



**Project title:** Radiation risk appraisal for detrimental effects from medical exposure during management of patients with lymphoma or brain tumour (SINFONIA)

**Grant Agreement:** 945196

**Call identifier:** NFRP-2019-2020

**Topic:** NFRP-2019-2020-14 Improving low-dose radiation risk appraisal in medicine

## Deliverable D5.2 - Platform description, data model and protocols

<b>Lead partner:</b>	CESGA (5)
<b>Author(s):</b>	Jorge Fernández, Carlos Mouriño (CESGA); Mercedes Riveira-Martin, Antonio López Medina (SERGAS), John Stratakis (UoC), Tom Van Herpe (QAELUM)
<b>Work Package:</b>	5
<b>Delivery as per Annex I:</b>	Month 32 (30.04.2023)
<b>Actual delivery:</b>	Month 32 (30.04.2023)
<b>Type:</b>	Report
<b>Dissemination level:</b>	Public

*"This project has received funding from the Euratom research and training programme 2019-2020 under grant agreement No 945196"*



### Table of Contents

Abbreviations	3
List of figures	6
Executive summary	7
1. Introduction	8
2. Development methodology	10
2.1. Reminder of SINFONIA software development methodology	10
2.2. Development process of post-Y1 versions	10
3. Analysis of needs	12
3.1. Y1 prototype features review	12
3.2. Feedback during Y1 prototype development and use	13
4. Repository architecture	16
4.1. Reminder of Y1 prototype architecture	16
4.2. Y2 iteration architecture	17
5. Platform implementation	19
5.1. Software stack	19
5.1.1. Girder resource management middleware	21
5.1.2. Orthanc DICOM server	21
5.1.3. Django web application framework	22
5.1.4. Embedded web viewers	23
5.1.5. Underlying database systems	24
5.1.6. SFTP service	25
5.1.7. LDAP users' directory	26
5.1.8. External access control	26
5.2. Hardware platform	27
5.3. Update on repository features	28
5.3.1. Homepage and interface layout	28
5.3.2. Resources organisation, navigation, and operations	31
5.3.3. Users and groups management	38
5.3.4. Interaction with the FTP server	40
6. Y2 version validation	42
6.1. Review of the delivered version	43
7. Conclusion	46
References	47

## Abbreviations

2FA	2-Factor Authentication
ACID	Atomicity, Consistency, Isolation and Durability (set of properties for DB operations)
ACL	Access Control List
AI	Artificial Intelligence
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
BSD	Berkeley Software Distribution
BSON	Binary JSON
CESGA	Centro de Supercomputación de Galicia (Galicia Supercomputing Centre)
CORS	Cross-origin resource sharing
CRUD	Create, Read, Update, Delete operations
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DARPA	Defense Advanced Research Projects Agency (US Department of Defense)
DB	Database
DBMS	Database Management System
DICOM	Digital Imaging and Communication on Medicine
DICOMDIR	DICOM fileset directory file
DMB	Data Management Board
DSL	Domain-specific language
D<x>.<y>	Deliverable <x>.<y> within SINFONIA project
EPUB	Electronic Publication file format
EU	European Union
FTP	File Transfer Protocol
GB	Gigabyte
GIF	Graphics Interchange Format
GDPR	General Data Protection Regulation
GNU	GNU is Not UNIX (recursive acronym)
GUI	Graphical User Interface

HMVC	Hierarchical MVC
HTML(5)	HyperText Markup Language (version 5)
HTTP(S)	HyperText Transfer Protocol (Secure)
ID	Identifier
JPG	Joint Photographic Experts Group format
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MIT	Massachusetts Institute of Technology
MP4	MPEG-4 Standard Part 14 video format
MPEG	Moving Picture Experts Group
MTV	Model-Template-Views software architecture pattern
MVC	Model-View-Controller software architecture pattern
M<x>	Month <x> (in project context, counting from September 2020 - M1)
NFS	Network File System
NSS	Name Service Switch
OHIF	Open Health Imaging Foundation
OOXML	Office Open XML file format specification
PACS	Picture Archiving and Communication Systems
PAM	Pluggable Authentication Modules
PDF	Portable Document Format
PNG	Portable Network Graphics
RAM	Random Access Memory
REST	Representational State Transfer
RGB	Red, Green, Blue
RT	Radiation Therapy
SERGAS	Servizo Galego de Saúde (Galician Health Service)
SKANDION	Skandionkliniken - Kommunalförbundet Avancerad Strålbehandling
SCO	Świętokrzyskie Centrum Onkologii (Świętokrzyskie Oncology Centre)
SFTP	Secure FTP
SQL	Structured Query Language

SSSD	System Security Services Daemon
SU	Stockholms Universitet (University of Stockholm)
UoC	University of Crete
UNIGE	Université de Genève (University of Geneva)
UID	Unique Identifier
URL	Unique Resource Locator
US(A)	United States (of America)
UTC	Coordinated Universal Time
VM	Virtual Machine
VPN	Virtual Private Network
WP	Work Package
YAML	"YAML Ain't Markup Language" file format (recursive acronym)
Y<x>	Year <x> (in project context, 12-month periods from September 2020)
<yyyymmdd>	Concatenation of 4-digit year, 2-digit month, and 2-digit day of month numbers
<hhmmss±tz>	Concatenation of 2-digit hour, 2-digit minutes, and 2-digit time offset from UTC+00

### List of figures

Figure 1: Post-prototype versions development process steps	10
Figure 2: Summary chart of needs and constraints fulfilled by the Y1 prototype	12
Figure 3: Block diagram of the Y1 prototype architecture	16
Figure 4: Overview of a three-tier application	17
Figure 5: Overview of the Model-View-Controller pattern	17
Figure 6: Block diagram of the extended platform architecture	17
Figure 7: Mapping of the software stack implemented to the extended platform architecture	19
Figure 8: Unfolded diagram of platform components' implementation	20
Figure 9: Screenshot of homepage for anonymous users	28
Figure 10: Screenshot of homepage for authenticated users with dark mode and user menu active	29
Figure 11: Screenshot of contact page for authenticated users	30
Figure 12: Screenshot of "My profile" page for "admin" user	30
Figure 13: Screenshot of Collections page for authenticated users	31
Figure 14: Screenshot of Folder page with Folder Operations menu unfolded on the right	31
Figure 15: Screenshot of "Edit folder" dialog	32
Figure 16: Screenshot detailing a Folder information panel	32
Figure 17: Screenshot detailing the basic contextual menu	33
Figure 18: Screenshot with an "Elements currently picked" dialog	33
Figure 19: Screenshot detailing a Move/Copy/Clear contextual menu	34
Figure 20: Screenshot detailing a drag-and-drop resource movement	34
Figure 21: Screenshot with an "Upload files" dialog	35
Figure 22: Screenshot detailing a stack of upload notifications	35
Figure 23: Screenshot of a fileset page previewing a DICOM file and with its list entry highlighted	36
Figure 24: Screenshot of an OHIF viewer tab displaying a DICOM study	36
Figure 25: Screenshot of an OnlyOffice viewer tab displaying a CSV file	37
Figure 26: Screenshot detailing the new SINFONIA ID search method	37
Figure 27: Screenshot with a search results dialog	38
Figure 28: Screenshot of the "Users" page of the repository	38
Figure 29: Screenshot partially displaying the Groups list of the repository	39
Figure 30: Screenshot with details and users list for CESGA group	39
Figure 31: Screenshot of a successful transfer from FTP to user folder in the repository	40
Figure 32: Screenshot of a successful transfer of files from a fileset to the user's FTP space	41
Figure 33: Screenshot of the dialog with information for using the FTP server	41
Figure 34: Summary chart of needs and constrains fulfilled by the latest Y2 increment	44

### Executive summary

One of the key objectives of the SINFONIA project is to develop and operate a repository to collect and pool data from imaging and non-imaging examinations and radiation therapy sessions, histologic results and demographic information related to individual patients with lymphoma and brain tumours. WP5 is in charge of tasks for achieving this key objective. Task 5.1 was focused on the definition of the platform architecture and the implementation of a functional prototype after gathering the requirements and demands from the SINFONIA consortium. That work is described in deliverable D5.1. Task 5.2 is dedicated to the creation of the platform including data storage, computational capabilities, and protocols. The platform is based on the aforementioned prototype, and yearly versions of the repository are being produced based on the feedback received from users. This document mainly covers the development of Y2 version.

These yearly versions are developed as if they were prototypes themselves, in line with the principle of continuous improvement inherent to iterative and incremental development methodologies introduced in D5.1. Thus, the development process for the yearly versions is inspired by the one followed for the prototype, but with variations to make it even more flexible and to allow the incorporation of minor changes or technical corrections on the fly.

The architecture of the platform extends the one defined for the prototype. In particular, it now supports a separate management of DICOM files by means of an Orthanc PACS (with respect to the Girder middleware, now focused on non-DICOM resources), a new web application integrating these two components, an LDAP service for user authentication, and an FTP service for bulk data transfers. All these components have been implemented in the form of Docker containers that communicate with each other inside a virtual machine hosted in CESGA's cloud infrastructure, exposing through proxy servers the parts strictly necessary for allowing users to work with the repository. The new web application includes functionalities requested by users but not yet covered in the prototype, such as the aforementioned document viewers, or others such as support for SINFONIA identifiers, a contact form for users, or improvements in resource movement/copy operations. Together with the web application, a new user interface has been developed, although based on the previous one so that user could quickly adapt to it.

The Y2 version of the platform was presented in the 3rd SINFONIA Consortium Meeting in Vienna in late October 2022. Users had the opportunity to learn about and test this new version in a workshop held within that meeting. Thanks to the incremental nature of the development process, part of the feedback gathered in the workshop, together with some bugs and other improvement possibilities detected internally by WP5, has been incorporated into the platform throughout the time elapsed until the expected delivery date of this document in M32 (April 2023).

As a result, the functionalities offered in this version of the platform satisfy a significant part of the needs expressed by the users, except for the integration and execution of AI algorithms, which belongs to Task 5.3 and is currently in its initial stages.

## 1. Introduction

The main objective of the 4-year SINFONIA project is to develop novel methodologies and tools that will provide a comprehensive risk appraisal for detrimental effects of radiation exposure on patients, workers, caretakers, and comforters, the public, and the environment during the management of patients suspected or diagnosed with lymphoma and brain tumours.

One of the key objectives of the project is to develop and operate a repository to collect and pool data from imaging and non-imaging examinations and radiation therapy sessions, histologic results and demographic information related to individual patients with lymphoma and brain tumours. This platform will be used in developing methods and techniques described in other WPs for patient-specific dose estimation, risk assessment and uncertainty determination. It will provide services such as data uploading/downloading, data preview, information search information, or implementation and allocation for execution of AI algorithms. The European (GDPR<sup>1</sup>) and other multi-national regulations on data protection obliges the repository to ensure the anonymization or removal of personally identifiable information. Thus, several guidelines and best practices regarding patient data privacy and protection will be prepared and distributed among project partners. This will enable them to focus on their core tasks since data storing and protection will be managed centrally.

The tasks for achieving the aims are included in SINFONIA Work Package 5 (WP5). Specifically, the WP is organised in Tasks 5.1 to 5.3. Task 5.1 was focused on the definition of the platform architecture after gathering the requirements and demands about the repository from the different project partners. This gathering is expected to continue throughout the project to improve the platform and release evolved versions by means of a continuous improvement methodology. Task 5.2 is dedicated to the creation of the platform including data storage, computational capabilities, and protocols according to the requirements gathered in Task 5.1. A first prototype was released meeting the project schedule at the end of Year 1 (Y1 prototype, M12), whereas major improved versions are planned to be released every year until the end of the project in Y4 (M48), starting with the Y2 version described in this document. Task 5.3 will conclude the development with the integration of the features to support AI algorithms.

The previous deliverable, D5.1, titled “Requirements and design of the prototype architecture”, was submitted for public release in M16 (December 2021) and it is available for review at the public information site of the SINFONIA project. That document covered the WP5 work in Task 5.1 and the first steps of Task 5.2. Namely, that document describes the procedure for capturing the users’ needs and requirements, and the analysis of those needs and the technologies available for achieving a suitable design that sets the foundation for implementing the SINFONIA data repository. Moreover, as the due date of that deliverable was several months after the due date for releasing the Y1 prototype, the implementation and validation phases of its development process were also covered for completion’s sake.

The current document, D5.2, titled “Platform description, data model and protocols”, covers the foundational work of WP5 in Task 5.2. In particular, it describes how the basic architecture of the Y1 prototype has been extended to achieve a first iteration of a full platform architecture, along with the implementation of the Y2 version of the repository from such extension. Both the architecture design and its software implementation are considering all the research performed, information collected to prepare the Y1 prototype, as well as the feedback received asking for new features and improvements. The definition of a data model according to the needs raised by the users was one of the fundamental tasks addressed during the development of the Y1 prototype. This data model remains essentially valid as part of the core of the

---

<sup>1</sup> [Regulation 2016/679](#) of the European Parliament and of the Council of 27 April 2016



subsequent iterations of the repository. Since it has already been described extensively in D5.1, this document simply makes the necessary references to that description of the model, rather than presenting it all over again. Regarding the protocols, these are implicitly defined by the programming and communication interfaces of the different software components that shape the architecture. Task 5.3 is marginally covered also, as the architecture, data model and protocols are generic enough to support the later integration of the AI algorithms execution platform.

The rest of this document is structured as follows: Section 2 recalls the software engineering methodology that guides the development of the SINFONIA repository, focusing on the particularities of the incremental releases' development process with relation to that of the Y1 prototype. Section 3 starts with an update on the fulfilment of users' needs and enriches them with the feedback received during the Y1 prototype lifetime, which is the source of information for the design of an extended architecture for the repository. Section 4 presents the architecture for Y2 and subsequent versions of the repository as an extension of the basic one from the Y1 prototype, which is recalled first. Section 5 describes the software stack that embodies the new architecture as the Y2 implementation of the repository, the hardware platform on which this software stack runs, and the major changes and improvements regarding the features this version offers. Section 6 describes the validation of the Y2 version performed both internally within WP5 and externally after presenting the updates to the Consortium, along with a review of the fulfilment status of the needs initially identified by the users and advancing some insights on how the architecture and its implementation might be extended in further iterations. Section 7 concludes the document by summarising its contents.

## 2. Development methodology

This section starts with a brief reminder of the software development methodology established for the SINFONIA repository, which was already presented in section 2 of D5.1. The general development process of an iteration of the repository is then recalled, also detailing the particularities of this process for the development of iteration Y2.

### 2.1. Reminder of SINFONIA software development methodology

As introduced in D5.1, the development of the SINFONIA repository follows a yearly prototyping methodology, this is, a hybrid software engineering approach that combines the rapid prototyping model and the incremental model.

