This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n: 761999

# EasyTV: Easing the access of Europeans with disabilities to converging media and content.

# D5.8 Final report on the set up and implementation of the EasyTV Service Registry and Catalogue

**EasyTV Project**

*H2020. ICT-19-2017 Media and content convergence. – IA Innovation action.*

**Grant Agreement n°: 761999**

Start date of project: 1 Oct. 2017

Duration: 33 months

Document. ref.: D5.8

## Disclaimer



The EasyTV Consortium consists of the following partners:

| | Partner Name | Short name | Country |
|---|---|---|---|
| 1 | Universidad Politécnica de Madrid | UPM | ES |
| 2 | Engineering Ingegneria Informatica S.P.A. | ENG | IT |
| 3 | Centre for Research and Technology Hellas/Information Technologies Institute | CERTH | GR |
| 4 | Mediavoice SRL | MV | IT |
| 5 | Universitat Autònoma Barcelona | UAB | ES |
| 6 | Corporació Catalana de Mitjans Audiovisuals SA | CCMA | ES |
| 7 | ARX.NET SA | ARX | GR |
| 8 | Fundación Confederación Nacional Sordos España para la supresión de barreras de comunicación | FCNSE | ES |
| 9 | Unione Italiana dei ciechi e degli ipovedenti | UICI | IT |

| | |
|---|---|
| **PROGRAMME NAME:** | H2020. ICT-19-2017 Media and content convergence - IA Innovation action |
| **PROJECT NUMBER:** | 761999 |
| **PROJECT TITLE:** | EASYTV |
| **RESPONSIBLE UNIT:** | ENG |
| **INVOLVED UNITS:** | ENG, UPM |
| **DOCUMENT NUMBER:** | D5.8 |
| **DOCUMENT TITLE:** | Final report on the set up and implementation of the EasyTV Service Registry and Catalogue |
| **WORK-PACKAGE:** | WP5 |
| **DELIVERABLE TYPE:** | Report |
| **CONTRACTUAL DATE OF DELIVERY:** | 31-12-2019 |
| **LAST UPDATE:** | 09-01-2020 |
| **DISTRIBUTION LEVEL:** | PU |

**Distribution level:**

**PU** = *Public,*

**RE** = *Restricted to a group of the specified Consortium,*

**PP** = *Restricted to other program participants (including Commission Services),*

**CO** = *Confidential, only for members of the LASIE Consortium (including the Commission Services)*

# Document History

| VERSION | DATE | STATUS | AUTHORS, REVIEWER | DESCRIPTION |
|---|---|---|---|---|
| v. 0.1 | 02/12/2019 | Draft | ENG | Table of Contents definition and document structure |
| v. 0.2 | 05/12/2019 | Draft | ENG | Introduction and first Service Registry draft |
| v. 1.0 | 18/12/2019 | Draft | ENG | First complete draft |
| v. 1.1 | 19/12/2019 | Ready for internal review | ENG | First version ready for review |
| v. 1.2 | 27/12/2019 | Review | MV | Internal review |
| v. 1.3 | 07/01/2020 | Review | CERTH | Internal review |
| v. 1.4 | 08/01/2020 | Review | ENG | Minor changes |
| v. 1.5 | 09/01/2020 | Final | ENG | Minor changes |

# Definitions, Acronyms and Abbreviations

| ACRONYMS / ABBREVIATIONS | DESCRIPTION |
|---|---|
| API | Application Programming Interface |
| CA | Certification Authority |
| CaaS | Container As A Service |
| CMS | Content Management System |
| CSS | Cascading Style Sheets |
| EU | European Union |
| FQDN | Fully Qualified Domain Name |
| HTML | HyperText Markup Language |
| HTTP | HyperText Transfer Protocol |
| HTTPS | Secure Hypertext Transfer Protocol |
| JS | JavaScript |
| JSON | Javascript Object Notation |
| NGO | Non-Governmental Organization |
| REST | Representational State Transfer |
| SSL | Secure Socket Layer |
| TLS | Transport Layer Security |
| UAAG | User Agent Accessibility Guidelines |
| UI | User Interface |
| UX | User Experience |
| W3C | World Wide Web Consortium |
| WAI | Web Accessibility Initiative |
| WCAG | Web Content Accessibility Guidelines |
| WP | Work Package |
| YAML | Yet Another Markup Language |

# Table of Contents

# List of Figures

# 1. EXECUTIVE SUMMARY

The aim of the present document is to report the activities and the outcomes of *T5.3 - EasyTV Service Registry and Catalogue* from the start of the project until M27. This deliverable is based on *D5.3 - Mid-term report on the set up and implementation of the EasyTV Service Registry and Catalogue*, which is considered a starting point.

Two assets, which are the output of the activities conducted during the period, are described as follows:

a) the EasyTV Service Registry: a Container as a Service (CaaS) environment to collect the EasyTV services "packaged" in the form of "images". This CaaS environment will enable external developers to build applications in a self-service manner and select from image content approved for use by the IT Operation team. External developers can then use these images to create new applications, quickly and securely.

b) the EasyTV Service Catalogue: a web-based catalogue where professional users can choose the services of their own interest among those running on the EasyTV platform.

Consistently, the deliverable has been divided into two different sections:

1. **EasyTV Service Registry**
    a. Introduction
    b. Service registry choices and motivations
    c. Use cases
    d. Architecture description
    e. Installation and configuration of the EasyTV Service Registry within the platform
    f. Image storage and backup
    g. Usage

2. **EasyTV Service Catalogue**
    a. Introduction
    b. Definition of the final user type
    c. UI design methodology
    d. UI prototyping and design
    e. Choice and configuration of the backend CMS
    f. Front-end and backend development and integration
    g. Final deployment

# 2. INTRODUCTION

The EasyTV Service Registry and the EasyTV Service Catalogue are both parts of the EasyTV core components together with the multi-terminal technical platform and the Service Development Kit.

The EasyTV Service Registry provides a way to methodically save, update and read software images and it is useful to quickly design, implement and deploy a new service by using an existing image module as a baseline. It is mainly targeted to developers and technical users.

The EasyTV Service Catalogue, on the other hand, brings information about the platform services available. A detailed description of each service available is offered, together with a technical description which enables further integration with external 3rd parties. It is mainly targeted to professional users (e.g. professional subtitlers, content producers) and software developers.

## 2.1. Purpose and scope

The report D5.8 is part of Work Package (WP) 5 aimed to achieve the integration of the four pillars that EasyTV component-based system is going to be built on:

- a **multi-terminal technical platform** which is a platform for the deployment and the optimization of the EasyTV services needed to make broadcaster contents "accessible" to users with disabilities;
- group of **platform-based service components** which are the outputs of WP2, WP3 and WP4;
- **service registry and catalogue** which allows a user to choose the service that fits her/his needs;
- a **service development kit** which allows third-party developers/"external" software companies to include brand "new" services in the EasyTV multi-terminal technical platform[1].

Specifically, the document is part of *Task 5.3 - EasyTV Service Registry and Catalogue* that includes all activities conducted in the project for the set up and the implementation of two main assets of the EasyTV system:

- Service Registry
- Service Catalogue

1.

The present “Final report” describes the activities carried out and the technical choices made to obtain the two assets in their final architecture and functionalities.

## 2.2. Relation to other tasks

This document should be considered in direct conjunction with other deliverables:

- it considers evidences from WPs’ tasks and deliverables so far achieved, especially tasks related to the EasyTV system’s requirements and architecture as:
    - WP1 - Requirements, specification and technical architecture
        - Task 1.1 – End user requirements gathering;
        - Task 1.2 – EasyTV system requirements specification;
        - Task 1.3 - Technical architecture development;

- It is strictly related to the development of all the other tasks of the WP5
    - Task 5.1 – Multi terminal technical platform to operate the EasyTV services;
    - Task 5.2 – Creation of the multilingual crowdsourcing sign language platform and repository;
    - Task 5.4 - EasyTV Service Development Kit;

    - Task 5.5 - Integration and technical testing.

- It is considered the continuation of *D5.3 Mid-term report on the set up and implementation of the EasyTV Service Registry and Catalogue*.

# 3. EASYTV SERVICE REGISTRY

## 3.1. Introduction

The EasyTV Service Registry is a CaaS (*Container as a Service*) environment built to collect the EasyTV services "packaged" in the form of software images. This environment should also enable external developers to build applications in a "self-service" manner by using the existing set of loaded images as a baseline to develop new modules/functionalities. In order to ease the user interaction and to expand the set of available use cases a web frontend application has been developed and integrated into the Registry environment, thus allowing an authenticated user to quickly get a list of available images and to gather further image details.

## 3.2. Service registry motivations and choice

According to the main role of the service registries, one of the main reasons to deploy a private registry on the EasyTV platform is the opportunity to have full control on where the software images are being stored. In this way, no external provider (hosted registry) is needed to manage multiple versions of the EasyTV service images and, as a matter of fact, privacy is increased because all the repositories are stored inside the EasyTV platform.

Another reason to choose a private registry is the fact that it is possible to customise the authentication and authorisation steps. An authentication middleware has been developed and deployed, thus offering the chance to personalise it if necessary.

These two issues are the main reasons to include a Service Registry in the platform. Moreover, It has been decided to use Docker Registry [1], which is the on-premise version of Docker Hub [2] because it obviously has full integration with Docker, but also because it is open source, has a big developer community and is offered with complete image repositories, automated builds and web hooks.

Other motivations to choose this kind of technology are:

- Get full control of where the images are being stored
- Fully ownership of the image's distribution pipeline
- Integration of the image storage and distribution tightly into self-in-house development workflow
- Get as many private repositories as wanted without increasing the price of the system.

## 3.3. Use cases

Use cases have been systematically defined and analysed: the results are presented here to have a clear view of the environment in its final form and to outline the available features and functionalities. It is noteworthy to mention that the typical user is a "technical" user (e.g. software analyst/developer).

The following use cases have been identified and analysed:

- View available images
    1. The user visits https://registry.easytv.eng.it.
    2. The system shows the authentication form.
    3. The user performs the authentication step.
    4. The system shows a list of all the available images.
- View image detail
    1. The user visits https://registry.easytv.eng.it.
    2. The system shows the authentication form.
    3. The user performs the authentication step.
    4. The system shows a list of all the available images.

5. The user selects the image of his own interest.
6. The system shows the detail of the selected image: details include creation date, image size, tag, OS version and CPU architecture, environment variables, exposed ports, layers' description.

- Load (push) a new image
  1. The user performs the login through the "*docker login*" command.
  2. (Optional step) The user tags the image to push through the "*docker tag"* command, thus ensuring that a specific image will be pushed onto the EasyTV Service Registry.
  3. The user loads (pushes) a new image through the "*docker push"* command, stating the name and tag of the image to push.
  4. The system notifies the user when the image has been successfully loaded.
- Update an existing image
  1. The user performs the login through the "*docker login"* command.
  2. The user updates the image through the "*docker push"* command, indicating the name and tag of the image to update.
  3. The system notifies the user when the image has been successfully updated.
- Download (pull) an existing image
  1. The user performs the login through the "*docker login*" command.
  2. The user downloads (pulls) an existing image through the *"docker pull*" command, indicating the image name and tag.
  3. The system notifies the user when the image has been successfully downloaded.

