips/web-developers.html>, accessed: 2018-07-02.
- [39] "Django's Cross Site Request Forgery protection." <https://docs.djangoproject.com/en/2.0/ref/csrf/>, accessed: 2018-05-31.
- [40] "Students@DETI website." <http://studentsandteachersdeti.web.ua.pt/>, accessed: 2018-07-09.