The rapid prototyping model was applied for obtaining the Y1 prototype, with the purpose of providing the Consortium with a tool that would support a set of essential functionalities and allow them to start exchanging information within the project as soon as possible. In line with the nature of this development model, the prototype would also help users to better identify their needs and provide feedback to the development team on how to refine and extend functionalities accordingly.

A combination of prototyping and incremental models is driving the development of the repository from Y2 to Y4, with successive major releases of the SINFONIA data repository occurring approximately at the end of Y2 (M24), Y3 (M36) and Y4 (M48).

### 2.2. Development process of post-Y1 versions

Yearly versions are developed as if they were prototypes themselves, in line with the principle of continuous improvement inherent to iterative and incremental development methodologies. This enables the incorporation into each development of internal and external review processes that allow users to report issues and propose suggestions. By these processes, minor patches and improvements can be applied on the fly, and at the end of the lifetime of each major version all the information gathered can enrich the inputs to the development process of the next one.

The development process of these post-prototype versions is structured in a very similar vein to the one followed in the development of the Y1 prototype. Nevertheless, there are some differences that are convenient to explain at the same time the stages of the process (depicted in Figure 1) are recalled:

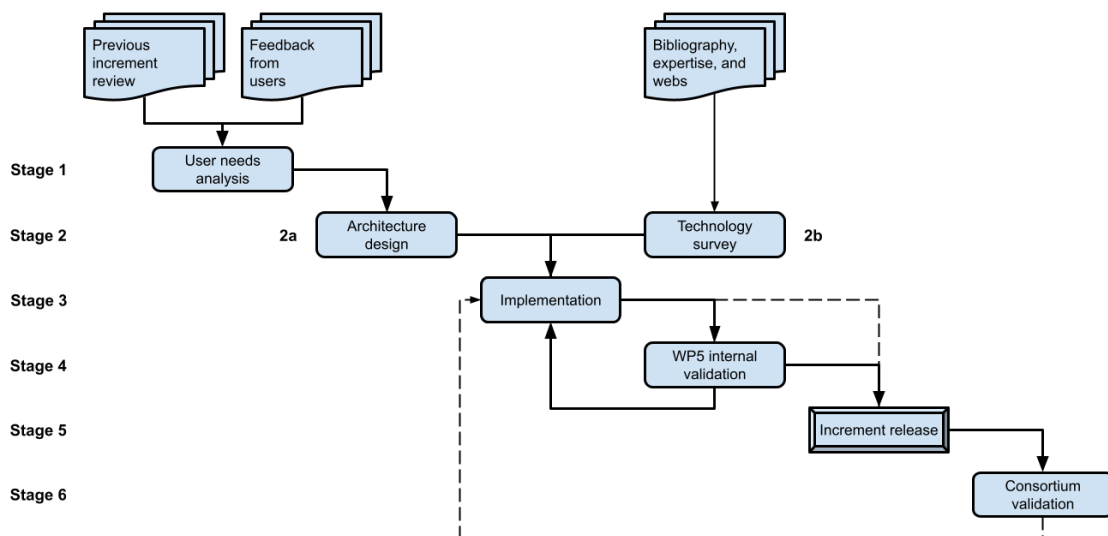


Figure 1: Post-prototype versions development process steps

1. **User needs analysis:** This stage has a dual input. On the one hand, the review about the grade of fulfilment the previous version reached with respect to its requirements, and on the other hand, the feedback received from users during the service life of that previous version (suggestions, requests for new features, notification of bugs, etc.), which is a valuable source of information to reveal unidentified needs and to prioritise improvements and new development of certain functionalities.
- 2a. **Architecture design:** After reviewing both the results of the validation process of the previous version and the feedback received from users, extensions to the architecture may be proposed if they are needed to support the implementation of the new increment. This is imperative in the case of the development of the Y2 increment, as the architecture of the Y1 prototype was designed with only basic functionalities in mind to make the repository useful.
- 2b. **Available technologies survey:** The knowledge acquired after the extensive survey of webs, source code repositories and computer science bibliography performed for the Y1 prototype is exploited, along with the expertise of the development team, to choose suitable tools to implement any extension to the architecture of the repository.
3. **Implementation:** The extended architecture becomes real after a software implementation process, building in also the corrections and improvements requested.
4. **WP5 internal validation:** Generally, once the implementation is ready, it is presented to the rest of the WP5 members for validation. Their feedback is considered to solve any issues they may find and to apply the improvements they may propose. In case of minor revisions whose application has already been agreed upon, or modifications affecting very specific technical issues, it would not be necessary to go through a new validation process and this step could be omitted (the dashed line from Stage 3 to Stage 5 represents this omission).
5. **Increment release:** Improvements and updates introduced by major releases are presented to the SINFONIA Consortium, and the new version is made available to its members.
6. **Consortium validation:** After the release, the Consortium members are encouraged to conduct their own individual validation by means of their daily use of the repository. Simultaneously, the development team within WP5 will remain attentive to any report of issues or proposals for suggestions it may receive. Depending on the importance of the issue and the extent of the changes required to resolve it, the solution may be incorporated directly into the running version (loopback from Stage 6 to Stage 3, and direct progress to Stage 5), or go through the whole process again from step 3 onwards.

### 3. Analysis of needs

Instead of repeating a complete process of requirements capture like the initial one conducted for the Y1 prototype (circulation of questionnaires among the users, analysis and prioritisation of answers - see Section 3 of D5.1 for details), for this increment we will start from the final validation assessment performed in Section 6.3 of D5.1, that details the grade of fulfilment the Y1 prototype reached with respect to the requirements collected from users in that initial capture. Moreover, we also review the most relevant feedback received during Y1 development and early lifetime of the prototype.

This section presents both sources of information, as the requirements drawn from them establish the baseline for designing the extended architecture to be presented in Section 4. This extended architecture which will lay the foundation for the development of the yearly major versions.

#### 3.1. Y1 prototype features review

The Y1 prototype was built on top of Girder, an open-source tool for rapid deployment of scientific data repositories. Sections 3.2.2 and 4.3 of D5.1 list the main features of Girder and justify in detail its choice. The key strengths of Girder, in terms of the needs expressed by the SINFONIA partners, are to support a resource organisation generic enough to host any kind of information (i.e., it is not tightly coupled with a particular type of procedure or experiment), and to offer a plugin that allows easy previewing and browsing of DICOM files.

This was done to provide users as soon as possible with a functional repository, enabling a quick and simple way for sharing information from the early stages of the project. For that purpose, a clean instance of Girder was taken, a series of customisations were applied to the appearance of its web pages, certain adjustments were made to its configuration to adequately restrict access only to members of the project, and different modifications were made to its code to support all the main features described in Section 5.4 of D5.1.

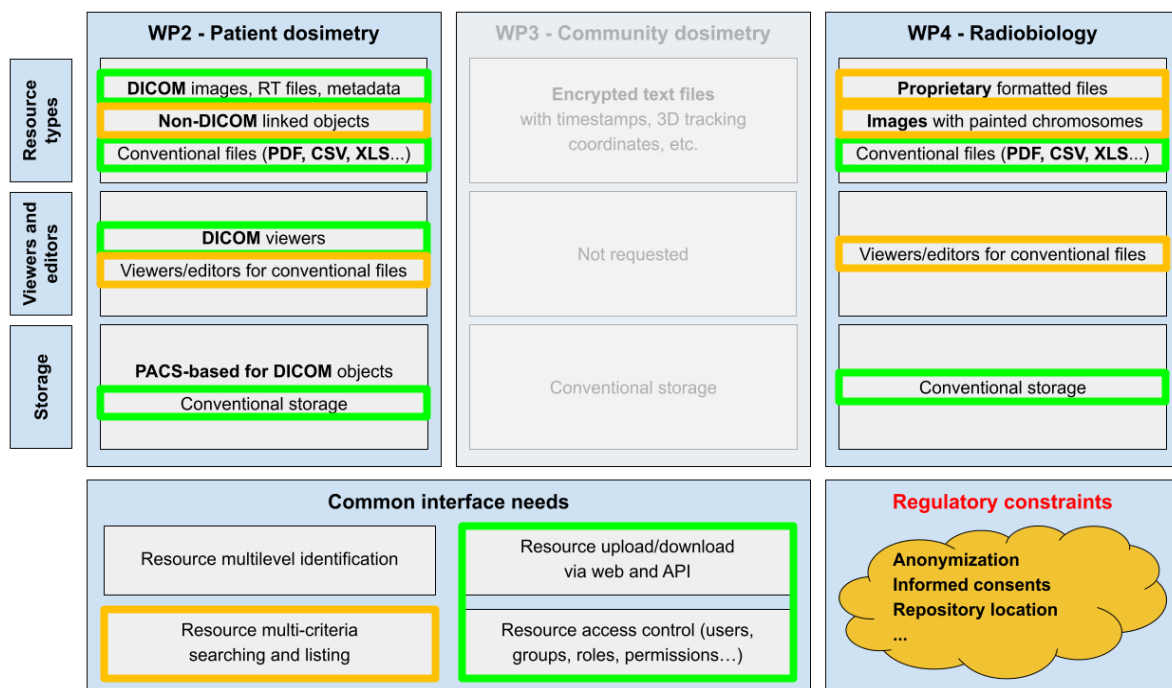


Figure 2: Summary chart of needs and constraints fulfilled by the Y1 prototype

The chart in Figure 2 summarises up to which extent the Y1 prototype released at M11 (July 2021) satisfied the needs detected by the analysis of the answers to the requirements capture questionnaires circulated

among the SINFONIA partners. Each requirement is marked in green, yellow, or not marked, depending on being totally, partially, or not fulfilled, respectively.

The Y1 prototype provided most of the common interface features through the customised Girder web application: upload and download features were enabled through both the web portal and the REST API as Girder offered them, just as was the definition of access control lists to set permissions on uploaded resources. Both DICOM and conventional files such as text documents, spreadsheets or images were accepted by the prototype.

Let us review now which requirements were partially fulfilled or not fulfilled (see Section 6.3 of D5.1 for the full details) at the end of Y1. As for the common interface features, the search engine provided by Girder was considered as adequate, as it was able to find resources by names, descriptions, and the content of DICOM tags, although it was assumed that in successive iterations users will be interested in more complex searches (combining several criteria, for example) and the presentation of results visually was not very refined either. Regarding the needs stated by WP2 users, the mechanism for linking non-DICOM and DICOM resources was not implemented as it depended on the incorporation of the project's data resource identification scheme, whose discussion at Consortium level was then at a very early stage. WP3 partners expressed their intention not to use the repository on a regular basis, mainly because of their specific needs to upload massive amounts of information in real time, which also implied a significant over dimensioning of the platform compared to the needs of most partners. This is why their requirements are greyed out in the chart. With respect to WP4 needs, the Y1 prototype did not provide any special features for manipulating specific-formatted files handled in their usual procedures or images with painted chromosomes, although they may be anyhow uploaded to and shared through the Y1 prototype.

There are also some requirements marked in the chart in green (i.e., as fulfilled) that were indeed implemented following very elementary but functional approaches, with the intention to improve or replace them in subsequent increments. This is the case for components such as the DICOM previewer or the search engine, both included nearly as Girder provided them, so that users could use them as a starting point to work with, and to identify other needs beyond the basic functions already offered by them.

Last but not least, regulatory constraints about privacy and security were enforced by means of the Anonymization Guidelines document, the restriction of registration to SINFONIA members and of navigation to logged-in users, the request to project members to explicitly accept the SINFONIA Data Repository Users Privacy Policy prior to the creation of their user accounts, and the hosting of the physical infrastructure on which the prototype was deployed on CESGA's premises in the EU (Santiago de Compostela, Spain).

### 3.2. Feedback during Y1 prototype development and use

During the internal validation process of the Y1 prototype (see Section 6.1 of D5.1 for a full description of that process), the rest of the WP5 partners made a number of suggestions for improvements and additional features for which the development team from CESGA considered that either because of their complexity or because of external issues not directly related to the work of WP5, it would be more convenient to include them in later increments of the repository.

UoC suggested the activation of an FTP service to provide users with a mechanism for massive data uploads and downloads. CESGA agreed with this, since the upload of large volumes of data (in range of GB) over HTTP could cause an overload of the repository web proxy and then lead to a denial of service for any other user who wanted to use the platform. A complementary solution to this issue being considered by the development team was the support of chunked and streamed transfers to speed up HTTP uploads and downloads.

UoC also recommended enhancing the security of the repository by limiting access to specific known IP addresses from the SINFONIA partners. While it is a very sound proposal, the development team considered that it could hinder the use of the repository from outside the networks of the users' institutions (e.g.: access from home, travelling to conferences, institutions not offering VPNs, changes in institutions' IP ranges, etc.). Moreover, for the moment the access to the repository is restricted to registered project users, and they have full control on the access permissions for the resources they upload, which is considered a sufficient measure at this stage of the repository lifetime.

QAELUM and SERGAS also made some proposals for improving the user experience with the repository web interface. For instance, QAELUM proposed the inclusion of a contact form for reporting issues or making suggestions (instead of the plain link to an email address provided in the prototype) and an extensive usage of tooltips<sup>2</sup> for guiding users through the application. SERGAS recommended the adoption of a more user-friendly mechanism for copying/moving resources within the web portal, for example by means of a dialog on which a tree-like list is shown to the user and they can choose a specific destination point for the resources to copy/move.

In subsection 3.1 we deemed the Girder-offered DICOM previewer to be just a starting point to offer this required feature. This is because users in general found this previewer useful but at the same time quite limited (e.g.: lack of bright/contrast auto levels, lack of support for some RT-related modalities, order of instances alphabetically according to filenames instead of by the numbers set in DICOM tags) and sometimes sluggish (DICOM files were being downloaded and locally processed in order to render them in the user's browser) and unstable (we had to make very specific tweaks to the code in order to make the previewer support both grayscale and RGB pixel data of images from several kinds of DICOM modalities). All these problems quickly revealed the need to provide a more stable previewer and/or to include a DICOM web viewer for research.

UoC also shared some ideas about their vision for integrating the execution of AI algorithms in the platform, such as providing a jupyterHub<sup>3</sup> service for running scripts remotely on uploaded data. The CESGA development team shares this vision, but they want to explore other alternatives depending on the specific needs of the partners (recall that the questionnaire only asked about the need to run algorithms or not, without going into technical details). With relation to this, SKANDION contacted CESGA to ask about the possibility of running compiled MATLAB implementations of algorithms in the platform.

In the very early stages of the prototype development, SERGAS and UoC prepared a proposal on a resource identification scheme based on the concatenation of several items: the SINFONIA project ID, an internal ID somehow connected to WP tasks and subtasks, the partner ID, a serial number for the patient, and additional partner-level notation for particular characteristics of the data uploaded (paediatric patients, specific anatomical regions, etc.). This topic was raised to the SINFONIA Data Management Board (DMB) by WP2 and WP5 representatives, which triggered a discussion that eventually led to a project-wide scheme based on the initial one. Further iterations of the repository were expected to support marking and searching resources using this scheme or any eventual evolution.