It is worth to mention that no use case related to image deletion exists, since it is advisable to keep all the existing images accessible and non-erasable because 3rd party software may rely on them and there is a potential risk of breaking dependencies if an image is not found. The alternative is to simply push a new image with a different tag, so no image is erased. Of course, IT operations team members (someone who has direct access to the server) can always erase images if such a requirement arises, but deletion is never achievable through the EasyTV Service Registry application.

The following diagram offers a view at a glance of the aforementioned use cases.

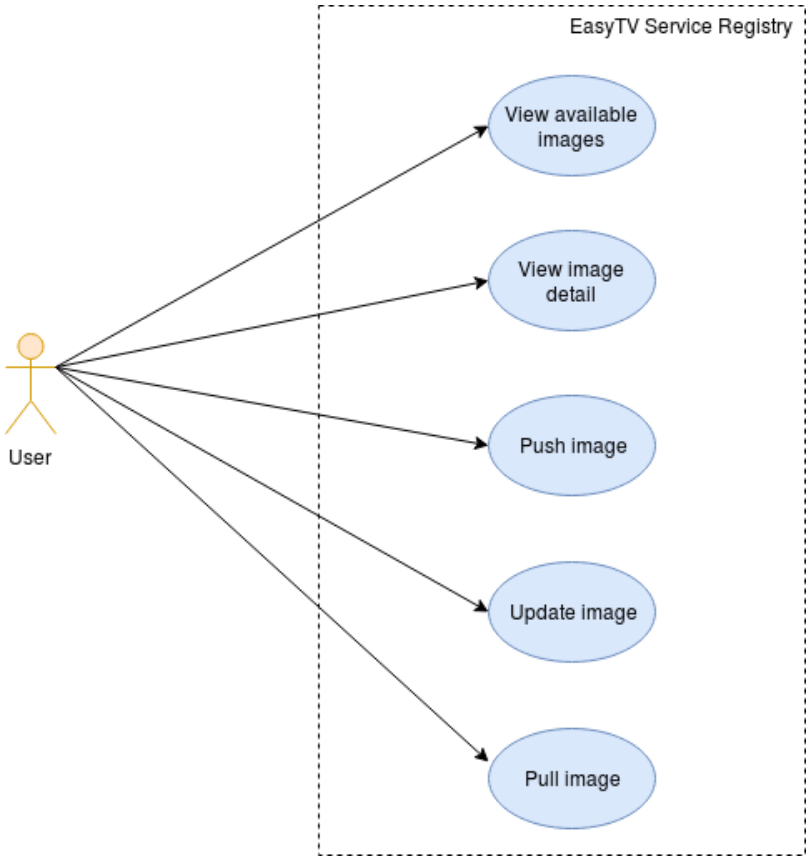


**Figure 1 - EasyTV Service Registry use cases diagram**

## 3.4. Architecture description

The chosen software architecture allows a certain grade of flexibility in terms of deployment and configuration without sacrificing basic security aspects.

The preferred solution employs a module which acts as a proxy and a custom web frontend with authentication to show all the available docker images and the related image details. All of the Service Registry components run as services inside a specific Docker stack on the server platform, thus benefiting from the available Docker and Docker Swarm features such as container orchestration and isolation.

The web frontend is based on Riot [3], a lightweight component-based UI library, and it connects to the Docker Registry running as a service. The authentication middleware, previously described in D5.3, is now integrated into the new proxy service. User permissions are still managed through the *htpasswd* utility, which can be used to authorise users access to the EasyTV Service Registry: its use, of course, is geared towards the IT operations team, since it is critical to avoid potential misuse of the Registry by unauthorised external users.

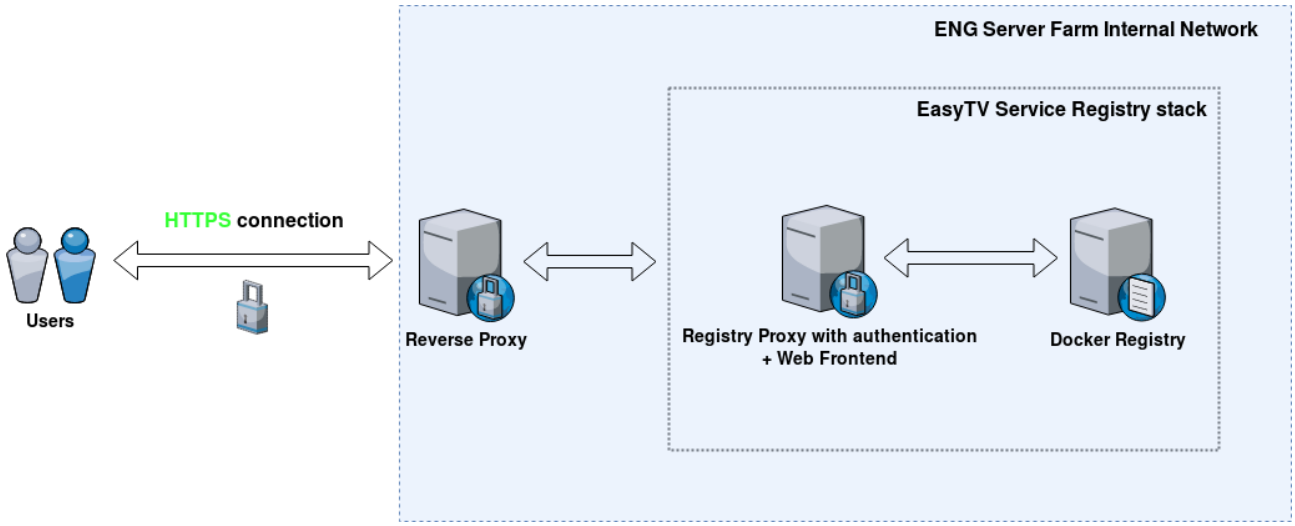


Figure 2 - EasyTV Service Registry network architecture

## 3.5. Installation and configuration

### 3.5.1. Reverse proxy and virtual hosting

The reverse proxy (Apache-based [4]) which sits on the boundaries of ENG network has been properly configured in order to make the Service Registry reachable at the following address: https://registry.easytv.eng.it. SSL/TLS certificates are supplied by the CA "Let's Encrypt", which offers DV (domain validated) certificates, the most common type of SSL/TLS certificates.

The following snippet illustrates the virtual host configuration and the reverse proxy configuration for the EasyTV Service Registry:

```
<VirtualHost *:443>

    ServerName registry.easytv.eng.it

    SSLEngine On
    SSLCertificateFile /etc/letsencrypt/live/registry.easytv.eng.it/cert.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/registry.easytv.eng.it/privkey.pem
    SSLCertificateChainFile /etc/letsencrypt/live/registry.easytv.eng.it /chain.pem

    ProxyPreserveHost on
    ProxyPass / http://127.0.0.1:5000/
    ProxyPassReverse / http://127.0.0.1:5000/

    <Location />
        Order deny, allow
        Allow from all
    </Location>

</VirtualHost>
```

### 3.5.2. **Authentication middleware**

The authentication middleware has been integrated into the Registry proxy component, as shown in Figure 2. Even though a node.js runtime with the node-http-proxy [5] module has been initially used to implement the middleware functionalities, the final choice was to integrate a customised nginx [6] version to leverage its performance and security features. The following configuration options inside the custom nginx configuration enable HTTP Basic Authentication:

```
[...]
auth_basic "Registry realm";
auth_basic_user_file /etc/nginx/conf.d/nginx.htpasswd;
[...]
```

The *htpasswd* [7] utility is used to manage the user file (or multiple user files) for basic authentication and is employed by the authentication middleware to check whether a user is authorised to perform certain operations. An administrator can easily add a new user by entering the following command:

```
$ htpasswd -c  nginx.htpasswd registryuser1
```

A password prompt then appears, asking to supply a password for “registryuser1” user. The output “nginx.htpasswd” file is then read by the middleware whenever an authentication operation must be performed.

### 3.5.3. **Frontend**

A web frontend has been added to the application stack in order for a user to have a complete view of the available images loaded on the Service Registry, and to gather additional details for each image shown. Many existing Docker Registry frontend solutions have been investigated, including (but not limited to) the following ones:

- http://port.us.org/
- https://github.com/kwk/docker-registry-frontend
- https://github.com/genuinetools/reg
- https://joxit.dev/docker-registry-ui/
- https://github.com/brennerm/docker-registry-frontend

In the end a custom solution has been developed, starting from Joxit’s *Docker Registry UI* project. This web interface uses Riot, the react-like user interface micro-library, and riot-mui components. It connects to the registry running as a docker service in the background and it issues the proper commands in order to get the needed information that are subsequently displayed on a web page.

Among the various features offered it is worth mentioning:

- List of all images
- List of all tags for a specific image
- Sort image by tag number
- Copy “*docker pull*” command to clipboard
- Secure Docker Registry support
- Image details: creation date, size, tag
- Additional image details (e.g. id, os, architecture) and complete layer creation history

The following screenshot shows some of the “history” detail of an image:

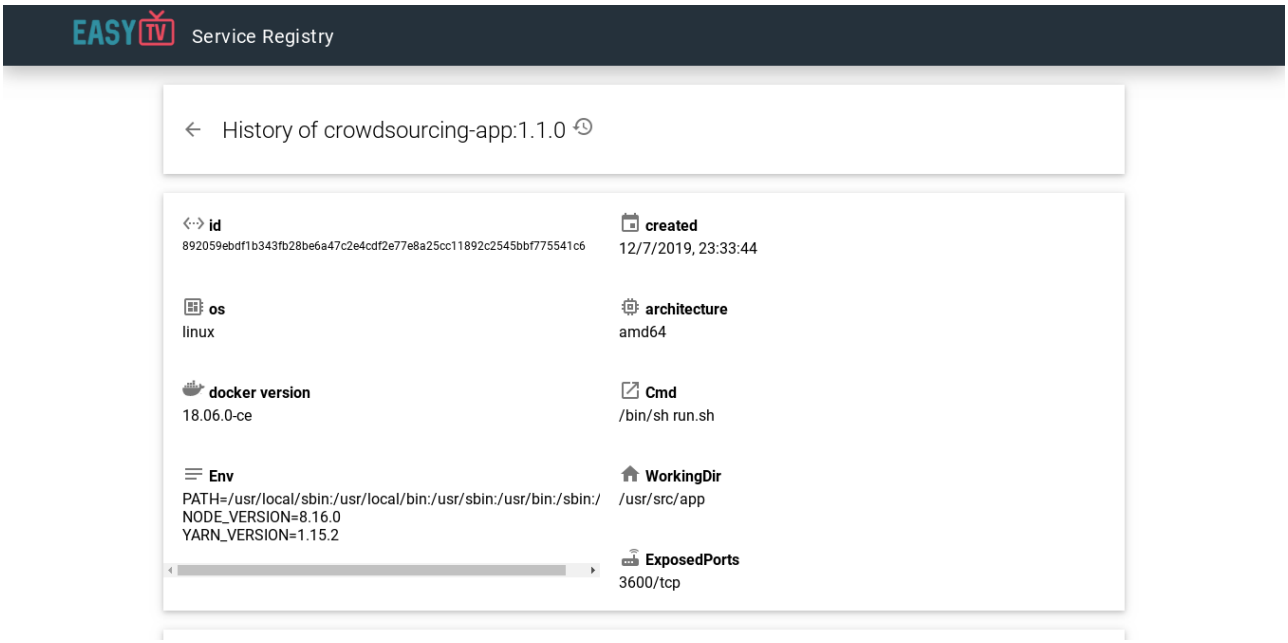


**Figure 3 - Image history details**

### 3.5.4. **Stack deployment**

The EasyTV Service Registry can be run in a *containerised* mode on the Docker platform, so it makes sense to deploy it as a *docker stack* of multiple component. Specifically, the final solution takes the form of a stack with two services: a Docker Registry and a web frontend plus an authentication middleware. The following YAML file has been used to deploy the running solution:

```
version: '3.0'
services:
  registry:
    image: registry:2.6.2
    volumes:
      - /dati/registry:/var/lib/registry
    networks:
      - registry_net

  ui:
    image: joxit/docker-registry-ui:custom
    volumes:
      - ./registry-config/htpasswd:/etc/nginx/conf.d/nginx.htpasswd
    ports:
      - 5000:80
    environment:
      - REGISTRY_TITLE=EasyTV Service Registry
      - REGISTRY_URL=http://registry:5000
    depends_on:
      - registry
```

```
    networks:
      - registry_net


networks:
  registry_net:
    external: true
```

Of course, all services communicate on the same Docker virtual network (which is an overlay network type named “registry_net”).