Regarding both the external validation process and the lifetime of the Y1 prototype, the Consortium generally seemed to feel rather comfortable with its features and interface, or at least this could be inferred from the few suggestions received from outside WP5 once it was released in M11 (July 2021) and officially presented in M13 (September 2021) in the 2<sup>nd</sup> CM meeting. However, it is also worth acknowledging that many partners

---

<sup>2</sup> Small box that displays additional information about an element when the user hovers their cursor over it.

<sup>3</sup> <https://jupyter.org/hub>

were still in the process of collecting data (and therefore had little opportunity to share information through the repository) and the active participation of many of them in an online demonstration session of the prototype that the development team gave in M10 (June 2021). That session was an important source of suggestions eventually materialised in some of the modifications made to the Girder interface and code, and which resulted in the released prototype.

After the release of the prototype and its official presentation, we received proposals from SU, that contacted us to propose the support for collaborative editing of documents (e.g.: spreadsheets, text files) and from SCO, which, when asked about the possibility of uploading DICOMDIR datasets directly as ZIP files, brought up a set of possible functionalities for handling compressed datasets in the repository, such as previewing, selecting of a subset of files for downloading or decompressing the dataset remotely after uploading it.

## 4. Repository architecture

This section describes how the architecture designed for the Y2 iteration of the repository extends and refines the architecture inherited from the Y1 prototype, so that it can accommodate both the needs of the users still to be covered and the proposals for improvement that are received from them. First, we review the basic architecture of the Y1 prototype already described in Section 4 of D5.1. Then, the new extended architecture is presented, motivating the changes and extensions applied to the basic architecture according to the analysis of users' needs and feedback presented in Section 3.

### 4.1. Reminder of Y1 prototype architecture

From its inception, the architecture of the Y1 prototype was created as the foundation for the development of the actual platform and its iterations from Y2 onwards. This basic architecture can be recalled in the diagram in Figure 3. In that initial approach to the repository architecture, the entry points for users into the repository were a web portal and a REST API, both provided by Girder, the middleware in charge of indexing, storing, and serving the different types of resources that the SINFONIA users would manage through the repository. These entry points were exposed to the Internet through a properly secured HTTPS proxy server. Regarding the underlying infrastructure, the repository server hosted the middleware and mapped the resources and the indexing database to a physical storage on which they were effectively saved.

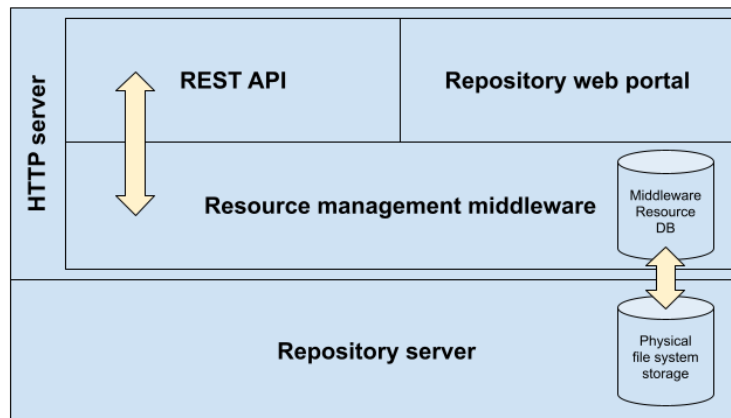


Figure 3: Block diagram of the Y1 prototype architecture

This architecture fits into the 3-tiered model, a software architecture pattern widely used in web applications. Figure 4, which is taken from [1], offers a visual explanation of the role of each of the 3 layers, which are, from the outermost to the innermost: presentation, logic, and data. In this case, the first presentation tier is composed of the web portal and the REST API, the intermediate logic tier consists of the code of the resource management middleware, and the bottom data layer is embodied in a database that indexes the resources stored in the repository. These three layers rely on a repository server on which their codebases are run, and data is physically stored.

From the point of view of the tasks performed by each tier of the architecture, it would also fit into the Model-View-Controller (MVC) software architecture pattern depicted in Figure 5 [2]. The central component of this pattern is the Model, which represents the information structure of the application. A View is any representation of information, which in web applications are usually HTML templates that are displayed in the user's browser filled with the specific data retrieved. The Controller just accepts input and converts it to commands for the model or view. In this case, the model *M* is the schema and the contents of the middleware database, the controller *C* is the middleware application code that runs on the server, and the *V* view is the web portal pages through which users interact with the repository.



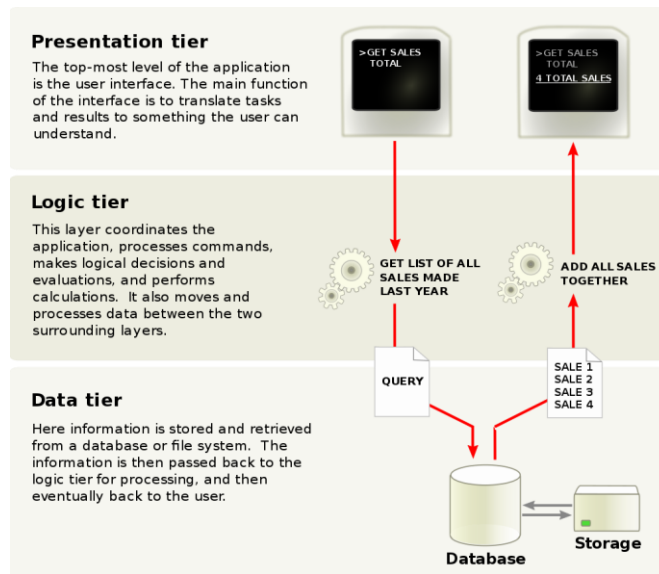


Figure 4: Overview of a three-tier application



Figure 5: Overview of the Model-View-Controller pattern

### 4.2. Y2 iteration architecture

Once a set of features has been identified to be improved or completed or that is awaiting implementation, and all the issues detected and suggestions proposed by the users during the lifetime of the Y1 prototype have been analysed, an extension of the basic architecture emerges as necessary. The diagram depicted in Figure 6 is the new extended architecture emerging from that process.

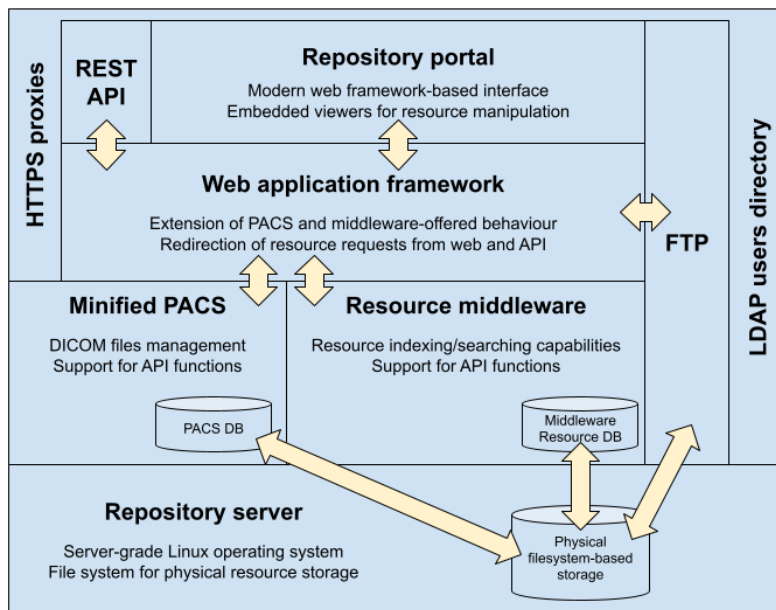


Figure 6: Block diagram of the extended platform architecture

Compared to its predecessor, one of the main changes the reader can perceive is the new “Web application framework” that has been inserted below the “Repository Portal” and the “REST API” components. This new component is expected to serve the web portal and the REST API by itself, allowing us to extend or adapt its behaviour according to the evolutions that the architecture is expected to undergo as users' needs also evolve. Thus, we will no longer assume the resource management middleware as a closed implementation of the MVC pattern but decouple tasks such as the generation of web pages and responses to users' requests when they interact with the repository. In particular, we are keeping the resource management middleware to play the roles of model and controller, encapsulating them into the same-named component of our new architecture. Recalling the information model presented in Section 4.1 of D5.1, the support for indexing and storing non-DICOM resources, and for managing users, groups, and permissions, will rely on this component. At the same level is another new component called “Minified PACS”<sup>4</sup>, introduced to provide an enriched support of the so-called “*DICOM model of Real World*” [3], that represents DICOM resources as coming from patients that can be submitted to a number of studies, with each study containing a number of series, and each series being a collection of DICOM files named instances.

With these changes the architecture becomes a hierarchical MVC (HMVC) since the internals of part of the components are still following by themselves a full MVC pattern. In our case, both the minified PACS and the resource management middleware will offer their respective programming interfaces (view) through which the web framework will be able to make the necessary requests to them, these requests will be serviced by their respective application codes (controller) that will query and/or make the necessary changes in the corresponding databases (model). While the comparison between the diagrams from Figures 3 and 6 may suggest the opposite, the number of layers in the architecture remains unchanged, albeit the boundaries between them have become a little more blurred. There is still a presentation layer (the portal and the API), a logic layer (now split between the functions we have kept for the middleware and those we have moved to the PACS), and a data layer distributed among the databases that index and store the managed resources. These three layers still rely on a repository server on which their codebases are run, and data is physically stored.

The core of the architecture is complemented with two other components to provide two additional services. First, an FTP service to enable massive data transfers and that could be accessible from the web framework in order to allow users to move resources between the repository and their accounts in that service. Second, a Lightweight Directory Access Protocol (LDAP) service where the access credentials of all users are stored and to which the different current and future components of the repository can be connected, thus enabling a centralised authentication point for the whole repository.

---

<sup>4</sup> Lightweight PACS for research purposes, as opposed to full PACS for clinical environments (see 3.2.4 of D5.1)

## 5. Platform implementation

This section details the stack of software tools used to implement the Y2 version of the platform from the extended architecture described in Section 4, the capabilities of the underlying hardware infrastructure, and a review of the main features offered by the latest update of the SINFONIA repository, released in January 2023 on top of the aforementioned Y2 version, and patched with some bugfixes in March 2023.

### 5.1. Software stack

Figure 7 shows the correspondence between the different components of the architecture described in Section 4 and the software tools and programming frameworks used to implement and interconnect those components. Below the reader will find a description of these tools and frameworks and their specific usage within the platform.

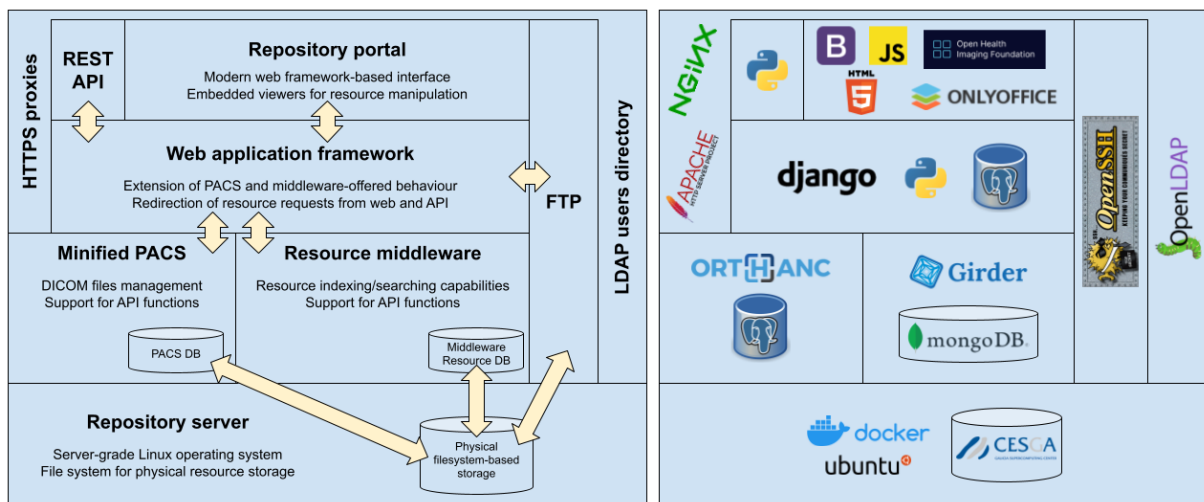


Figure 7: Mapping of the software stack implemented to the extended platform architecture

We will start this description from the bottom of Figure 7. From Y2 version onwards, most of the components of the SINFONIA platform are assembled as Docker images and deployed as containers. Docker<sup>5</sup> is an OS-level virtualization service that enables the delivery of software in packages called containers [4], allowing the automatic deployment of applications to make them work efficiently in different environments. Containers are isolated from one another and bundle their own software, libraries and configuration files, they can communicate with each other through well-defined channels such as ports and sockets [5], and they can store their files both an internal filesystem (kind of volatile, i.e., it disappears if the container is removed) and in directories mounted from the machine on which they are running, either from local or remote file systems. A single machine (either physical or virtual) can run several containers simultaneously, as they share the services of the same underlying operating system kernel [6]. This is a saving-resources alternative with respect to deploying each application in its own virtual machine within a cloud [7]. Containers are deployed from images that are compiled after a recipe called Dockerfile. A Dockerfile is a text document that contains a list of Docker DSL (domain-specific language) instructions to download and install the desired set of applications and assemble them as an image [8]. Users do not usually develop their own images from scratch but base them on previously available images (e.g.: fresh installations of operating systems). Those base images are downloaded from the so-called *registries*, that are servers where images are stored in and distributed from [9]. There are both public (e.g.: Docker Hub<sup>6</sup>) and private registries. The core of Docker

<sup>5</sup> <https://www.docker.com/>

<sup>6</sup> <https://hub.docker.com/>

(named Docker Engine) is licensed under the Apache License 2.0<sup>7</sup>. Dockerfiles, Docker images, and all the software they may internally install and deploy in the form of a container are individually licensed by their own distributors or developers.

Figure 8 is an unfolded version of Figure 7 on which the internal stacks of the components are revealed and the dependencies between them have been explicitly outlined. The boxes with red dashed lines represent the Docker containers that make up the platform implementation, with the circles that receive arrows represent port-based endpoints of those containers. All the containers and their dependencies are defined by means of a Docker Compose YAML file. YAML<sup>8</sup> is a human-readable format used for configuration files and data exchange. It is designed for simplicity and readability, using indentation to define data structure and a minimal amount of punctuation. It is commonly used with programming languages and containerization technologies like Docker. Docker Compose<sup>9</sup> is a tool for defining and running multi-container Docker applications by means of files of such type, and it allows to manage the whole application lifecycle (build, start, stop, log, etc.) through simple commands. In the following subsections the reader will find a description of the architectural components from Figure 7 and how they are implemented by means of the containers highlighted in Figure 8.

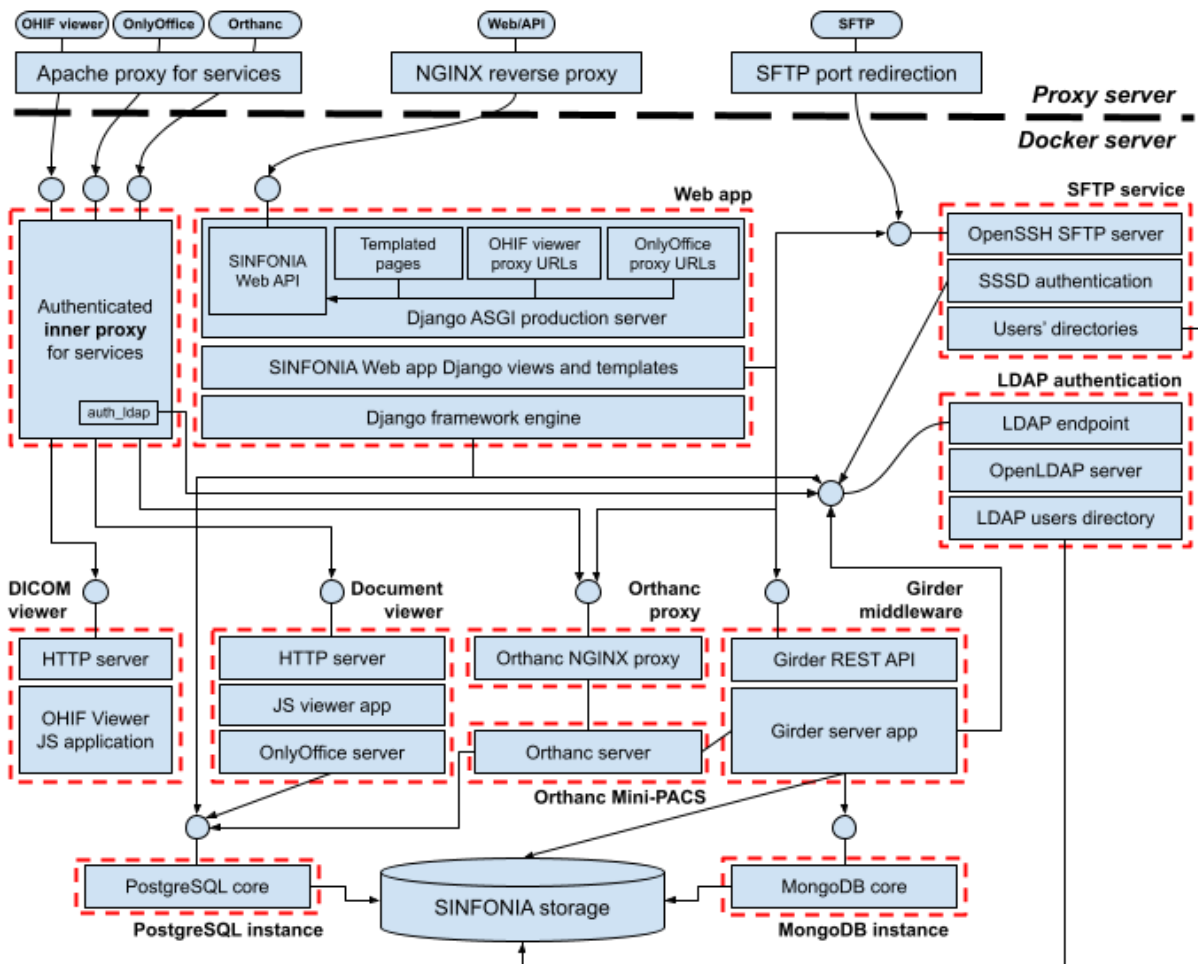


Figure 8: Unfolded diagram of platform components' implementation

<sup>7</sup> <https://www.apache.org/licenses/LICENSE-2.0>

<sup>8</sup> <https://yaml.org/>

<sup>9</sup> <https://docs.docker.com/compose/>

### 5.1.1. Girder resource management middleware

In this extended version of the architecture, the customised Girder<sup>10</sup> deployed for the Y1 prototype is still part of the core of the platform. It continues to play the role of resource management middleware keeping only its Python server-side application, which executes the operations requested through the Girder API. By design, Girder relies on a MongoDB<sup>11</sup> database on which the resources uploaded to the repository are indexed within Collections, Folders and Filesets, along with information regarding Users, Groups and Permissions. Because of that, Girder and MongoDB are depicted together in the box of the right block diagram of Figure 7, representing the software parts of the resource management middleware. The reader is encouraged to review D5.1 to refresh the main features of Girder, its association with MongoDB, the reasons for choosing Girder to develop the Y1 prototype on top of it, and the tweaks and customisations we made for such deployment. Keeping the server component of Girder enables us to implement the extended architecture on top of the existing infrastructure, without having to find an alternative for supporting the information model described in D5.1 and indexing the resources already uploaded by users during the life of the Y1 prototype.

The Docker container that implements the “Resource management middleware” component is referred to in Figure 8 as “Girder middleware”. The image is built after an adaption of the Dockerfile released with the code of Girder 3.1.3, and it is based on a public image of Debian<sup>12</sup> Linux, namely the 10 Buster version, with Python 3.7 and Node.js<sup>13</sup> 12. The container runs our customised version of Girder, with some additional modifications to improve the way its DICOM plugin indexes those files in MongoDB, making it aware that there is an Orthanc instance effectively managing and storing them (hence the line connecting “Girder server app” and “Orthanc server” boxes in the diagram). Node.js is a JavaScript<sup>14</sup> runtime used by Girder for compiling its original JavaScript web application, bundling its code and dependencies, and serving the compiled code via a web server. The Girder web portal is being kept for internal use as it eases administrative tasks such as plugin configuration or error debugging. Internally it is also connected to the LDAP authentication service (by means of a Girder plugin for that), to the MongoDB instance that Girder needs to index resources, and to the physical storage of the repository (the specific path of the storage on which the files are actually kept is mounted as a Docker volume). The container exposes a port that enables the Girder REST API to listen for requests sent by other containers.

### 5.1.2. Orthanc DICOM server

Orthanc<sup>15</sup> is an open-source DICOM server that condenses the functionalities of a PACS into a lightweight implementation (i.e., a “Minified PACS”, as referred to on the left-hand side of Figure 7) that is not subject to the strict medical-device requirements and certifications of other equivalent systems intended for clinical practice in healthcare centres. This makes Orthanc especially useful for research environments such as the SINFONIA project. Orthanc implements the DICOM standard [10] and its “Model of the Real World” [11] already described in Section 4.2, so it is well suited to better support that part of the information model. Note that the Y1 prototype was barely aware of this part of the model through the Girder DICOM plugin, used to preview images and tags of DICOM files in the Girder web portal. Instead, Orthanc offers, in addition to the low-level DICOM interface required by the standard, its own RESTful to manage DICOM resources programmatically, which eases the interaction with the upper web application framework component that

<sup>10</sup> <https://girder.readthedocs.io/en/v3.1.3/>

<sup>11</sup> <https://www.mongodb.com/docs/manual/introduction/>

<sup>12</sup> <https://www.debian.org/>

<sup>13</sup> <https://nodejs.org/en/about/>

<sup>14</sup> <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

<sup>15</sup> <https://www.orthanc-server.com/static.php?page=about>

will be introduced in the following Section 5.1.3. By default, Orthanc stores all the DICOM files it receives in an internal folder on the underlying filesystem, along with a SQLite<sup>16</sup> database for indexing those files and keeping other ancillary information such as the metadata, the history of changes, and a map of a selected number of DICOM tags per file in order to avoid repeated accesses to the storage [12]. SQLite is an embedded SQL database engine that does not have a separate server process, reading and writing directly to ordinary disk files, each containing a complete SQL database with tables, indices, triggers, and views. There are plugins to replace both the filesystem-based storage and the SQLite-based index with PostgreSQL [13] and MySQL/MariaDB databases [14]. Orthanc also implements the DICOMweb standard<sup>17</sup>, thereby offering services for retrieve, store, and search for DICOM resources by means of a RESTful API. This enables the option to connect external components that expect DICOMweb-based endpoints, such as a DICOM web viewer able to supersede the Girder-based previewer of the Y1 prototype, or other PACS or similar servers already managed by the project partners.

“Orthanc Mini-PACS” is the container implementing the “Minified PACS” component. It runs an instance of Orthanc 1.11.2 with the 4.0 version of the PostgreSQL plugin configured to enable both indexing and storage of DICOM files by means of an instance of that DBMS (database management system), to which it is internally connected. Instead of exposing ports directly to accept connections to their APIs (DICOM, DICOMweb, Orthanc REST API), it must be put behind a proxy for enabling security measures required by the OHIF viewer (this component is presented later in Section 5.1.4) in order to allow cross-origin resource sharing (CORS). The proxy is implemented by means of a NGINX 1.11 Alpine Linux-based instance referred to as “Orthanc Proxy” in Figure 8. This also gives more flexibility for introducing authentication or load balancing mechanisms if some of the APIs are publicly exposed in the future.

### 5.1.3. Django web application framework

Regarding the web application framework, we opted to use Django<sup>18</sup>. Django is a free and open-source, Python-based web framework with a comprehensive documentation and a large user base that allows developers to take web applications from concept to launch in a very short time [15]. Django actually follows an MTV (Model-Template-Views) architecture, which implies slight internal differences with respect to the separation between the layers of a pure MVC architecture. However, at this level of review we can consider that it practically follows the latter [16], thus fitting in our design. Thus, the Controller is composed of a set of functions called “views” (the V in MTV) that are executed in response to requests received by the server, usually as a consequence of user interactions with the application's web pages. The View part of the MVC model corresponds to the templates (the T in MTV). These templates are written in HTML5 complemented with Django's own tag language that facilitates the embedding of objects from the views. Although the MVC/MTV design pattern specifies that the application logic must remain in the Controller/View, it is possible to relieve the server of some of this burden by including JavaScript code in the pages, so that it is this code which transforms the data received, or sends asynchronous requests that do not return a complete rendered page but simply an object with the information to be displayed on the client side.

In the SINFONIA web application, the Django views, playing the role of the MVC controller, build and send requests to the Girder and Orthanc APIs, which may respond with both JSON objects (information to be displayed, in general) and binary objects (files to be downloaded or previewed, for example). These objects (either as they arrive, or transformed/combined somehow), can be used by templates to generate a page to be displayed in the browser, or sent directly in response to an asynchronous request from the JavaScript code

---

<sup>16</sup> <https://www.sqlite.org/about.html>

<sup>17</sup> <https://www.dicomstandard.org/using/dicomweb>

<sup>18</sup> <https://www.djangoproject.com/>

of a page already loaded on the client-side. In terms of user experience and look and feel, the combination of JavaScript code, CSS<sup>19</sup> stylesheets and the Bootstrap<sup>20</sup> web frontend framework for the inclusion of fonts, icons, buttons, forms, pop-up windows, etc. provides the portal with a fast, clean, and modern graphical interface.

The Django web application is run inside the “Web app” container. It extends the public Python 3.8 image (based on Debian 11 Bullseye) by preparing a Django 3.2 environment, downloading the code of the SINFONIA portal web application from a private repository, and installing all the Python dependencies needed to run the app. Some of these dependencies are the Django plugins needed to allow the Django engine to use PostgreSQL as its backend database and to validate credentials against and perform CRUD (create, read, update, delete) operations on entries of the LDAP users directory. Since the role of model (from MVT/MVC pattern) of the application is actually played by Girder and Orthanc, Django views (controllers in MVC) must communicate with both services through the ports they respectively expose. Some views may interact also with the “SFTP server” container, which will be introduced later, as resource movements between Girder/Orthanc and the SFTP server must be explicitly done by means of the web portal. Django's default web server is not optimised for production environments, lacking important features such as security and performance enhancements. In production environments, it is recommended to put Django behind a proxy (NGINX reverse proxy in the diagram), along with a production-ready server gateway interface. From the January 2023 update onwards we opted for the 4.0.0 version of Daphne<sup>21</sup>, an open-source ASGI (asynchronous server gateway interface) server release under the BSD licence that enables the implementation of interesting asynchronous features such as streamed file downloading.

### 5.1.4. Embedded web viewers

As a solution to the shortcomings and instabilities that Y1 prototype users reported with relation to the previewer of the Girder DICOM plugin, one of the measures<sup>22</sup> taken is the inclusion of an external DICOM viewer from this Y2 version onwards. The Orthanc documentation provides a list of free and open-source viewers that are at least compatible with the low-level Query/Retrieve service of the DICOM standard, most of them being also compatible with the DICOMweb high-level API. Among the latter, OHIF Viewer<sup>23</sup> was chosen. It is a web-based viewer written in JavaScript that can be configured to query DICOMweb endpoints and easily served as a web page with an URL that points directly to a given DICOM study. Moreover, it has maintained extensions for viewing, annotating, and reporting on DICOM images in 2D (slices) and 3D (volumes). It is an initiative led by the Open Health Imaging Foundation<sup>24</sup>, a foundation launched in 2015 in the USA with the aim of developing an open-source web-based medical imaging platform for the global community. Namely, the OHIF viewer is released under a commercially permissive MIT licence.

OnlyOffice Document Server<sup>25</sup> is an online office suite comprising viewers and editors for texts, spreadsheets, and presentations. It is compatible with popular formats from both commercial (Microsoft Office's OOXML - docx, xlsx, pptx- and Legacy -doc, xls, ppt- formats) and open source (LibreOffice's OpenDocument formats - odt, ods, odp-) office suites, and with other commonly used files such as PDFs, plain text files, HTML pages, EPUB electronic books or CSV files. It is released under different modalities depending on being exploited as a document server for business environments, for integration in commercial software solutions, or as an

<sup>19</sup> <https://developer.mozilla.org/en-US/docs/Web/CSS>

<sup>20</sup> <https://getbootstrap.com/>

<sup>21</sup> <https://pypi.org/project/daphne/>

<sup>22</sup> The other one is to implement a general embedded file previewer, described further in this document

<sup>23</sup> <https://docs.ohif.org/>

<sup>24</sup> <https://ohif.org/about>

<sup>25</sup> <https://www.onlyoffice.com/en/office-suite.aspx>

open-source software solution for free community use. In this last case, a GNU Affero General Public License version 3.0<sup>26</sup> applies. The full integration of the suite in the SINFONIA project would require the development of a specific OnlyOffice connector to enable collaborative editing of Girder-stored documents, which is in itself a complex software project that would have consumed a significant part of the time available for the development of this Y2 iteration of the repository. Because of that, for the moment we include just the web-based OnlyOffice document viewer, which is easily enabled by providing the JavaScript code of the OnlyOffice viewer component with a valid URL to download the file to review. For that purpose, a Django view reacting to such requests and sending them down to the Girder middleware server has been implemented.