### 3.5.5. **Portainer integration**

Portainer [8], the Docker management UI described in D5.1, can be configured to use the EasyTV Service Registry in order to get the desired images to be executed as containers.

In order to add a new registry to Portainer, log in as an administrator and go to “Registries” on the main menu. Finally, click on “Add registry” and fill in the needed fields (name, URL, username and password). The new registry will be added and it will be ready to be used.

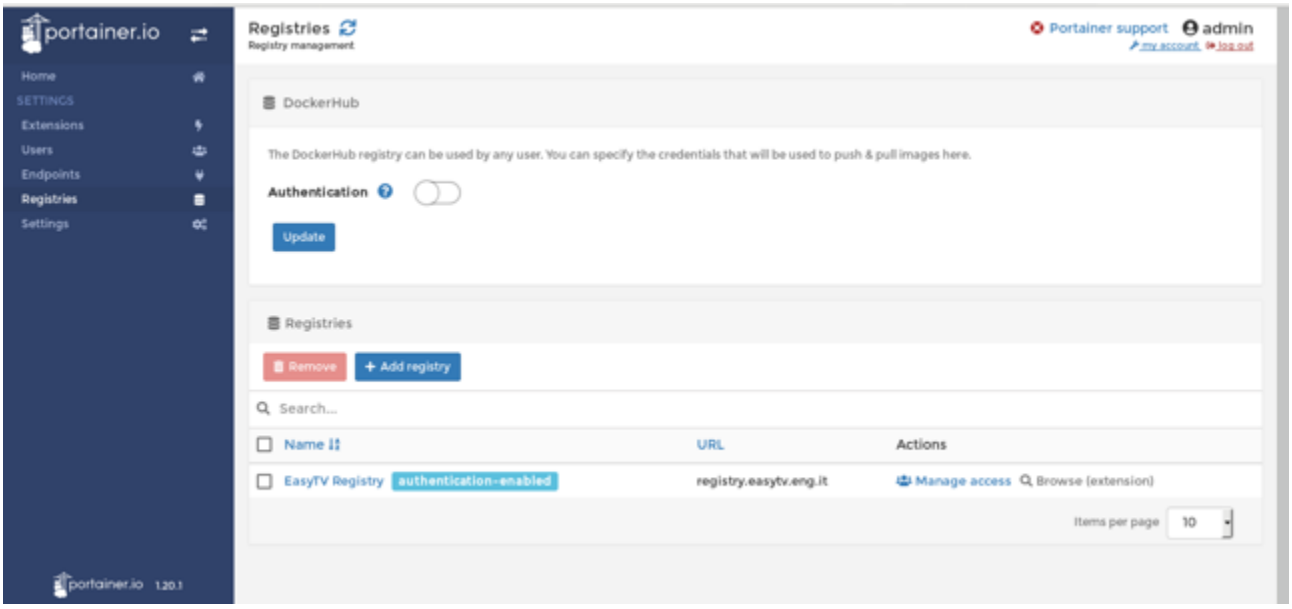


**Figure 4 - Registry added in Portainer**

To pull an image from the registry, go to “Images” on the main menu, select the new configured registry and write the image name and tag. Finally, click on “Pull the image” to start the image pull.

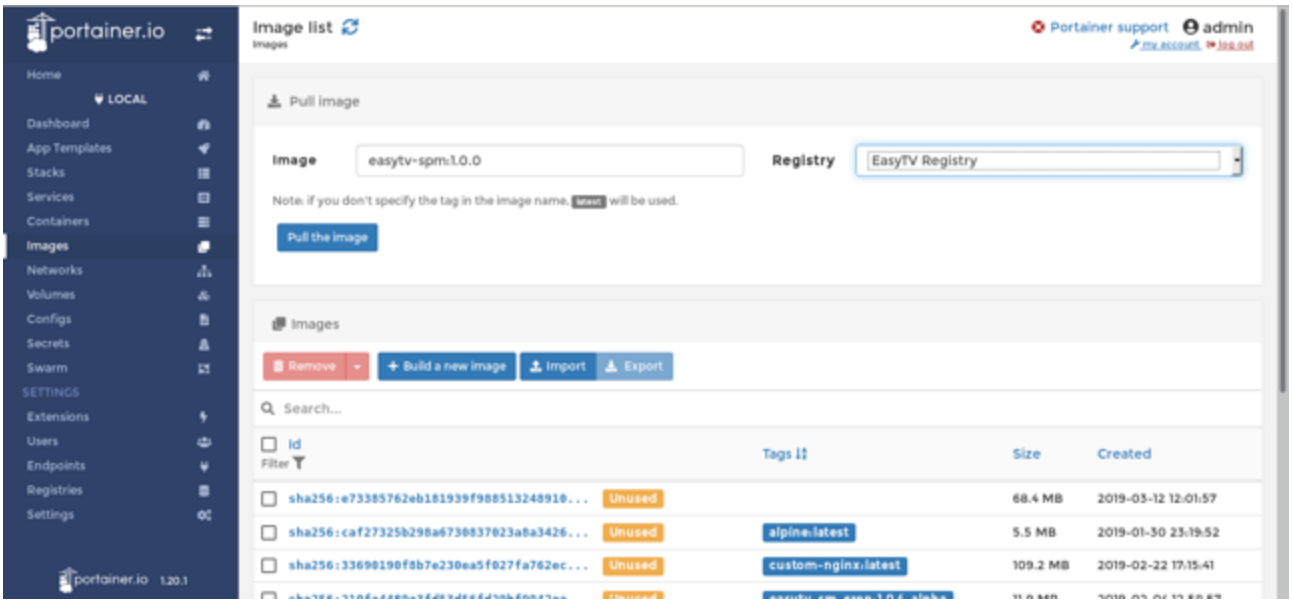


Figure 5 - Registry selection in Portainer

## 3.6. Image storage and backup

Data is persisted on the server by using a *bind mount* [9] technique: in this way a directory is used to store all the data independently of the docker service lifecycle. A backup system has been set up, which is in charge of periodically making a backup copy of the folder that contains all of the service images. Once the backup copy is created, it is automatically moved onto a secure location.

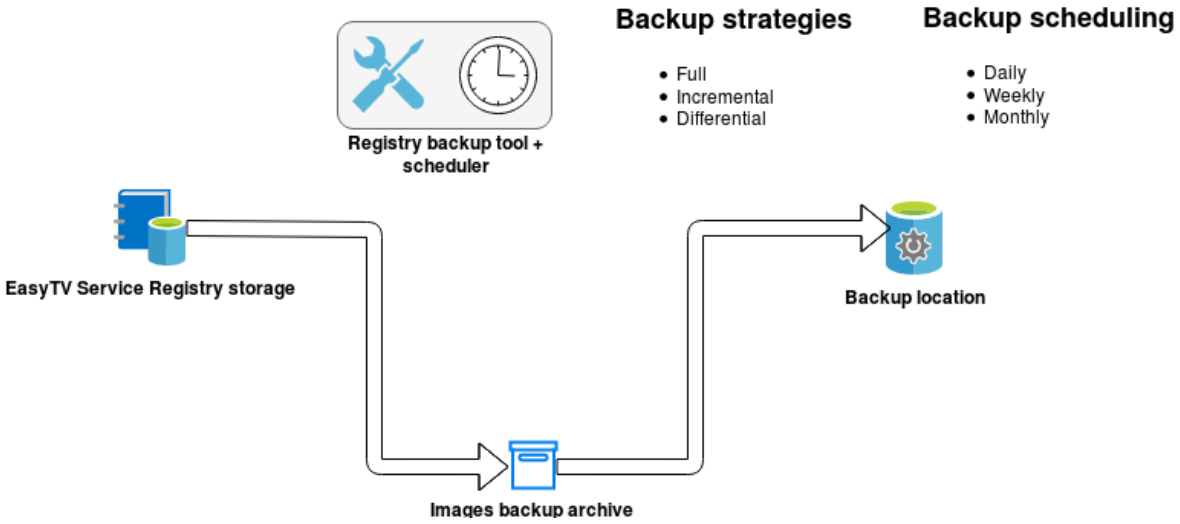


Figure 6 - Backup flow

The backup tool can be configured to perform the backup at different times and with different strategies. Typically, a backup should be performed daily, weekly or monthly. The three most common strategies are briefly described:

- Full backup: a complete copy of the source dataset is created.
- Incremental backup: only data that has changed since the latest backup job is considered.
- Differential backup: data that has changed since the last full backup is considered.

In case of need, a system administrator / IT operations team member can restore the folder which contains the service images by choosing a backup copy among the existing backup copies.

## 3.7. Usage

### 3.7.1. Docker CLI

It is clear from the use cases description that a user can interact with the EasyTV Service Registry both with the web frontend interface and the Docker CLI. The latter provides the only way to actually "*push*" and "*pull*" images to/from the registry. A complete flow is detailed here, highlighting the commands used.

The first operation to perform is a login, which has to be performed only once for each registry:

```
$ docker login registry.easytv.eng.it
```

Once the user has been authorised, it is possible to push a new image (suppose the image has been already tagged properly with registry name and version):

```
$ docker push registry.easytv.eng.it/easytv_s1:latest
```

If the user wishes to use an image as a baseline for the development of a new service, it is possible for him/her to pull an already available image from the registry:

```
$ docker pull registry.easytv.eng.it/easytv_img_A:latest
```

On the other side, if a user has downloaded an image from another registry, it is possible to tag this particular image to ensure it will be pushed onto the EasyTV Service Registry. As explained in (https://docs.docker.com/engine/reference/commandline/tag/), to push an image to a private registry (and not the central Docker registry) a developer must tag it with the registry hostname. The tag command is used to accomplish this (followed by a push command):

```
$ docker tag easytv_s2 registry.easytv.eng.it/easytv_s2:latest
$ docker push registry.easytv.eng.it/easytv_s2:latest
```

### 3.7.2. HTTP API v2

The Docker Registry HTTP API [10] is "the protocol to facilitate distribution of images to the docker engine". It interacts with instances of the Docker Registry and it is useful because it enables the development of custom registry clients.