Each embedded viewer has its own Docker container to serve its respective JavaScript application. The code of the OHIF viewer is served by the “DICOM viewer” container, which runs the official OHIF viewer Docker image. The code is prepared on build time, whereas on run time there is a NGINX proxy that responds to requests from the user's browser to run the application locally for inspecting the DICOM study whose Study Instance UID is specified in the URL. Regarding the OnlyOffice viewer, it is provided by the “Document viewer” container, which runs the official Docker image of the Community (open-source) Edition of OnlyOffice 7.2 Document Server. Similar to what is described for the OHIF viewer, the OnlyOffice Document Server sends back the code of its document viewer in response to a request from the user's browser to inspect a compatible file. Despite the collaborative editing of documents not being enabled, OnlyOffice nonetheless expects an instance of PostgreSQL to be able to connect to.

Both viewers have a similar workflow, provided that their respective JavaScript applications must be served by the corresponding platform component to be executed locally in the user's browser. In the case of the OHIF viewer, the URL for viewing a specific DICOM study contains its standard-compliant Study Instance UID. In response to the request for that URL, the container sends the code to the browser and it uses that UID to request the images and metadata of the DICOM files to the Orthanc server through its DICOMweb endpoints. Regarding the OnlyOffice viewer, the request to the URL to preview a compatible file is indeed answered by the Django web app, which sends back a template with an external script to make the browser download the JS code from the container. The template also includes a portal-based URL through which the viewer can access the document to be displayed. As the code of the viewers is downloaded from and run in users' browsers, servers of both viewers and Orthanc have to be put behind a proxy, which is implemented by means of the “Inner proxy” container. This container is built after the official image of Ubuntu 18.04, on which Apache 2.4.29 is installed along with the plugins *proxy*, *proxy\_http*, *headers* and *authnz\_ldap*. The proxy is configured to ask for valid credentials and validate them against the LDAP authentication service when a user's browser requests access to either of the two viewers. The three former plugins are needed to properly redirect the traffic to the viewers' containers, whereas the latter one enables the aforementioned credentials validation.

### 5.1.5. Underlying database systems

As mentioned throughout the whole Section 5.1, some of the stack components make use of database systems for indexing and/or storing the resources they manage. One of these systems is MongoDB, which is a NoSQL database program that gathers data in collections of BSON<sup>27</sup> (Binary JSON) objects. BSON differentiates from JSON on extending the supported types and on prefixing large elements with a length field, which makes it more efficient both in storage space and scan-speed [17]. MongoDB is designated as NoSQL in contrast to conventional DBMSs, on which data is manipulated by means of SQL queries. Moreover, NoSQL database systems do not work with fixed structures such as tables, do not normally support *JOIN*

<sup>26</sup> <https://www.gnu.org/licenses/agpl-3.0.en.html>

<sup>27</sup> <https://bsonspec.org/>



operations and do not fully guarantee ACID properties (atomicity, consistency, isolation and durability), which makes them scale well horizontally [18]. Girder is designed to work with a MongoDB instance as its indexing backend, so there was no prior decision process as to which specific DB to use for this purpose.

The other database system widely used by several components of the repository architecture is PostgreSQL, all of which share a common instance of it to work with. PostgreSQL<sup>28</sup> is an open-source object-relational DBMS whose origins date back to 1986 as part of a DARPA-sponsored project, having a core platform with more than three decades of active development. Thus, it has earned a strong reputation for its proven architecture, reliability, data integrity, robust feature set, and extensibility. With relation to Orthanc, the role of both its SQLite indexing database and filesystem-based storage backend have been taken by a PostgreSQL instance by means of the Orthanc official PostgreSQL plugins. These plugins enable a revision mechanism implemented in Orthanc to protect files of race conditions<sup>29</sup> by locking and serialising concurrent modifications. OnlyOffice merely requires an active PostgreSQL instance to be deployed, although the database is used intensively only when exploiting collaborative document editing. In a similar vein, the model of a Django-based web application is supported by default by a SQLite database. In this case we are not implementing the SINFONIA information model by means of Django but by leveraging the existing models offered by Girder (non-DICOM resources, data organisation, users, and access management) and Orthanc (DICOM resources). In principle, since it will hardly be used (some basic functionalities of Django and its plugins require the existence of a backend database), it was feasible to keep SQLite. However, taking into account the special orientation of PostgreSQL to concurrent environments [19] (and a multi-user web application is a clear example of this type of environment), we have opted to use the PostgreSQL instance as the backend database for Django from the outset. This allows the information model to be extended beyond the abstractions supported by Girder's and Orthanc's models, if needed in future iterations.

The instances of both databases are run in containers built after their respective official Docker images, namely PostgreSQL 14.5 and MongoDB 4.4. They just expose the default ports for allowing connections from Django, OnlyOffice and Orthanc in case of the PostgreSQL instance, and from Girder in case of the MongoDB one. Both instances are configured to store their indexes and data in the SINFONIA repository storage.

### 5.1.6. SFTP service

OpenSSH<sup>30</sup> is a set of open-source software tools that provides secure connectivity through cryptographic protocols for remote networks. One of the tools that OpenSSH provides is a SFTP (Secure File Transfer Protocol) server, which allows for massive secure files transfer over a network. The main advantages of SFTP are the protection of data in transit by means of encryption, its speed for transferring large and/or multiple files simultaneously, its seamless integration with VPNs and firewalls and the possibility of accessing servers through friendly interfaces such as web portals or SFTP GUI-based clients [20].

In our case, we have configured an OpenSSH-based SFTP server that sets a separate directory tree for each user of the repository, so that any attempt by a user to access files or directories outside of their own tree will fail, as they do not have the permissions to access anything beyond its own space in the SFTP server. Users can connect to the SFTP server using either GUI clients such as FileZilla<sup>31</sup> or WinSCP<sup>32</sup>, or the command-line tools that operating systems usually have for this purpose. The container exposes a port to allow the server to accept these connections. In turn, for security reasons, external connections to any other OpenSSH-

<sup>28</sup> <https://www.postgresql.org/about/>

<sup>29</sup> [https://en.wikipedia.org/wiki/Race\\_condition#In\\_software](https://en.wikipedia.org/wiki/Race_condition#In_software)

<sup>30</sup> <https://www.openssh.com/>

<sup>31</sup> <https://filezilla-project.org/>

<sup>32</sup> <https://winscp.net/>

based service than SFTP (e.g., SSH service for accessing the underlying machine's command prompt) are not allowed.

The FTP server is implemented by means of a namesake container built after the official Debian 10 Buster image. This server runs a OpenSSH 7.9 instance restricted to provide just the SFTP service. The OpenSSH's SFTP implementation is compliant with SFTPV3 [21] and bundles AES-256 encryption and SHA-256 hashing. Instead of maintaining the SINFONIA user accounts manually, an SSSD<sup>33</sup> (System Security Services Daemon) client is used. SSSD is a collection of daemons that handle authentication, authorization, and user and group information from a variety of sources such as LDAP. SSSD provides PAM (Pluggable Authentication Modules) and NSS (Name Service Switch) modules to allow users in the directory to be recognized as valid users of a system [22]. Thus, the service is configured to create an isolated environment when a valid user connects to the FTP client of their choice, so that the user can only upload to, download from, and browse its own folders. Although everything is ultimately held in the SINFONIA repository storage, it has to be noted these FTP user directories are not coupled with the general web application storage indexed by Girder and Orthanc. As will be explained later in Section 5.3.4, the web interface implements a function for the user to browse his FTP directory and copy the desired resources to the general storage of the web application. Similarly, if a user wishes to use the FTP service to download a large volume of resources to his computer, he must first copy them from the general storage to his FTP directory.

### 5.1.7. LDAP users' directory

All the authorization and authentication operations within the repository ultimately rely on an instance of a OpenLDAP<sup>34</sup> directory. OpenLDAP is an open-source implementation of the Lightweight Directory Access protocol (LDAP), which is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over a network [23]. One of the most common uses for LDAP is to provide a centralised storage for access credentials, so that different applications and services could connect to the LDAP server to validate users [24]. All repository services for which it has been stated that authentication is required, check the validity of the credentials provided against an OpenLDAP server deployed within the platform.

This service is provided by the "LDAP authentication" container. It is built on top of the version 1.5.0 of the "Osixia Docker image for OpenLDAP"<sup>35</sup>, which is distributed under a commercially permissive MIT licence. This container runs an OpenLDAP 2.4.57 server that exposes the LDAP standard port to the rest of Docker containers, allowing them to validate credentials against it. The SINFONIA user directory is ultimately maintained by this container (i.e., any changes made by any other component of the repository must propagate to this directory), being physically kept in the SINFONIA repository storage.

### 5.1.8. External access control

As the reader has presumably inferred from Figure 8, there are three components that are not in Docker container form and live in the so-called "Proxy server" machine, in contrast to all other components, that are Docker containers (outlined with red dashed lines) running inside the "Docker server" machine.

The first of these three components is the NGINX<sup>36</sup> reverse proxy. As for the Y1 prototype, an instance of NGINX is deployed to act as a reverse-proxy server. Recalling the description provided in D5.1, a reverse proxy is a server put in front of another one to retrieve resources or attend to requests on behalf of the hidden

<sup>33</sup> <https://sssd.io/>

<sup>34</sup> <https://www.openldap.org/>

<sup>35</sup> <https://github.com/osixia/docker-openldap>

<sup>36</sup> <https://www.nginx.com/>

one. In the Y1 prototype, the reverse-proxy was redirecting the requests to the Girder API. In turn, in this new extended version of the architecture, the NGINX instance is acting as a reverse proxy for accessing the Django-based web portal (and sending requests to the API endpoints when available).

The second one is the “Apache external proxy”. As described in Section 5.1.4, in the Y2 version there are new services which must be somehow accessible from the Internet too: the OHIF viewer, the Orthanc server and the OnlyOffice document server. Regarding the viewers, this requirement is determined by the need to download from and run their code locally in the user's browser. Moreover, as the OHIF viewer needs to submit requests to Orthanc's DICOMweb terminals in order to be able to access the study to display, the Orthanc server must be also somehow exposed. Regarding the OnlyOffice server, the viewer code has to be also downloaded to the user's browser, whereas the document to be displayed is accessed through a portal-based URL, and the web portal is already located behind the aforementioned NGINX proxy. The internal Apache proxy is configured to request valid credentials to users in order to prevent unauthorised accesses. However, the internal proxy is not directly exposed on the Internet, but this “Apache external proxy” has been deployed in front of it and at the same level as the NGINX proxy of the web portal. Thus, public requests for both OHIF (for the viewer itself and for Orthanc-managed DICOM studies) and OnlyOffice viewers are not sent directly to these components but captured and redirected by means of the external proxy. This prevents the authentication service from having to be directly accessible from the same network on which the external proxies are running.

The reason for putting these proxies between the platform and the Internet remains the same as for the Y1 prototype: improving the security of the whole repository infrastructure by (1) encrypting the connections under the HTTPS protocol, (2) dismissing requests from allegedly malicious IP addresses previously included in an access control list, and (3) avoiding the direct public exposure of their components.

Moreover, there is the third non-containerized component, “SFTP service endpoint”, which is just a redirection from a public port pair in the proxy machine to the SFTP port exposed by the Docker container. Let us remember that the SFTP server has its own security measures: it is configured to operate as an SFTP server only for LDAP authenticated users and does not accept external SSH connections to gain access to its command prompt.

## 5.2. Hardware platform

The Docker Engine that hosts all the containers runs in the “Docker server” machine, a virtual machine (VM) created for this purpose in CESGA's OpenStack-based cloud infrastructure<sup>37</sup> made up of 114 compute nodes with 2 Intel Xeon Gold 6240R @ 2.40 GHz 24-core physical CPUs. The machine has 8 virtual CPUs and mounts a virtual hard drive of 60GB and 32GB of virtual RAM. The operating system is Ubuntu Linux Server 20.04.5 LTS Focal Fossa. Every mention of the dependencies the containers have with relation to the SINFONIA repository storage refers to the SINFONIA's 200GB partition in CESGA's NetApp FAS9000 storage cabin<sup>38</sup>. This partition is mounted in the machine through NFS (Network File System), which is a distributed file system protocol that allows remote clients to access files over a network as if they were on a local file system. Daily incremental and weekly full backups are performed for the data stored in that partition (as part of the automated backup policy of CESGA storage infrastructure) along with manual backups of the virtual hard drive of the VM performed after relevant changes in its configuration. The machine is attached to a SINFONIA

---

<sup>37</sup> <https://cesga-docs.gitlab.io/cloud-openstack/index.html>

<sup>38</sup> <https://www.cesga.es/en/infrastructures/storage/>

private subnet within the private network of the cloud platform, so that it cannot be directly reached from either the Internet or any other CESGA cloud-based subnet.

The two proxies and the SFTP port redirection are set into the separate “Proxy server” machine. This one is also a virtual machine (VM) in the aforementioned CESGA’s cloud infrastructure, with 8 virtual CPUs, a virtual hard drive of 20GB, and 16GB of virtual RAM. The operating system is Ubuntu Linux Server 18.04.6 LTS Bionic Beaver. Neither the proxies nor the configured SFTP redirection are dependent on the SINFONIA storage. Regarding the network interfaces, this machine plays the role of a bridge between the Internet and the repository services. It is connected to the Internet through an interface with a public IP mapped to the *sinfonia.cesga.es* domain name. A firewall prevents other connections than those to the “Web/API”, “OHIF viewer”, “Orthanc” and “OnlyOffice” endpoints, on which the proxies are listening and redirecting the traffic, and to the “SFTP” endpoint, properly redirected to the inner endpoint of the LDAP-authenticated SFTP server. Those redirections are routed to the “Docker server” machine through another interface connected to the SINFONIA private network. As for the “Docker server” machine, manual backups of the virtual hard drive of the VM performed after relevant changes in its configuration.

### 5.3. Update on repository features

One of the main objectives in mind when designing the user interface of the Y2 version (and consequently the basis for the next ones) is to maintain as much as possible both the workflow and the look-and-feel of the original Girder interface used in the Y1 prototype. This offers users a much less steep learning curve when jumping between these two versions. Thus, it is assumed that the reader is familiar with the operation of the prototype, otherwise it is strongly recommended to read Section 5.4 “Prototype features” of D5.1 before the reader moves on, as most of the following feature descriptions below will rely on comparisons with those of the Y1 interface as they were explained in that section of D5.1.