Some preliminary clarifications are needed. An image is a combination of a JSON manifest and individual layer files and the process of pulling an image involves retrieving these two components. Pushing, instead, works in the opposite order as pull: first of all, the layers are pushed into the registry, then the manifest is uploaded to the registry to complete the process.

Only the most important operations will be covered here, namely **pull**, **push**, **list** and **delete**. Other details can be found in [10].

**Pull an image**

First the manifest needs to be fetched with the following url:

```
GET /v2/<name>/manifests/<reference>
```

Where name and reference are two parameters which identify the image and a tag or digest respectively. Note that the manifest will contain multiple digests associated with each image layer.

Then, it is possible to get the layers with the following url:

```
GET /v2/<name>/blobs/<digest>
```

**Push an image**

To start an image uploading, a POST request is used:

```
POST /v2/<name>/blobs/uploads/
```

Once all the layers of an image are pushed the client can provide the image manifest:

```
PUT /v2/<name>/manifests/<reference>
```

**List repositories and image tags**

Images are stored in collections (repositories) identified by a name. To get the catalogue (that is, the list of available repositories within a registry) the following endpoint is used:

```
GET /v2/_catalog
```

Then, for each repository, it is possible to list all the available tags:

```
GET /v2/<name>/tags/list
```

**Delete an image**

The following endpoint can be used to delete an image from the registry:

```
DELETE /v2/<name>/manifests/<reference>
```

The <reference> parameter must be a digest.

# 4. EASY TV SERVICE CATALOGUE

## 4.1. Introduction

The EasyTV Service Catalogue is a web-based catalogue where a professional user (see 4.2 for the definition of “professional” user) can choose the services of her/his own interest among those running in the EasyTV multi terminal technical platform.

The activities conducted to obtain the EasyTV Service Catalogue asset in its final form span from requirement gathering to the actual implementation tasks. Specifically:

- User type definition
- Design methodology (UI related)
- Development approach definition
- Prototyping and UI design
- Frontend & headless CMS (backend) development and configuration
- Integration & final deployment

## 4.2. User type definition

One of the main points during the design phase of the Service Catalogue has been the correct identification and definition of the users which interact with the Service Catalogue. In the end, a set of users labeled “professional users” have been proposed: a professional user can be either a content producer, an expert user with a high level of expertise in sign language and/or proficiency in translation activities (e.g. a professional subtitler), or a technical user (e.g. a software architect or a software developer).

It is clear from the above definition that the Service Catalogue does not target a final user with visual or hearing impairments, but aims at a final user which is regarded as “professional” in the sense previously explained.

## 4.3. Design methodology

An overview of global design trends has been conducted in order to evaluate the proper design approach and/or methodology, taking into account the main EasyTV system requirements.

### 4.3.1. Design Trend

With the increasing complexity of society, caused by multiple factors (including globalization, migration, sustainability), traditional design methods have become insufficient.

Influence of systems theory, united with the need to consider heterogeneous groups of persons/users (grouped according to different criteria) and not the individual, have led to a “Systemic” approach.

The strategic role of Systems Thinking has been underlined and explained in [11], which explores design as inquiry for action.

The **Systemic design** is intended for a recent initiative in design that aims to face challenges characterised by complexity, uniqueness, value conflict, and ambiguity over objectives. It proposes to integrate Systems Thinking and Human-Centered Design with the intention of helping designers cope with complex design projects.

Systemic design is an approach that supports questions related to the interdependence between products, services, processes and policies that have social repercussions to be addressed.

Following this perspective, the design process becomes both systemic (integrative and

interconnected) and systematic (methodical)[2]. It allows different teams to develop an elevated perspective of the challenge and translate novel insights into rapid action.

All systemic design is provisional and open to redesign and projects carried out with this approach are entangled with social systems.

Systems Thinking leads to modelling and intervening in the world as if it is composed of open, purposeful and complex wholes. As introduced, key aspects of systemic approach are its inquiring, open, integrative, collaborative and interdependence core. Reciprocal influence between parts of a greater whole and their environment are taken into account: interdependencies between system components and their environment give rise to self-organisation, learning, adaptation, evolution, complexity that is managed.

Systemic Design approach is covering a key role in Digital Transformation process, developing methodologies and approaches that help to integrate systems thinking with design towards sustainability at environmental, social and economic level.

In the last years, specifically in the web design community, one of the most operative application of the systemic approach in design practice and activities is the adoption of a **design system**.

### 4.3.2. **Design System**

As defined by Nathan Curtis “almost always, a design system offers a library of visual style and components documented and released as reusable code for developers and/or tool(s) for designers. A system may also offer guidance on accessibility, page layout and editorial and less often branding, data viz, UX patterns and other tools”[3].

Accordingly to the User Centred Design principles, a design system is a set of standards for design and code along with components that unify both practices, becoming a documentation and a modular toolkit for designers and developers. Engineering teams are able to refer to a single source at the implementation phase and this impacts inevitably in a positive way system consistency (i.e. all call-to-actions look the same across screens).

[2] Harold G. Nelson and Erik Stolterman, "The Design Way: Intentional Change in an Unpredictable World, Second Edition"

[3] Nathan Curtis “Defining Design Systems.Getting to the Root of What Your System Really Is” (2017) https://medium.com/eightshapes-llc/defining-design-systems-6dd4b03e0ff6

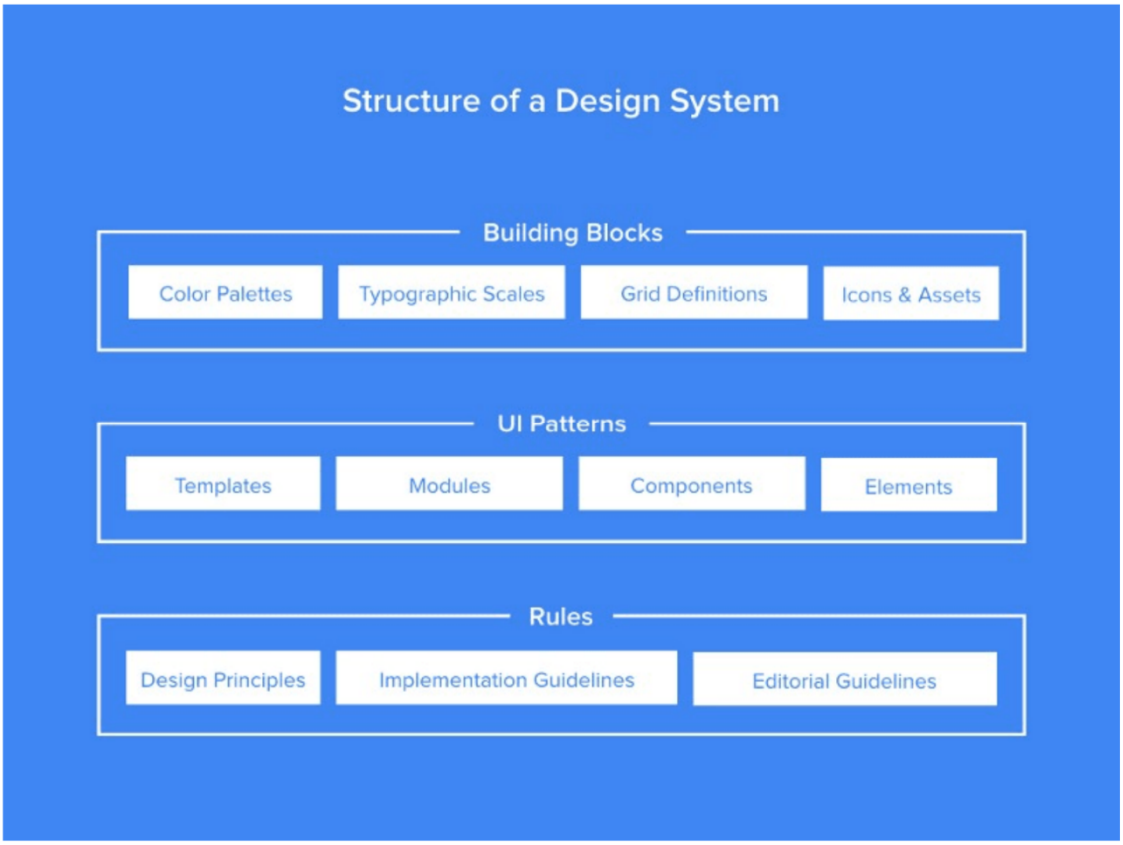


**Figure 7 - Example of Design System Structure by UX Pin**

A design system is not conceived as a deliverable, but as a set of deliverables and they evolve constantly with the products, tools and new technologies. Design systems are flexible, adapting naturally to changes in the product and synchronizing design and code, for an easier way to create consistent experiences.

One of the strengths of this design solution is its scalability: a design system is a collection of reusable components that can be assembled together to build any number of interfaces, following a set of standards guiding the use of those components.

Futhermore, components of a design system are adaptable in terms of "interfaces-crossing": they allow to be adopted regardless of interface sizes, device and operative system environments used.

The following design systems represent an example of the widely diffusion of this approach to design:

- *Commercial Design Systems*: Material[4] from Google, Fluent[5] from Microsoft
- *Government Design Systems*: US Government Design Standards[6]

Considering the potential opportunities offered by a system design, that allow designing systems of components in a very flixible, scalable, adaptive and consistent way, an important aspect to evaluate, in order to achieve the purposes of the present deliverable, is the identification of a robust methodology to adopt.

In 2013 Brad Frost introduced the **Atomic Design**[7] as a design system methodology.

[4] https://material.io/guidelines/#

[5] https://fluent.microsoft.com/

[6] https://standards.usa.gov/

The Atomic Design Systems methodology considers all the details that contribute to the creation and maintenance of robust design systems using chemistry metaphor.

The starting point for Frost is the consideration that interfaces are made up of smaller components. His basic conceptual operation consisted of breaking entire interfaces down into fundamental building blocks and work up from there.

Atomic design is composed of five distinct stages working together to create interface design systems in a more deliberate and hierarchical manner.


**Figure 8 - Structural components of Atomic Design Methodology**

There are five distinct levels in atomic design:

- *Atoms*: parts of an interfaces serve as the foundational building blocks that comprise all user interfaces. These atoms include basic HTML elements like form labels, inputs, buttons, and others that can't be broken down any further without ceasing to be functional;
- *Molecules*: are relatively simple groups of UI elements functioning together as a unit (i.e. a form label, search input, and button can join together to create a search form molecule);
- *Organism*: are relatively complex UI components composed of groups of molecules and/or atoms and/or other organisms. These organisms form distinct sections of an interface;
- *Templates*: Templates are page-level objects that place components into a layout and articulate the design's underlying content structure. They focus on the page's underlying content structure rather than the page's final contents. Design systems must account for the dynamic nature of content, so it's very helpful to articulate important properties of components like image sizes and character lengths for headings and text passages;
- *Pages*: are specific instances of templates that show what a UI looks like with real representative contents in place. Pages are essential for testing the effectiveness of the underlying design system.

[7] http://atomicdesign.bradfrost.com/

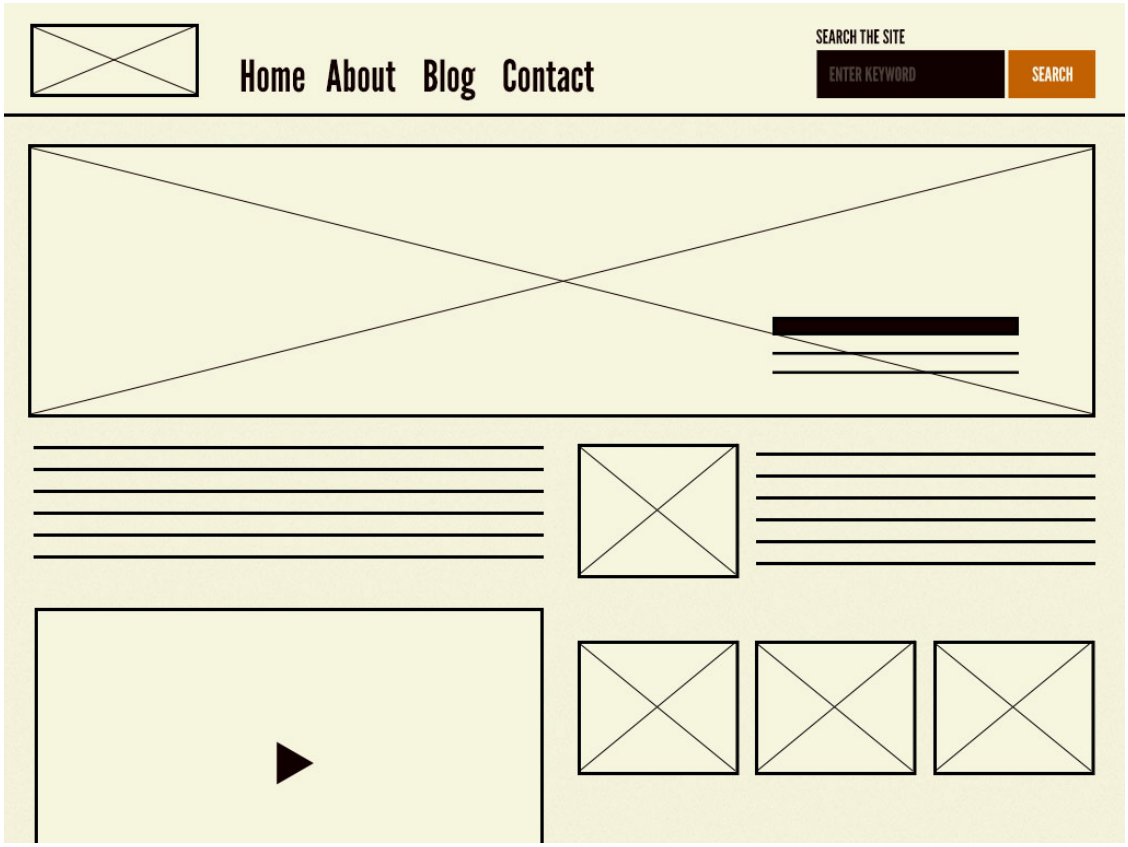


**Figure 9 - Example of web page wireframe composed using Atomic Design Methodology**

As emerged, rather than a linear process, Atomic Design is a mental model considered helpful for constructing a user interface design system as both a cohesive whole and a collection of parts at the same time.

### 4.3.3. An "Accessible Design System" for the EASYTV Catalogue

Considerations resulting from the overview of design trends, combined with consequent focus on design system as a widespread adopted tool to create interfaces, have led the design approach to the set up and implementation of the EasyTV Catalogue.

A design system represents a suitable design solution for the Catalogue for different reasons:

- *Scalability*: Design systems allow to manage design at scale. Its components are modular and can be reused in different combinations, that allow to build different interfaces solutions using the same basic "design blocks" (definable atoms, molecules, organisms, in Atomic Design methodology);
- *Openness*: it allows to be "open" in absence of functional requirements that cannot be defined at this stage. It evolves with the evolution of the project;
- *Consistency*: it contributes to create a consistent branding, look and feel, and User Experience;
- *Efficiency*: it improves efficiency connecting different teamworks of the project and optimizing timing and resources involved in the stages of design and front-end development;

In conclusion, this section on "design methodology" has led to focusing on design system as a design tool that allows to provide a consistent approach for the EasyTV Catalogue set up. Specifically, starting from its "openness", it will be possible, in the next design phases, to overcome any effective technical implementation constraints of the service catalogue.

## 4.4. Development approach

Starting from the previous considerations regarding the design approach, the main characteristics and parameters foreseen for the Catalogue, and in particular for its **interface development**, are the following:

- Accessibility
- Usability
- Responsiveness
- Modularity
- Openness

Interfaces strongly impact the User Experience of the users. For this reason, User Interfaces (UI) design and Front-End development of the Catalogue are both strategic (and interconnected) phases. They allow to achieve the main goals of the whole EasyTV project.

As claimed in the previous paragraph, referring to the design approach of the Service Catalogue, designing an "Accessible Design System" represents a proper solution considering the requirements emerged.

### 4.4.1. Front-end Framework

Website or web application user interface usually consists of:

- HTML code that is responsible for a structure (information order);
- CSS (Cascading Style Sheets) which main task is to visually format website;
- JavaScript (JS) code, used to incorporate dynamic elements, e.g. slideshows, calculators or expanding menu.

To support the creation of the web-based Service Catalogue that includes the properties emerged, the adoption of a **Front-End Framework** can be considered.

Front-End Frameworks are Web Design Kits consisting of a set of ready components, which allow developers to start a project quickly and efficiently considering fundamental aspects as: accessibility, responsiveness, compatibility for different browsers and customization of the user interface.

Moreover, a front-end framework helps to consider other elements involved in the development of a website such as code optimization for better performance, scalability and compliance to standards.

Typically, Front-End Frameworks contain the following components:

- a grid which makes it simple to organize the design elements;
- defined font styles and sizing that varies based on its function (different typography for headings versus paragraphs, etc.);
- Pre-built website components like side panels, buttons, and navigation bars.

Since there are many front-end frameworks available nowadays, an overview on the most popular and widespread Front-End Frameworks has been conducted (based on their GitHub popularity[8]):

- Bootstrap
- Semantic UI
- Foundation
- Materialize
- Material UI
- Pure
- UIkit

[8] https://github.com/search?o=desc&q=stars%3A%3E1&s=stars&type=Repositories

Each framework has its own strengths and weaknesses, and specific areas of application. As emerged from the analysis, **Bootstrap** is currently the most popular front-end component library.

Its main characteristics are:

1. It is an open source toolkit and it's hosted, developed, and maintained on GitHub.
2. Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels, as well as optional JavaScript plugins.
3. The last version of Bootstrap, Bootstrap 4, presents improved responsive features that help to easily and efficiently scale websites and applications with a single code base, from phones to tablets to desktops with CSS media queries.
4. Regarding browser compatibility, Bootstrap 4 is compatible with all modern browsers (Chrome, Firefox, Internet Explorer 10+, Edge, Safari, and Opera).
5. Among the Front-End Frameworks, Bootstrap is the larger in terms of community support and makes available an excellent documentation.

For all these features, Bootstrap 4 (last version of Bootstrap) represents a suitable option for the front-end development of the EasyTV Service Catalogue and as such it has been chosen among the other options.

The choice of a structural and logical javascript-based (or typescript-based) framework has also been considered. There are many frameworks already available, and new ones are continuously developed and published month after month. However, there are some projects which enjoy greater popularity due to their specific features and functionalities, namely:

- Angular
- React
- Vue
- Ember
- Backbone.js

Angular has been chosen as it best fits the needs of the project: it offers a full-fledged MVC framework and many “out-of-the-box” functionalities, such as templates, Ajax requests, routing, forms and much more. Another advantage is the fact that it uses the typescript programming language, thus introducing static type checking and class-based object-oriented programming.

## 4.5. User interface prototyping and design

The Atomic Design Systems methodology described above has been adopted as an approach to realize the design of the EasyTV Service Catalogue and subsequently enable the development stage.

At the beginning of the process that conduct to the design phase (final branded and fully developed product) there are a set of fundamental preliminary activities which constitute the prototyping stage.

To define a system of components for building templates and pages of that services, firstly the relationship between UX (user experience) and UI (User Interface) has been considered. In this perspective it is required to define and evaluate the expected user interaction with the application and their visual elements (components users will see and use on their devices).

Building prototypes allows to test the feasibility and usability of design solutions before starting graphic design activity and begin writing code.

One of the main purposes of a prototype is to test whether or not the interaction with the interface elements and the flow of the product is consistent comparing to the design objectives: allow specified users to achieve certain goals through the specific features of the application, with effectiveness, efficiency and satisfaction.

Therefore, a prototype is a tool for visualizing the interactive design work at different stages and at

various levels of detail:

- *Low-fidelity* prototypes are often paper-based and do not allow user interactions. They range from a series of hand-drawn mock-ups to printouts. In theory, low-fidelity sketches are quicker to create. Low-fidelity prototypes are helpful in enabling early visualization of alternative design solutions, which helps provoke innovation and improvement;
- *High-fidelity* prototypes are computer-based, and usually allow realistic (mouse-keyboard) user interactions. High-fidelity prototypes take as close as possible to a true representation of the user interface. High-fidelity prototypes are assumed to be much more effective in collecting true human performance data (e.g. time to complete a task).

Whether it is static or dynamic, prototypes support designers to detect usability problems since usability challenges suddenly become easy to spot and fix thanks to an iterative process of work.

Between the sketching stage, when ideas and possibilities to pursue emerge, and the design phase, where visual contents and components have their finally graphic rendering, *wireframing* activity allow to define pages structure and the overall flow.

Wireframes are the most popular type of deliverable among UX professionals[9]. They are used to communicate page layouts, information hierarchy, and, in the case of interactive wireframe, interactions (allowing to perform usability tests on). They can use varying levels of fidelity (or similarity to the final product) to communicate different types of ideas at different stages in the process.

A Wireframe page comprises simple symbols, lines, and indicators of interactivity: it is conceptual and suggest a sense of structure and layout, helping to clarify requirements and constraints.

Wireframes fidelity to the final graphic design is determined by the level of interactivity, visual design, and content. There are several tools and techniques that can be used for building wireframe, among the most utilized there are: Adobe XD, Sketch, Framer, Flinto and Axure.