#### 5.3.1. Homepage and interface layout

The homepage of the web portal, shown in Figure 9, still welcomes visitors with a message informing them that they are browsing anonymously, so they must either log into the web to continue or contact the administrators to ask for an account. The general layout of the web portal keeps the header with SINFONIA logo and title (the full project name is shown only in the homepage), an actions menu in the left, and a footer with some useful links (“About SINFONIA” towards the official web of the project<sup>39</sup>, “Contact” for reaching the repository administrators, “Privacy Policy” towards the document containing the SINFONIA Data Repository Privacy Policy, and “Cookie Settings” to review their cookie preferences) and the mandatory reference to the EU research program and the Grant Agreement number of the project.



Figure 9: Screenshot of homepage for anonymous users

<sup>39</sup> <https://sinfonia-appraisal.eu>

The “Log In” button is kept on the top right-hand corner. If valid credentials are provided, the landing page is shown and this button becomes a dropdown with options to log out, to browse “My Folder” (user’s personal collection), to go to the user’s profile page, or to toggle the “Dark Mode” (a mode designed for dim light situations by displaying light texts on a dark background, in contrast with the conventional “Light Mode”) of the web interface. Besides the messages about being successfully logged and recommending the reading of the “Anonymization Guidelines”, a couple of new hooks are provided to guide users in their first steps in the portal: links to the “Dashboard” (list of repository collections with access permissions) and to the “FTP page” (embedded browser of user’s FTP account). Along with the links to sections “Users” and “Groups” shown to registered users, an entry for the “FTP” zone has been added to the left menu. Figure 10 shows this new version of the landing page and general layout, with the user actions dropdown unfolded and the “Dark Mode” toggled.

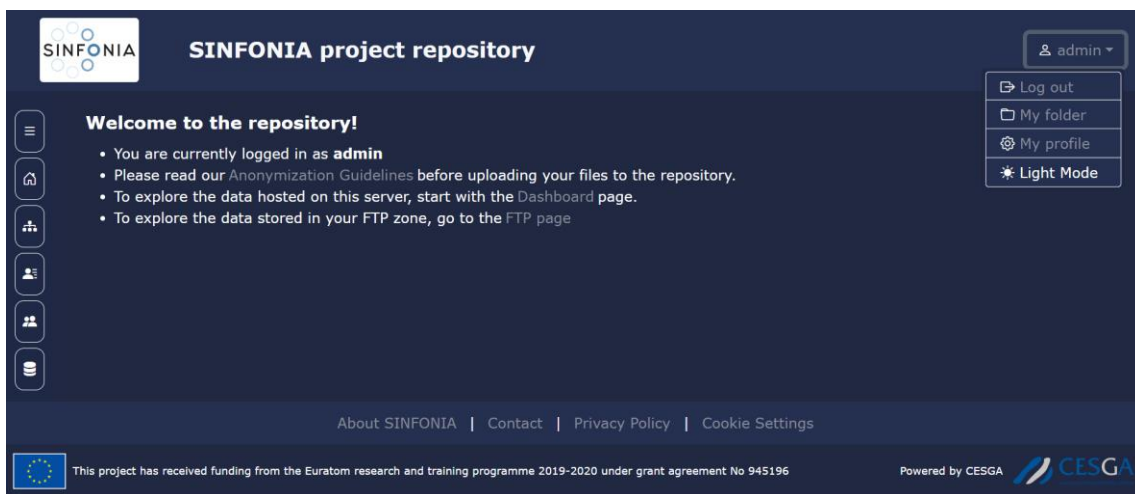


Figure 10: Screenshot of homepage for authenticated users with dark mode and user menu active

Let us notice that for logged-in users, the “Contact” link in the footer now leads to a page with a form (Figure 11) on which they can write a message for the portal administrators requesting for “Technical support”, or to notify “Privacy issues” in relation to any resource stored in the repository that is not properly anonymised. The text of the message can be enriched with screenshots or other relevant files the user may want to provide. As seen in Figure 9, the “Contact” link is shown to anonymous users too, but in that case, it leads to a page informing of the [sinfonia@cesga.es](mailto:sinfonia@cesga.es) support outlet. This distinction is a matter of trust in users for security reasons. It is assumed that at this stage of the project, with the repository still private to the SINFONIA community and registration upon request to the administrators, we should not have malicious users within the registered ones.



Figure 11: Screenshot of contact page for authenticated users

Users can see and edit some details of their accounts by clicking on the “My profile” entry of the user actions dropdown. In that page (Figure 12) users can edit their profile details, change their password, and activate an app-based 2FA (2-factor authentication) mechanism. The tab for setting keys to grant access through the API, as well as the link that was in the Y1 prototype’s footer to read the Girder API documentation, will be restored in future increments once the AI algorithm execution environment is integrated and a proper SINFONIA API is ready for use.

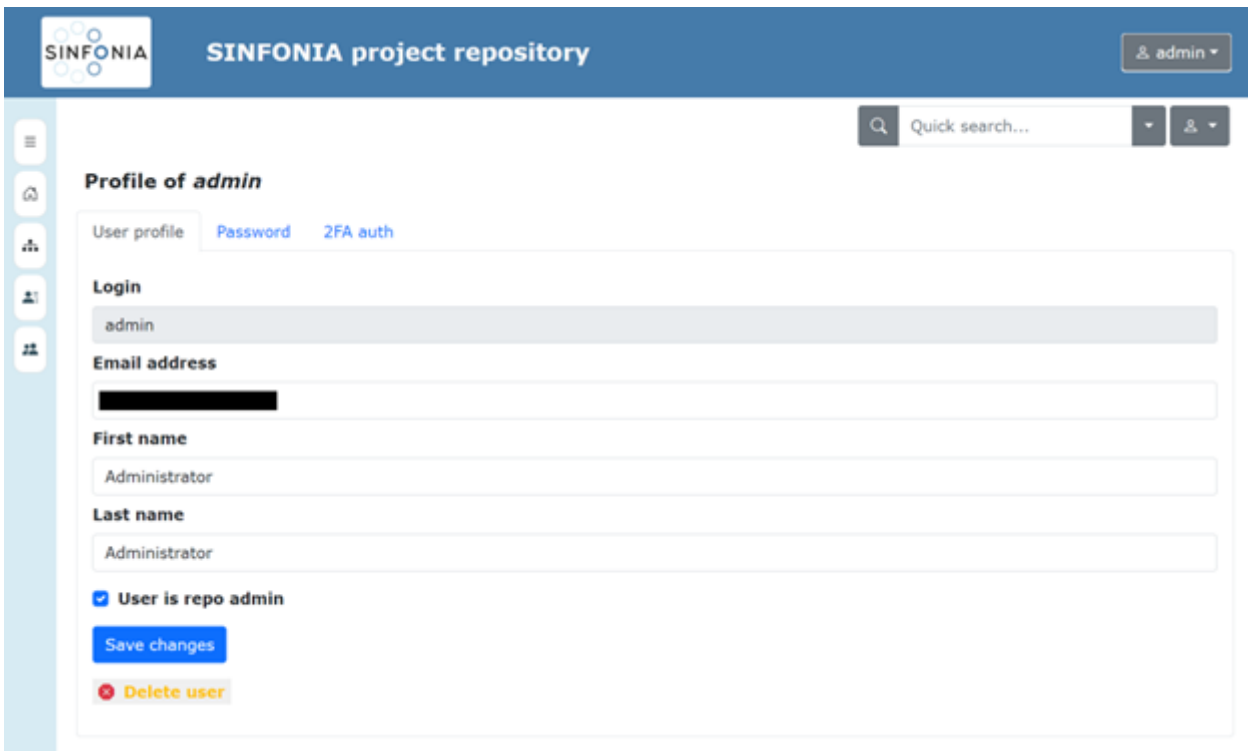


Figure 12: Screenshot of "My profile" page for "admin" user

### 5.3.2. Resources organisation, navigation, and operations

Data storage keeps its original organisation in separate collections for each WP of the project, as Figure 13 shows. This page can be accessed by clicking either on the aforementioned “Dashboard” link of the landing page or on the namesake button from the left menu.

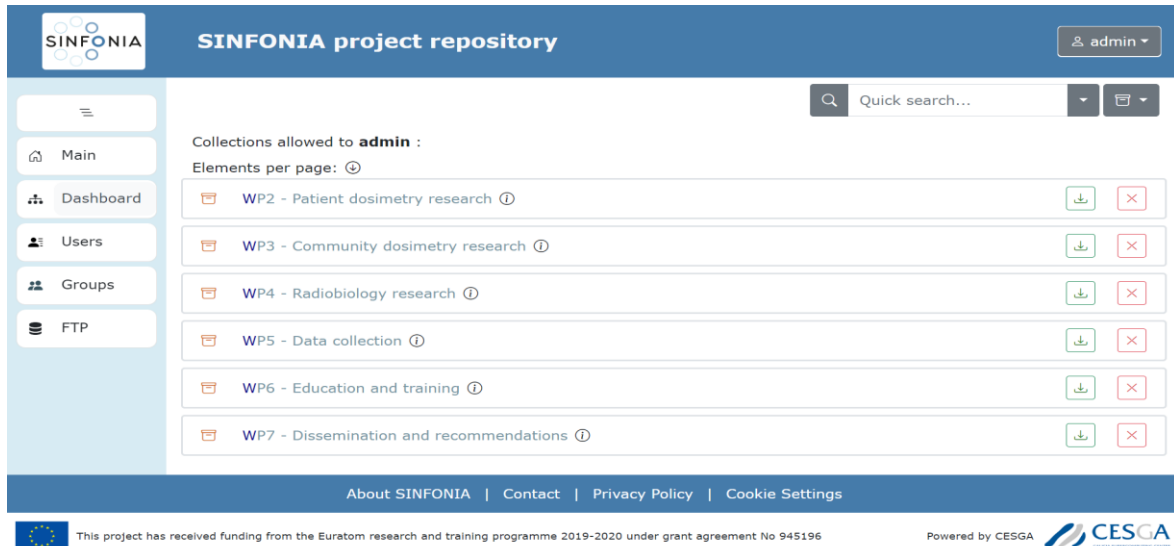


Figure 13: Screenshot of Collections page for authenticated users

Figure 14 depicts the general appearance of a resource page, a folder in this case. Apart from the visual differences with respect to the equivalent page in Girdler, most of the components remain the same. A breadcrumb, the quick search box, and a set of three buttons (upload, permissions, resource actions) can be seen in an upper bar at the top of the page. The last button (the one with a folder icon) is indeed a dropdown menu (unfolded in the figure) with relevant actions for the type of resource visited.

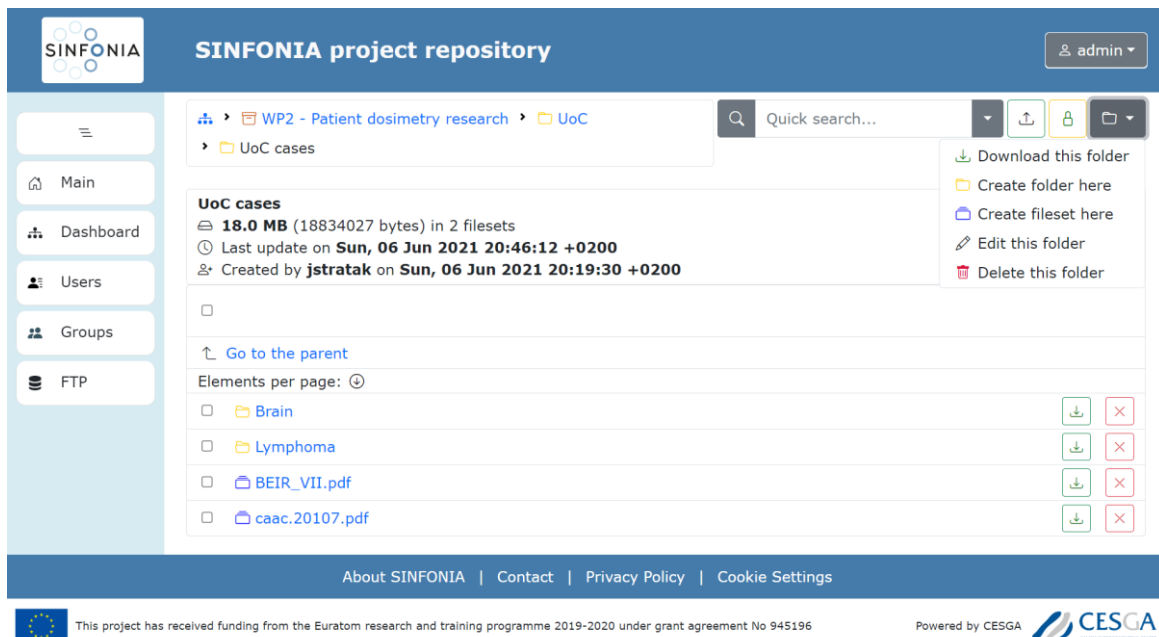


Figure 14: Screenshot of Folder page with Folder Operations menu unfolded on the right

Regarding the resources permissions and its setting by means of access control lists no major changes are included in this version, with the button, the dialog and the underlying process remaining virtually the same. With respect to the “Edit” feature, besides modifying name and description, in this version users can also assign SINFONIA IDs to resources, as Figure 15 shows. Notice the general and specific parts of the ID. This feature is also available when creating a folder or item, not just when editing.

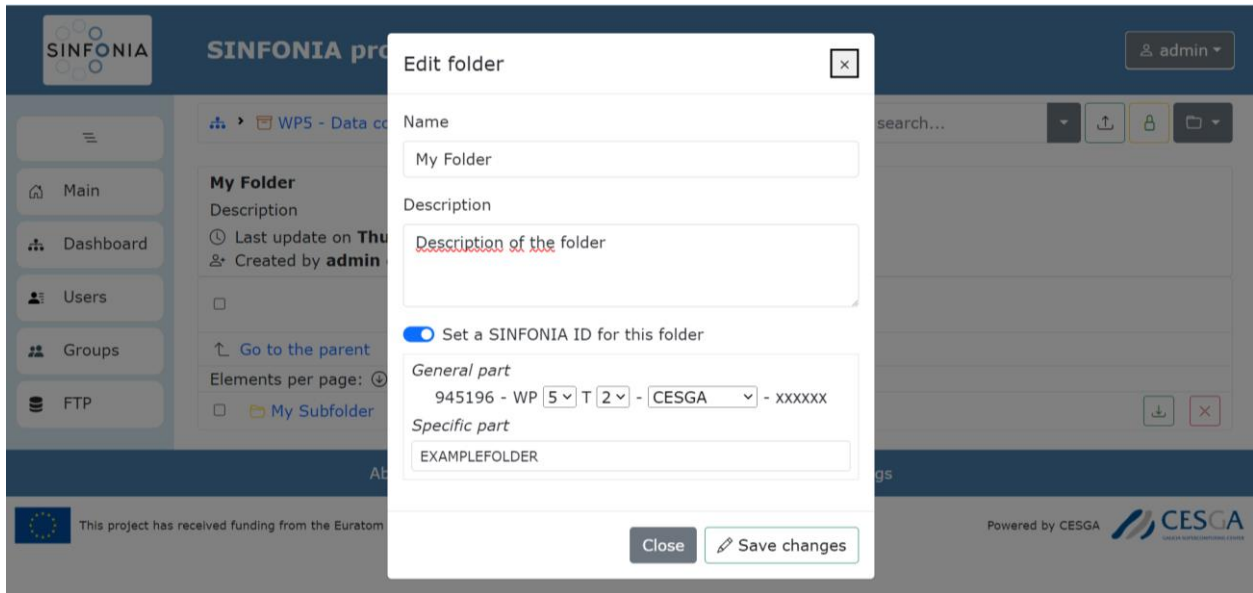


Figure 15: Screenshot of “Edit folder” dialog

Below the upper bar with the breadcrumb, the search box, and the resource action buttons, a panel with basic information of the resource is shown. This panel (Figure 16) replaces and extends the dialog displayed in Girder by means of its blue “Information” button, which does not appear anymore next to the upload and permission ones.

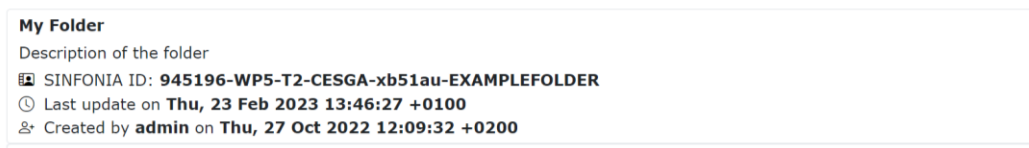


Figure 16: Screenshot detailing a Folder information panel

Most of the space on resource pages is taken up by the list of contents. Both the basic information panel and the list of contents are loaded dynamically, which is a major new feature of the web interface of this Y2 increment. This means that full pages are not being served when browsing the resources, but rather the information of the resource to be visited is requested and the content is updated in the user's browser, which significantly reduces the load in the interaction with the server. This dynamic behaviour has also enabled the pagination of resource lists, which is another new feature of this version.

For each entry users can perform actions like open the element (by clicking on its name), download or delete it (by clicking on the respective buttons on the right of the entry). These actions can be done also through a new contextual menu (Figure 17) implemented for this version of the interface. Users can reach this menu by right-clicking on the corresponding element entry.



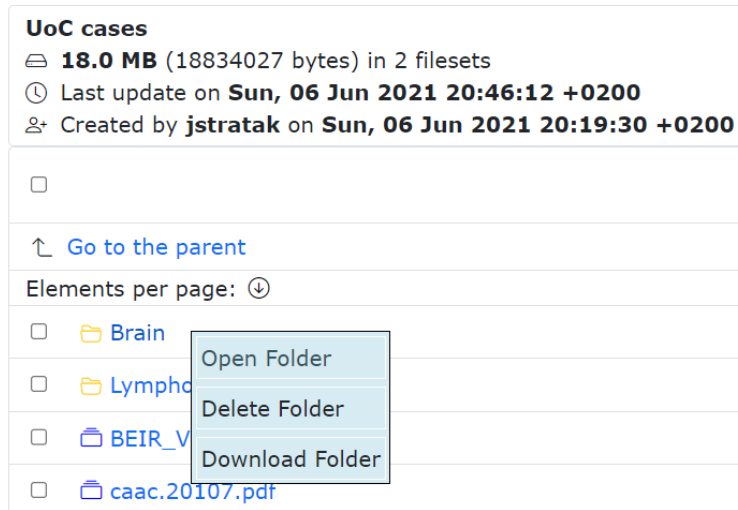


Figure 17: Screenshot detailing the basic contextual menu

Selection of resources is still done by clicking on the checkboxes on the left of the list. However, in this new version every checked resource automatically gets picked for eventual copy or move operations (in the original Girder interface resources must be checked first and then picked explicitly). The number and type of resources selected and picked are summarised in a blue button next to the “Select all” checkbox. When clicking on this button, an “Elements currently picked” dialog appears (Figure 18) with the list of resources classified by type, each with a button for unpicking it on the right. Moreover, the user can also pack and download the whole selection, delete it from the repository, or transfer it to its FTP account.

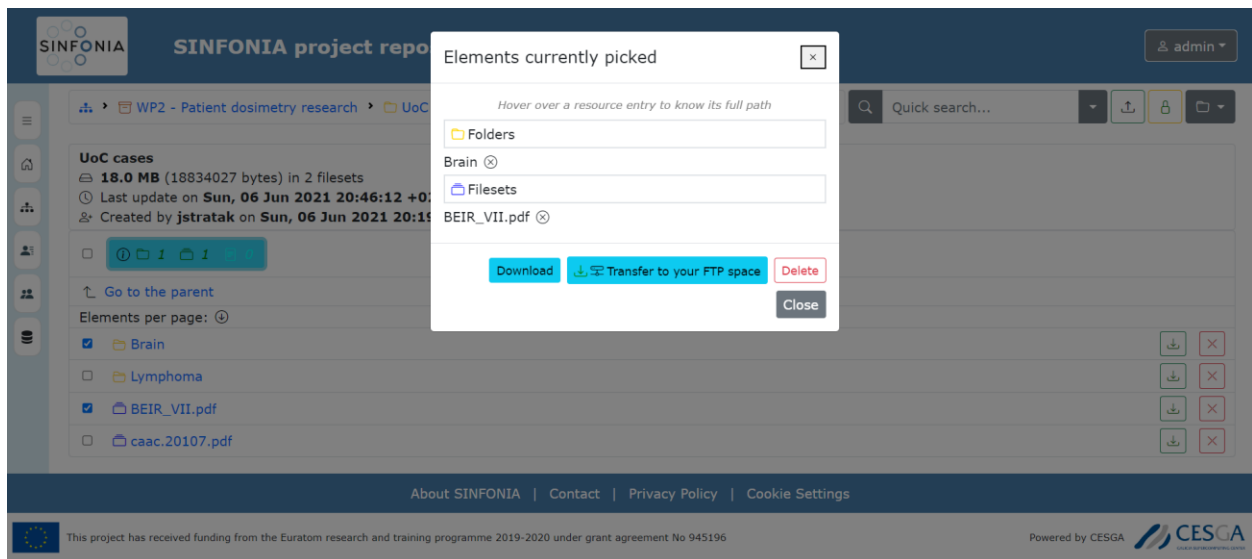


Figure 18: Screenshot with an “Elements currently picked” dialog

To move or copy the elements picked, the user has to go to the page of the destination resource and then right-click anywhere to reveal another version of the contextual menu (Figure 19), that will include the “Copy here” and “Move here” actions. Users can also back out and clear the selection of items with the action “Clear picked elements”.

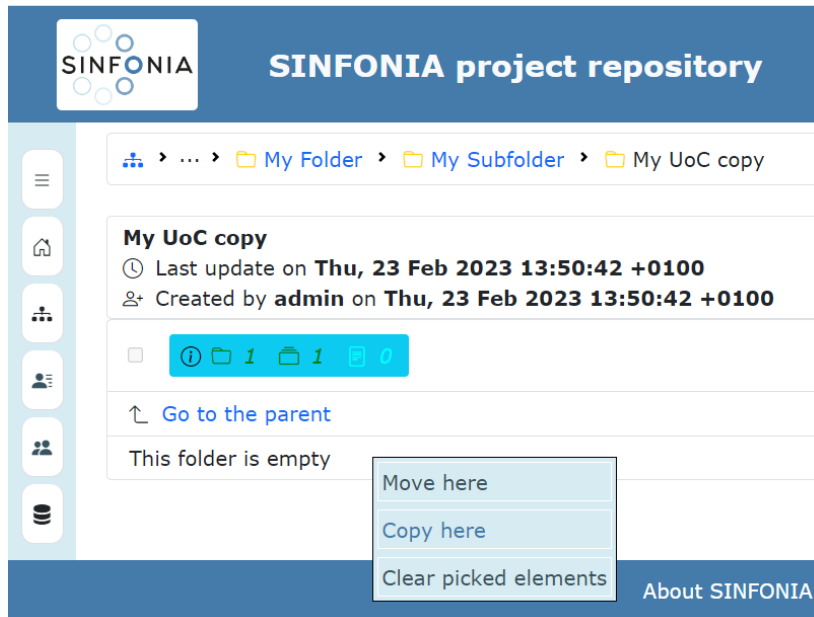


Figure 19: Screenshot detailing a Move/Copy/Clear contextual menu

A resource can be moved also by dragging its entry over and dropping it onto the destination's one (e.g., moving a fileset from a folder to a subfolder, as shown in Figure 20) or onto the "Go to the parent link" to move it to its parent resource. A plus sign (circled in red in the figure) will appear under the cursor in case the destination is valid for the resource to be moved.



Figure 20: Screenshot detailing a drag-and-drop resource movement

As in the Y1 prototype, files can be uploaded by clicking on the upload button in the upper bar. However, the upload dialog and its underlying process have undergone changes that are worthwhile to mention. As shown in Figure 21, when uploading files to a folder users can choose if they want to pack all the files in a new fileset (which must be given a name) or store them in a separate fileset for each. This latter option is disabled when uploading several files to a fileset, as only files can be contained in a fileset. After confirming the upload, a background process starts, so that the user is able to continue browsing the portal. A stack of notification toasts with spinners is kept in the bottom right-hand corner, with the file names disappearing as their uploads are completed. A similar stack is generated for downloading resources once a user invokes the process (e.g., by selecting several resources and clicking on download in the "Elements currently picked").

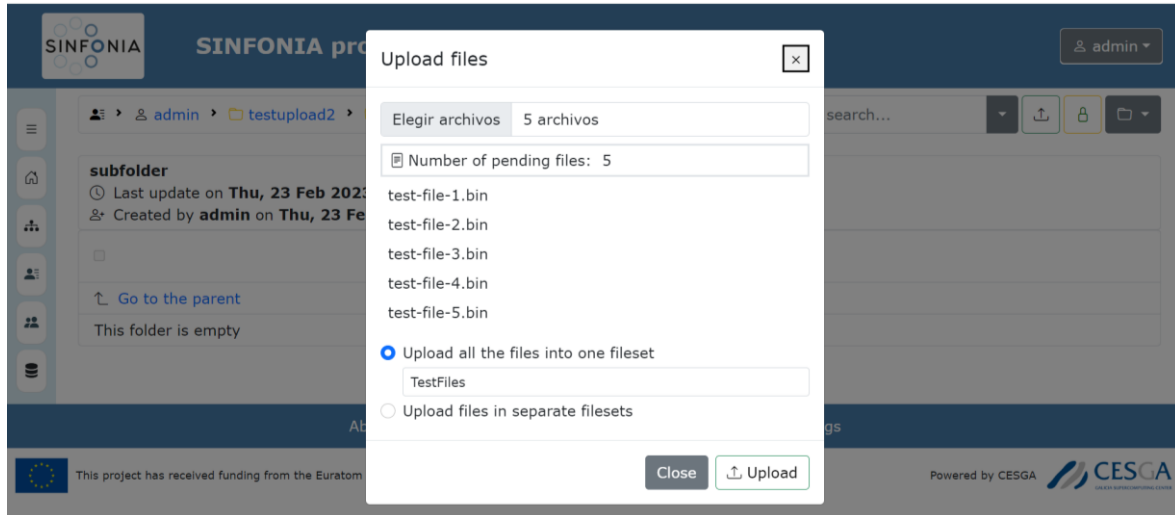


Figure 21: Screenshot with an "Upload files" dialog

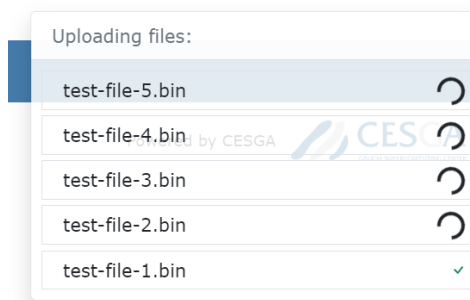
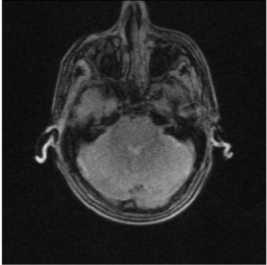


Figure 22: Screenshot detailing a stack of upload notifications

Let us jump now to the fileset page, which is another part of the portal that has received a significant number of updates. One of these major improvements is the embedding of a general previewer, inspired on the DICOM one from Girder but extended to render many more types of resources. Figure 23 shows the page of a fileset mixing DICOM and non-DICOM files. By default, the previewer displays the first entry in the file list. If a user clicks on the "Eye" button of any file entry in the list, the component responds by adapting to the particular type of resource to be previewed. For illustration purposes, both the entry previewed and its corresponding "Eye" button are highlighted in red in the figure.

When dealing with DICOM files, the previewer is meant to mimic the appearance of its Girder counterpart (it shows a snapshot of the image contained in the DICOM file and its tag list) but modifying its internal behaviour to improve the user experience in response to users' feedback. Namely, both the snapshot and the tag list are ultimately served by Orthanc in a much more lightweight form. The snapshot is a PNG image (not a browser-run render anymore) and the tag list is generated after a JSON object (not by means of a browser-run DICOM headers parsing to extract them). If the user clicks on the button "Open study in external viewer" placed under the snapshot, a new tab with an instance of the external OHIF DICOM viewer will be open, showing the study of the instance contained in that DICOM file (Figure 24). This button makes the browser ask the "OHIF viewer" endpoint of the Apache proxy for services (see Section 5.1.8) for downloading the JS code of the viewer for running it in the user's browser. The viewer will send a DICOMweb request for loading the study through the Orthanc endpoint of the proxy.

**My Mixed Fileset**  
 3.2 MB (3318615 bytes) in 7 files  
 Last update on **Thu, 27 Oct 2022 12:19:56 +0200**  
 Created by **admin** on **Thu, 27 Oct 2022 12:19:56 +0200**



Reverse	Value
0008,0005 SpecificCharacterSet	ISO_IR 100
0008,0008 ImageType	ORIGINAL\PRIMARY\M\ND
0008,0016 SOPClassUID	1.2.840.10008.5.1.4.1.1.4
0008,0018 SOPInstanceUID	1.3.6.1.4.1.9328.50.16.47270848771588000885696
0008,0020 StudyDate	19040718
0008,0023 ContentDate	19040718
0008,0032 AcquisitionTime	074229.459986
0008,0050 AccessionNumber	2819497684894126
0008,0060 Modality	MR

Open study in external viewer

Go to the parent

Elements per page: 20

<input type="checkbox"/>	1-01.dcm	130.1 KB			
<input type="checkbox"/>	20200588 Logo Sinfonia_DEF.png	59.9 KB			
<input type="checkbox"/>	20221027-1 - WP5-Workshop.pdf	2.4 MB			

Figure 23: Screenshot of a fileset page previewing a DICOM file and with its list entry highlighted

Figure 24: Screenshot of an OHIF viewer tab displaying a DICOM study

Images and videos in several usual formats (such as GIF, JPG or PNG for images, and MP4, WebM or Ogg for videos) can be previewed too. The embedded viewer adapts by replacing the snapshot on the left and the tag list on the right with a centred viewport, with simple Play/Pause controls in the latter case. The previewer is also able to display a listing of the hierarchy of directories and files contained in a ZIP file without the need to unzip it for inspection.