[9] https://www.nngroup.com/articles/common-ux-deliverables/

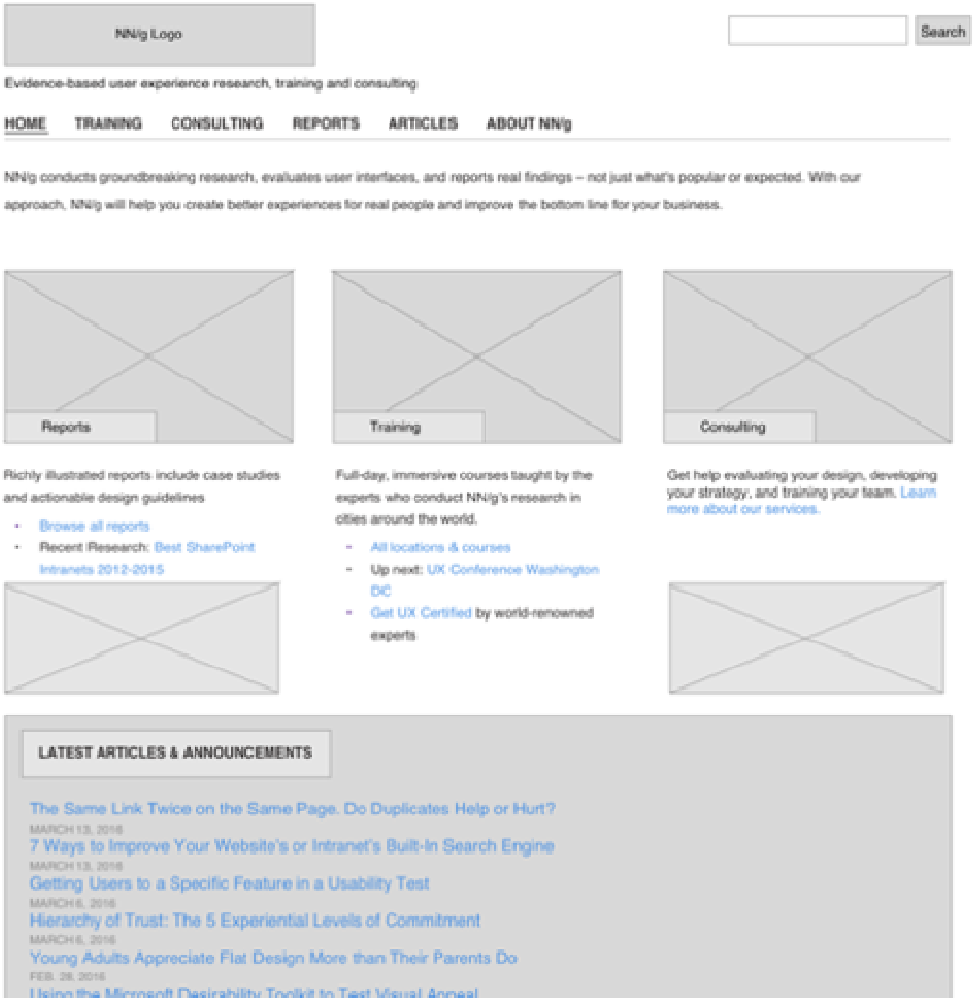


**Figure 10- A wireframe of a web page conveys layout ideas, content, and page-level design for websites and apps (https://www.nngroup.com/articles/wireflows/)**

Since it focuses both on prototyping and design, Adobe XD platform has been selected as the most suitable tool to support the overall design process for the EasyTV Service Catalogue. Consequently, with Adobe XD high–fidelity wireframes and pages with interactions and graphic design elements (UI components) have been built.

### 4.5.1. **Wireframe**

At the following link is available the interactive version of Service Catalogue wireframes: https://xd.adobe.com/view/27158834-bfd3-42aa-4c7d-2d00e4c67fe4-5124/screen/ec86fd28-8c42-44c6-8ff4-dbb27413895a/Service-CatalogueWF-HOME

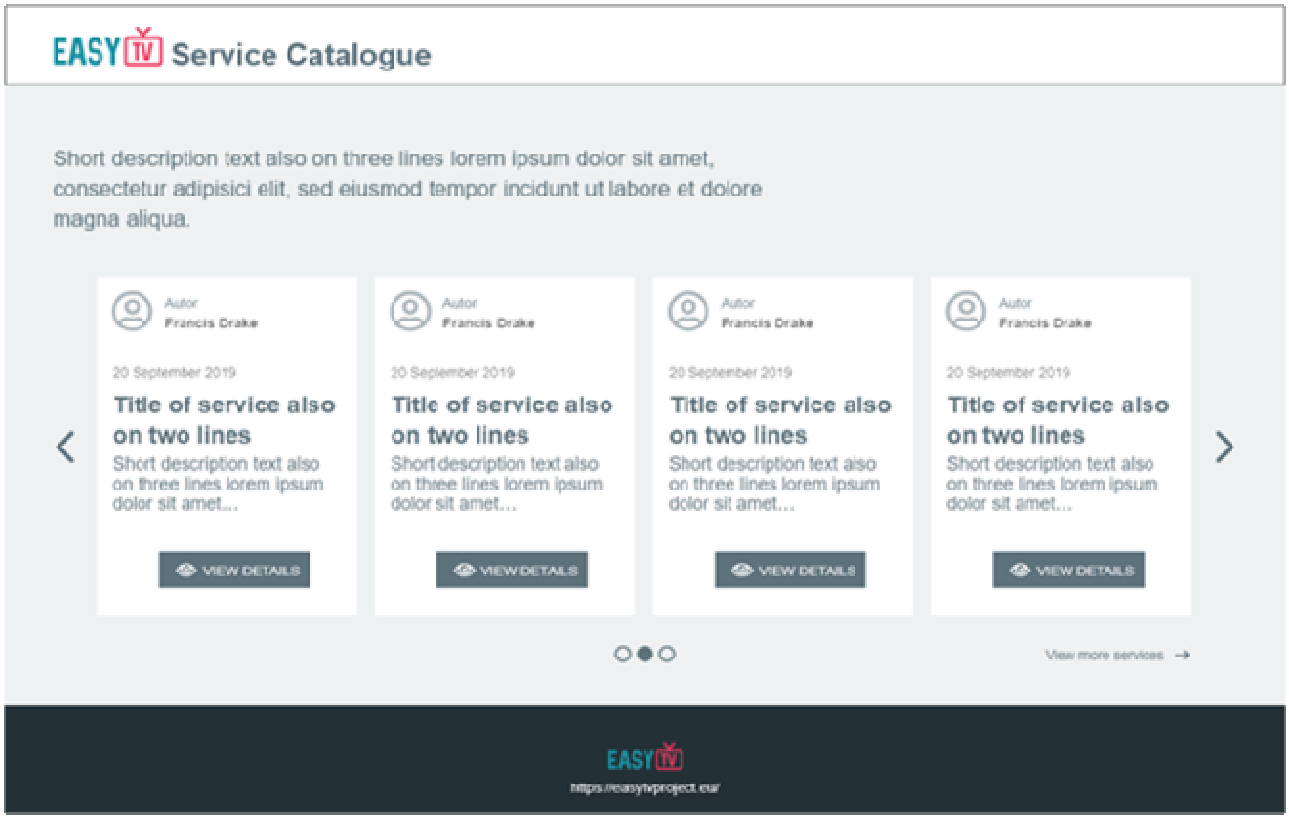


**Figure 11 - Service Catalogue Wireframe: Home page**

1. **Card** component is used when is nice to have a content preview of a section. Elements include multimedia, text, links, graphs, and captions. Cards are a user interface component that acts as an entry point to more detailed information. Varying sources of information are pooled together and presented in a simplified way.

2. Clear and visible **call to action** (CTA) as “View details” lead the user by supporting him in navigation.

3. **Carousels** allow multiple pieces of content to occupy a single, coveted space. In this case through the arrows, user can forward and rewind the carousel.

4. Alternatively, **dot navigation** allows users to immediately see how many pages there are and support users’ interactions making easier to browse the pages.

5. “View more services” **link** leads to a page where all services are available.

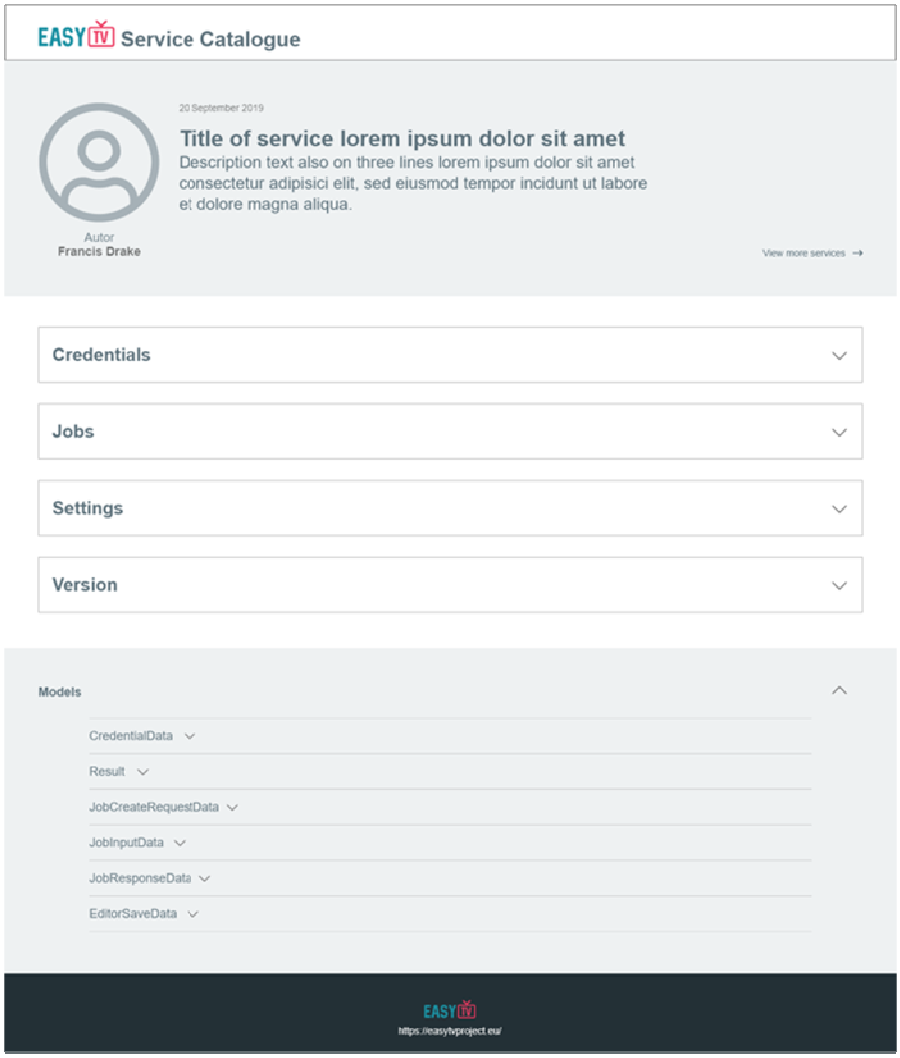


**Figure 12 - Service Catalogue Wireframe: Service detail page**

1. The **product page** shows all services details: service's author and description, and all the specific features.
2. Each **panel** allows to view more detail of the relative section, according to the user's information needs only when requested information will be displayed.

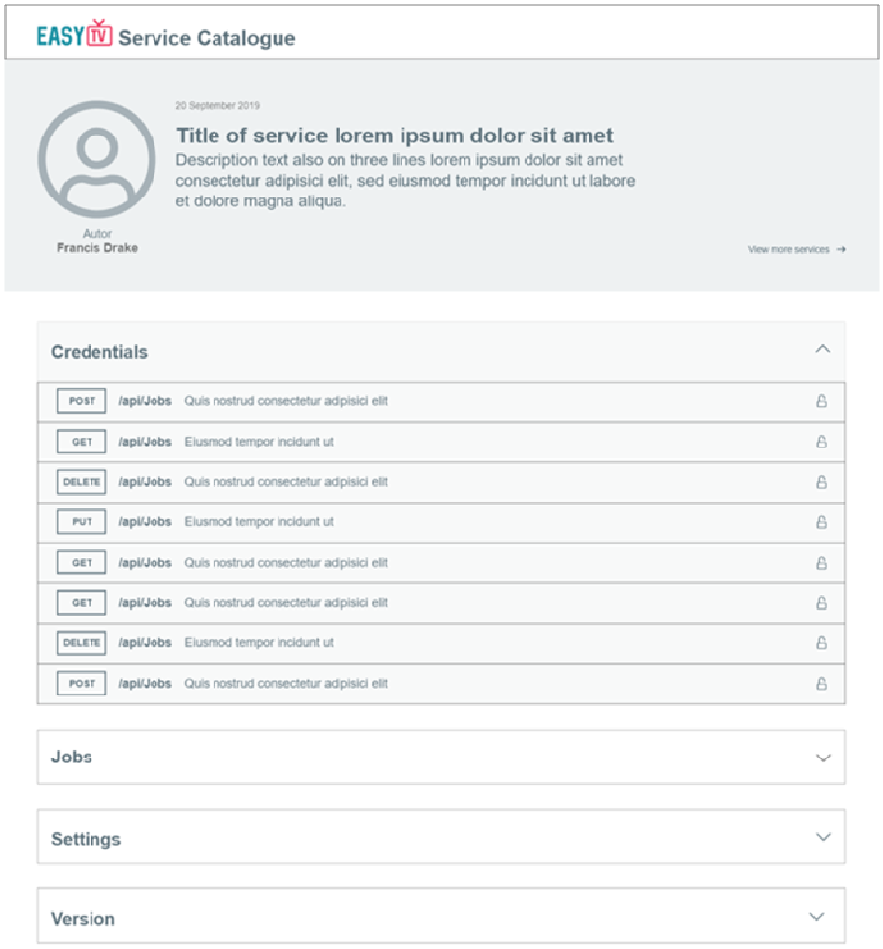


**Figure 13 - Service Catalogue Wireframe: Service detail page with open panel**

1. When a **panel** is open it is possible to view all the information related to the section.
2. Information are presented in different ways (Tags, text and icon) in order to give a visual hierarchy to the information for their **immediate** understanding.

### 4.5.2. **Design**

At the following link the interactive version of Service Catalogue design is available:

https://xd.adobe.com/view/27158834-bfd3-42aa-4c7d-2d00e4c67fe4-5124/screen/20a22920-cbae-49de-98d3-2fb3f3c48693/Service-CatalogueDES-HOME.

Starting from wireframes, the design process has led to the graphic design of prototyped pages. Compared to wireframes, graphic design optimizes the visualization of pages' components and structure, introducing elements as typography, colors, pictures.
Elements displayed in the graphic design are fundamental to evaluate the user experience perceived relating the final product.

Below the navigation flow is presented:

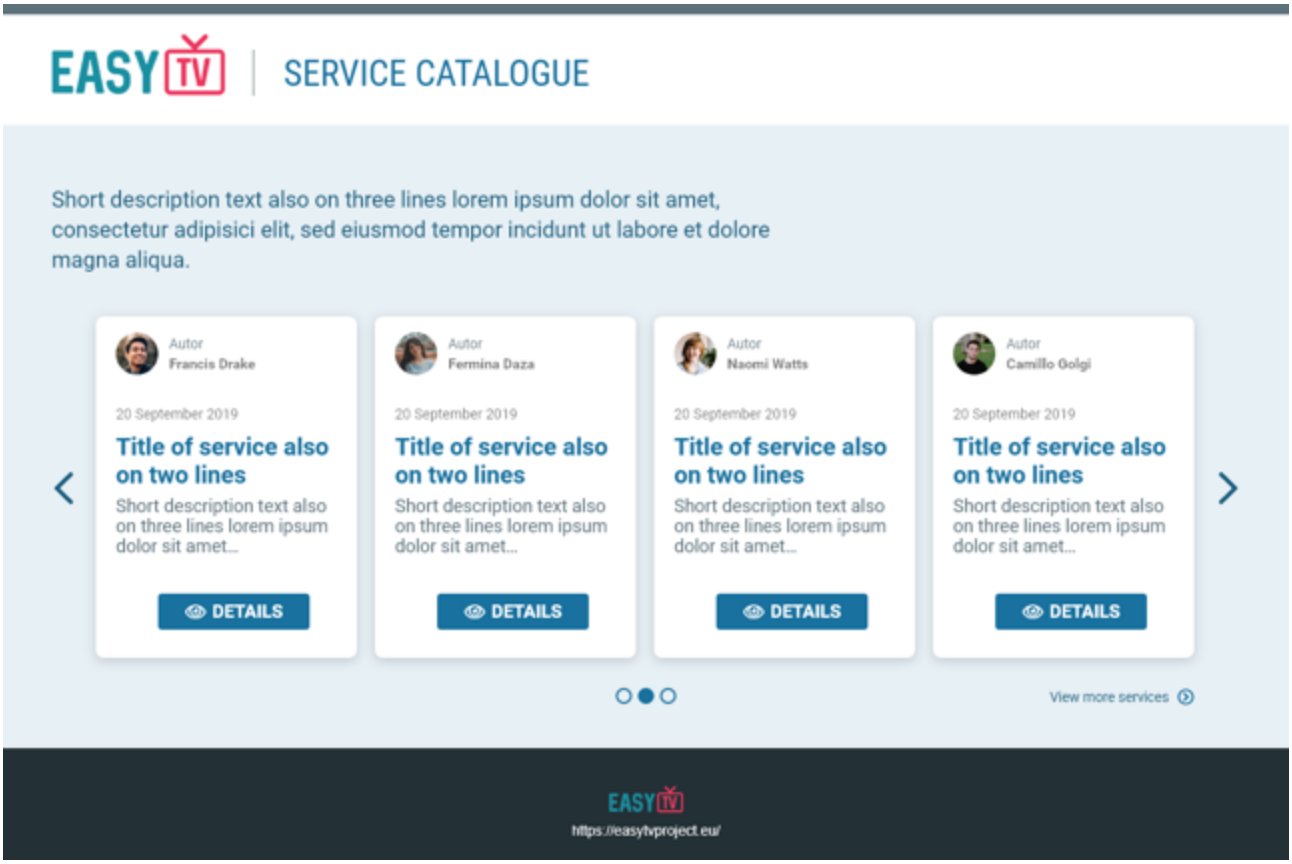


**Figure 14 - Service Catalogue Design: home page**

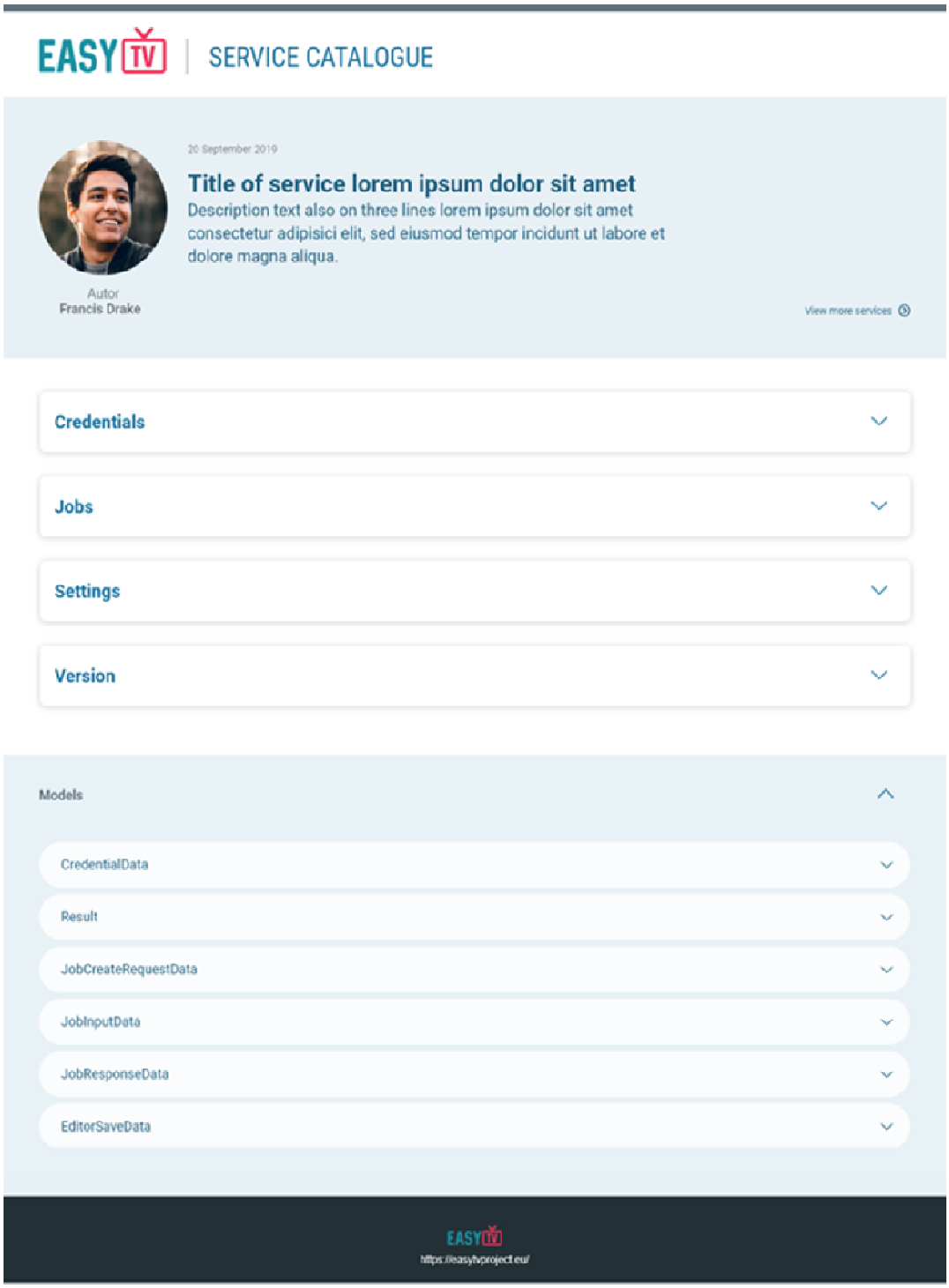


**Figure 15 - Service Catalogue Design: Service detail page**

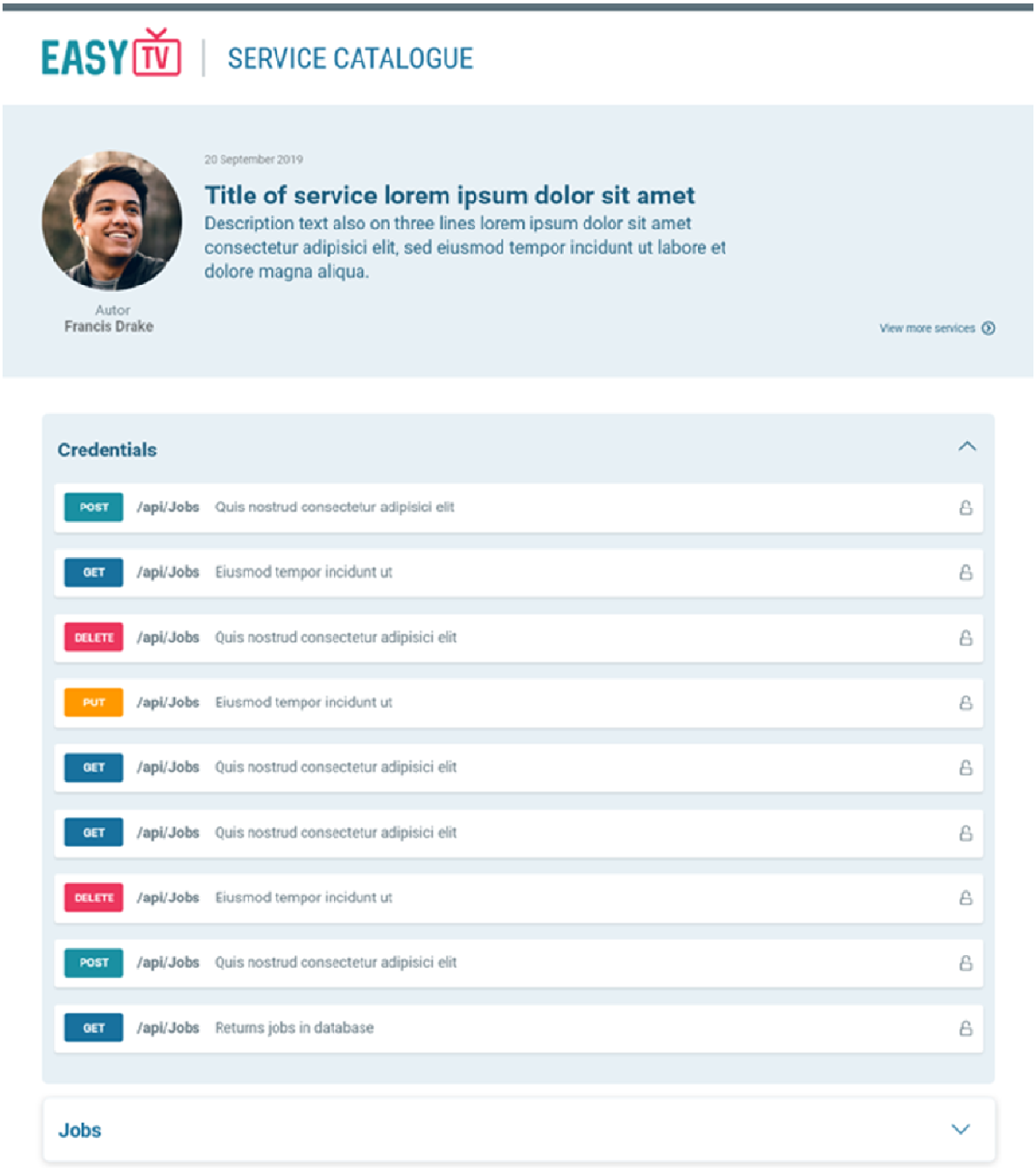


**Figure 16 - Service Catalogue Design: Service detail page with opened panel**

## 4.6. CMS choice and configuration

In addition to the user interface design and development, a CMS to manage Catalogue's data is needed: in particular, a system which can guarantee at least basic data operations such as search/create/read/update/delete is desired. Moreover, this kind of application should also allow a certain grade of flexibility in terms of UI development and integration. A specific category of CMS, named "*headless CMS*", fits the requirements listed above and in the majority of the cases offers additional features.

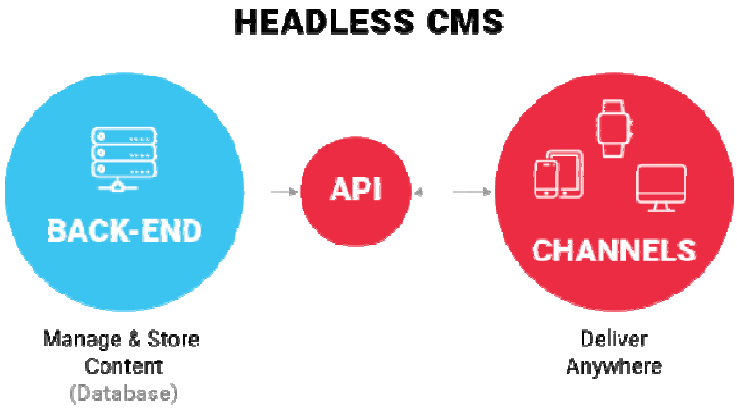


**Figure 17 - Headless CMS[10]**

The "headless" comes from the concept of removing the "head" (the frontend), leaving the "body" (the backend) only, effectively decoupling these two entities. A RESTful API (JSON or XML) or a GraphQL is usually employed to deliver content wherever needed. There are many advantages offered by this approach:

- Flexible content delivery
- Front-end agnostic
- Enhanced security
- Simpler deployment
- Easier third-party integrations

A large selection of headless CMS products is available on the market, with a wide range of licensing options. Here follows a partial list of popular products:

- Contentful
- ButterCMS
- Strapi
- dotCMS
- Netlify CMS
- Directus

After a period of evaluation, the choice has been to use Strapi: Strapi is a node.js based headless CMS which is easier to customise and configure compared to other products. It an Open Source software product and it is extensible by design, giving the opportunity to develop and install custom plugins if needed. Data is consumable from any client, including mobile apps and IoT devices, thanks to a well-designed API (RESTful or GraphQL).

Strapi has been installed on ENG server and properly configured through its administration web interface. Two content types have been created: "User" and "service". The "User" content type is

[10] https://www.brightspot.com/blog/decoupled-cms-and-headless-cms-platforms

employed during the authentication phase to verify that a user is allowed to access the Catalogue. The “Service” content type, on th other hand, models a service that can be accessed through the Catalogue. The following fields are part of the “Service” content type:

- Name – Service name
- Description – Service description
- Last_update – Last update date
- Author - Author
- Author_image – Author image or logo
- Swagger_json_file – API description in Swagger format
- Frontend_uri – Link to the frontend (if applicable)

Users have been created, then suitable roles and permissions have been defined through the “Roles & Permissions” plugin, which enables a JWT-based authentication mechanism.

## 4.7. Integration and final deployment

An Apache reverse proxy has been activated and configured in the way described in 3.5.1: the public domain address, in this case, is https://catalogue.easytv.eng.it. Of course, HTTPS is available through SSL/TLS certificates supplied by the “Let’s Encrypt” CA.

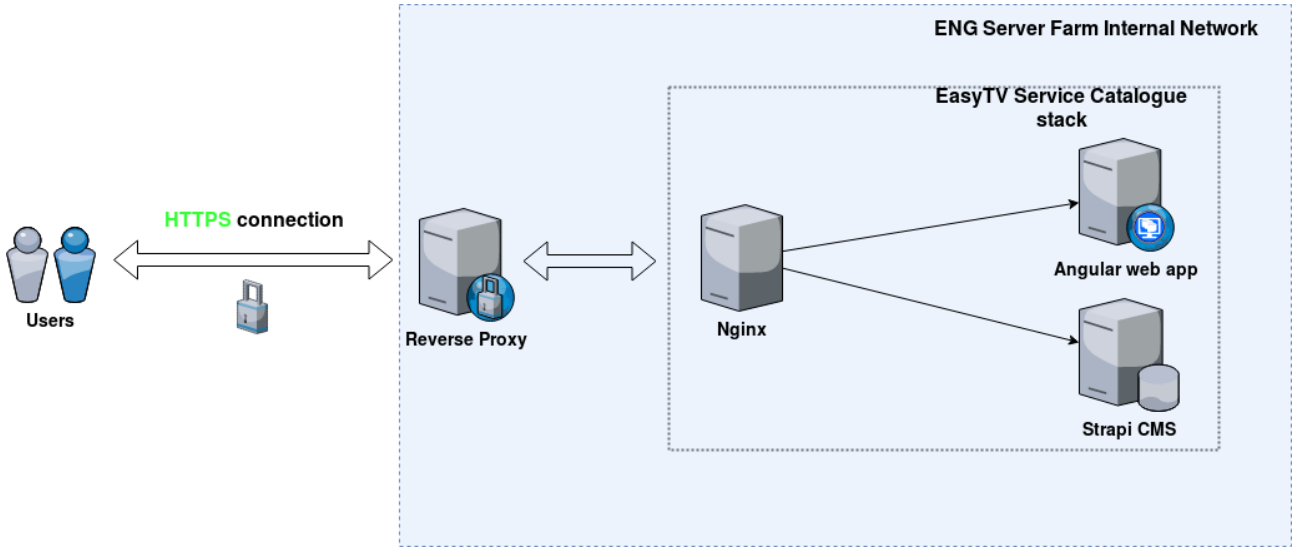


Figure 18 - EasyTV Service Catalogue network architecture

The final solution is the result of the integration process between the frontend (Angular web application) and the backend (Strapi headless CMS).

Nginx has been used to “glue” these two components together by means of an appropriate configuration (leveraging the “proxy_pass” directive):

```
[...]
        listen       8088;
        server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;


        location /ui/ {
```

```
        alias /home/easytv/catalogue/ui/dist/;
        try_files $uri$args $uri$args/ /ui/index.html;
    }

    location = / {
        return 301 https://catalogue.easytv.eng.it/ui/;
    }

    location / {
        proxy_pass http://localhost:1337;
    }
[...]
```

The base URL of the deployed Angular web application has been changed accordingly to suit the above Nginx configuration.

HTTP mocks inside the Angular application have been progressively substituted with real CMS API endpoint calls and tested to ensure that data could be managed properly. Additional tests on page routing were made to make sure that page flow works flawlessly.

Data regarding the services was finally gathered from each partner and entered in the system.

# 5. CONCLUSION

In this document, we reported the activities and related to the set up and implementation of the EasyTV Service Registry and Catalogue (Task 5.3).

In reference to EasyTV Service Registry, we:

- presented Service Registry motivation and choices;
- described the architecture;
- described the development, installation and configuration of the Registry;
- provided an overview of the backup strategy;
- described the detailed usage from a user perspective.

Regarding the EasyTV Service Catalogue, we:

- defined the kind of user which interacts with it;
- described the UI design methodology;
- described the frontend and backend development and integration;
- described the final asset deployment.

Both the Registry and the Catalogue are reachable at the addresses https://registry.easytv.eng.it and https://catalogue.easytv.eng.it respectively. Further improvements in terms of content (images and services available) are expected until the end of the EasyTV project.

# 6. REFERENCES

[1] Docker. Docker Registry. [Online]. https://docs.docker.com/registry/

[2] Docker. Docker Hub. [Online]. https://hub.docker.com/

[3] Riot. RiotJS. [Online]. https://riot.js.org/

[4] Apache. Apache Web Server. [Online]. https://httpd.apache.org/

[5] http-party. node-http-proxy. [Online]. https://github.com/http-party/node-http-proxy

[6] nginx. Nginx Web Server. [Online]. http://nginx.org/

[7] Apache. htpasswd utility. [Online]. https://httpd.apache.org/docs/2.4/programs/htpasswd.html

[8] Portainer. Portainer. [Online]. https://www.portainer.io

[9] Docker. Docker bind mounts. [Online]. https://docs.docker.com/storage/bind-mounts/

[10] Docker. Docker Registry HTTP API. [Online]. https://docs.docker.com/registry/spec/api/

[11] Nelson and Stolterman, *The Design Way: Intentional Change in an Unpredictable World*., 2012.