Visualisation of office documents, in turn, is not served by this embedded previewer. For this kind of files, a button with a right-pointing arrow (see the third line of the file list from Figure 23) appears instead of the “Eye” one. By clicking on such buttons, a request is sent to the Django app for generating a page that embeds the OnlyOffice viewer (Figure 25) remotely through the proxy along with a JSON object to tell that viewer the portal-based URL it must follow to access the document.

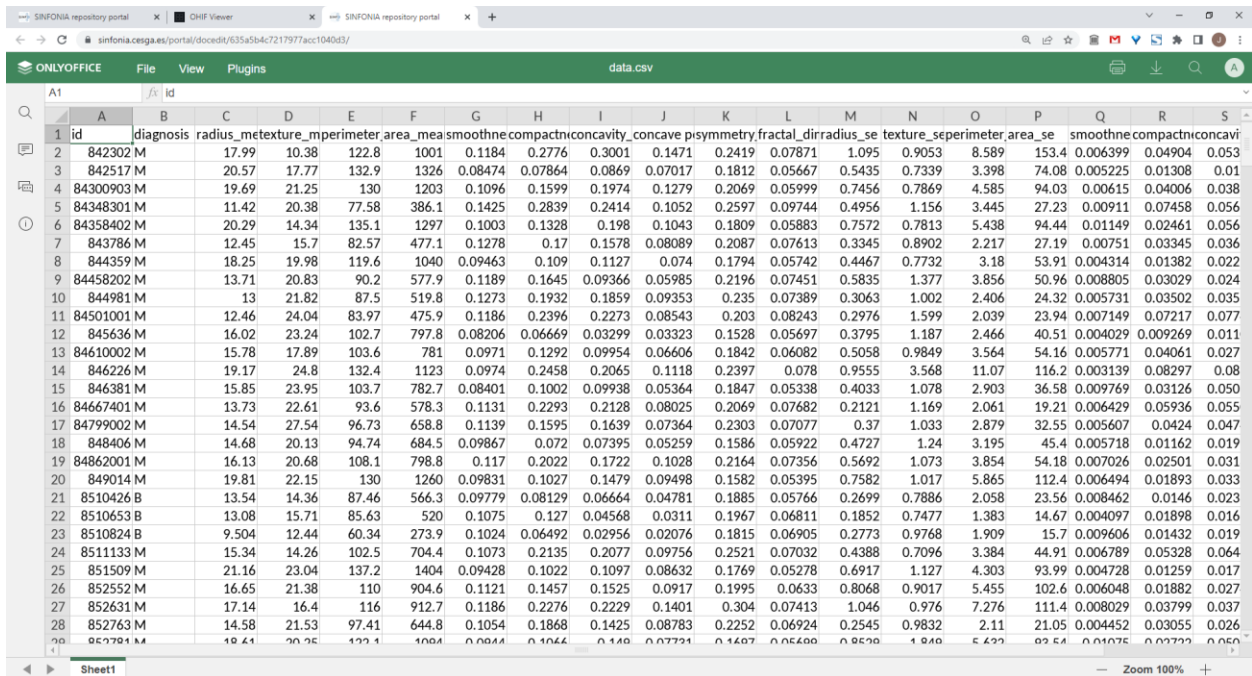


Figure 25: Screenshot of an OnlyOffice viewer tab displaying a CSV file

Selection, copying and moving of resources within fileset pages have received the same updates as described above for folders pages. Such improvements enable operations that the original Girder interface does not support, such as copy/move of files between filesets or from a fileset to folder (automatically creating a new item to contain files) or a native way to delete several files in a single operation (in contrast with the tweak done to Girder’s interface of the Y1 prototype to support that). The content of the fileset pages is also dynamically generated and has a similar pagination function.

Regarding the search engine, a new search criterion called “SINFONIA IDs” has been added, as Figure 26 shows. The internal behaviour of “DICOM tags” criterion has been modified to look for the queried string in both keys and values of DICOM files’ metadata, instead of the original method from Girder, that looked only on values.

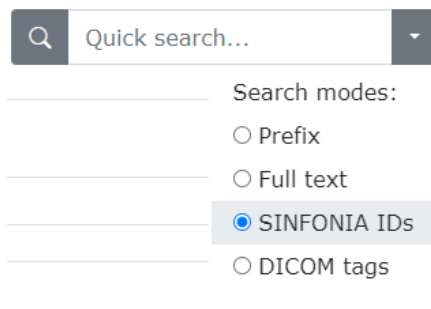


Figure 26: Screenshot detailing the new SINFONIA ID search method

Moreover, results are shown in a dialog (Figure 27) instead of in a separate page, so that users can remain in the page they were and, at the same time, open several search results by asking the browser to open the links in the dialog in new tabs or windows.



Figure 27: Screenshot with a search results dialog

### 5.3.3. Users and groups management

Buttons “Users” and “Groups” from the main menu keep the behaviour they had in the Y1 prototype. By clicking on “Users” a paginated list with all the repository users is shown (Figure 28), with each name being indeed a link to their “personal collection”. Users can also access their collection through the “My Folder” entry of the user actions dropdown in the top right-hand corner.



Figure 28: Screenshot of the “Users” page of the repository

Similarly, the “Groups” button takes the user to a list with the groups defined in the portal, with each entry being a link to a group details page. In this case, the list is split in two, one with the groups for which the user is a member, and another one for the rest of public groups (in contrast to the private ones, that are visible just for their own members). Figure 29 shows an excerpt of this page for a plain user who is a member of a couple of groups and has no administrative privileges in any group or in the repository.

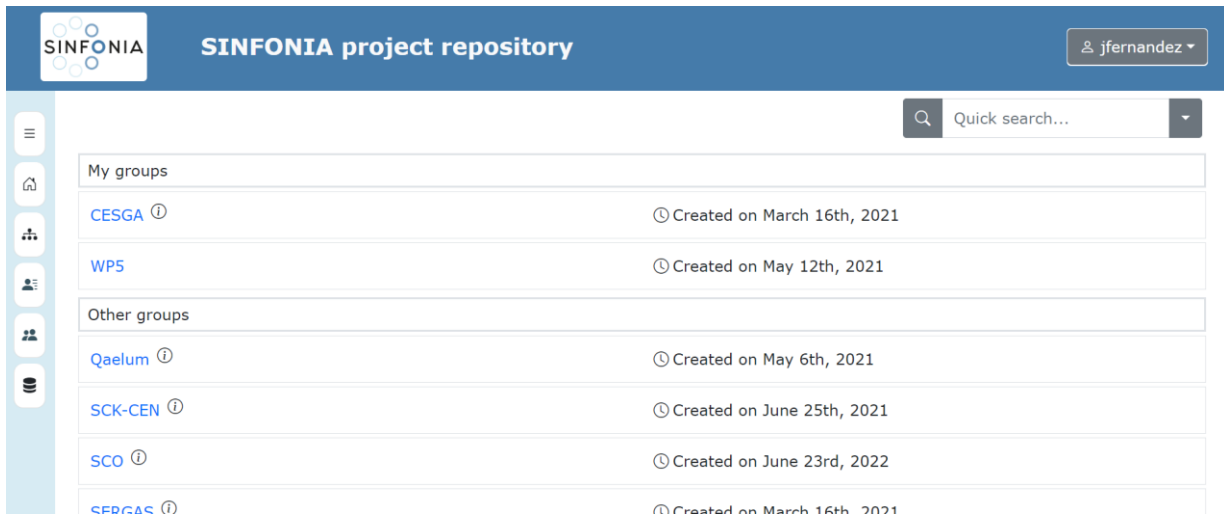


Figure 29: Screenshot partially displaying the Groups list of the repository

As an example, Figure 30 shows the group details page for CESGA group. Besides the changes on the appearance, in this version the category of moderator users has been deactivated, leaving the categories of administrators and ordinary users, or including a search box for adding users to the group, are kept.

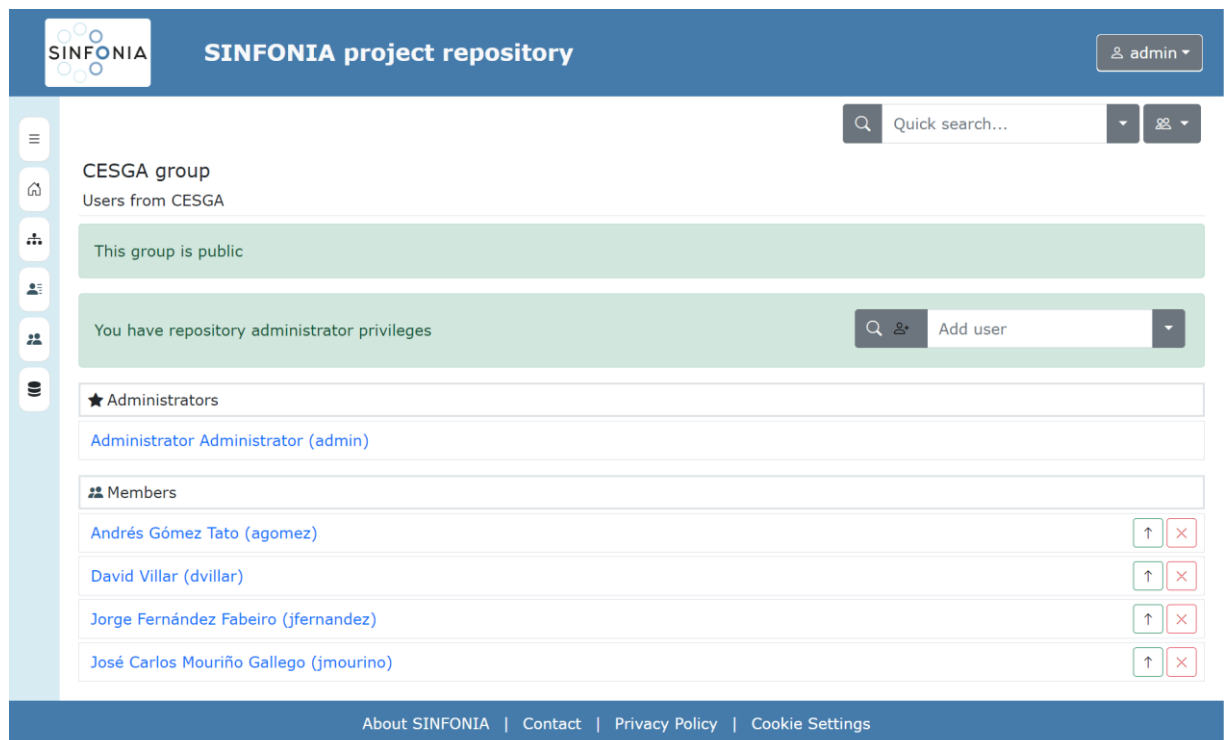


Figure 30: Screenshot with details and users list for CESGA group

### 5.3.4. Interaction with the FTP server

Last but not least, the FTP interface is the other major addition to the web portal of this Y2 increment of the repository. As commented in 5.1.6, user directories in the FTP are not coupled with the Girder-indexed general storage of the web application. Let now us introduce the mechanisms users have to browse the contents of their FTP folders within the portal and to transfer resources among both storage zones.

Despite both zones being uncoupled, users can browse the contents of their accounts in the FTP service folder by clicking either on the entry “FTP” on the left menu or on the “Explore your FTP space” button shown under the breadcrumb in their personal collection page. The appearance and navigation of these pages is very similar to those of the repository's conventional resource pages, but internally the views that handle their requests connect internally to the FTP service with the user's credentials to obtain this information.

Users can pick resources from these FTP pages at their will using the checkboxes on the left and upload them to the repository by means of either the contextual menu or the “Elements currently picked” dialog opened by the selection summary button. Once the operation is completed (Figure 31), user will find in their personal collection a folder with a name of the form “upload\_<username>\_<yyyymmdd>\_<hhmmss±tz>” with a copy of the selected contents inside.

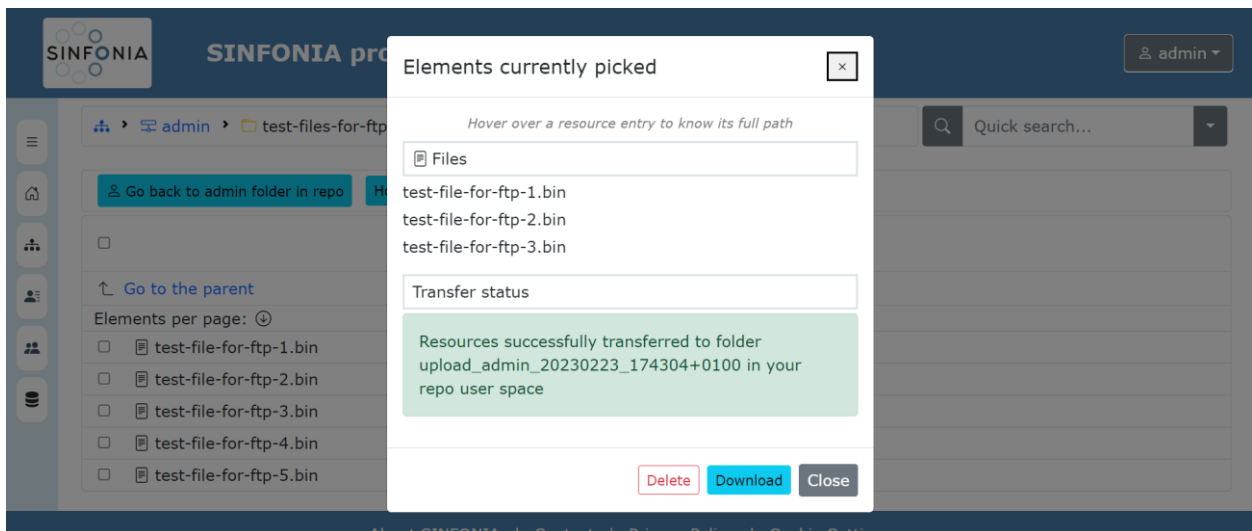


Figure 31: Screenshot of a successful transfer from FTP to user folder in the repository

Likewise, users can pick elements in the conventional resources pages and, instead of moving or copying them to any other conventional location, download them to their FTP service account (Figure 32). In this case, a folder with a name of the form “download\_<username>\_<yyyymmdd>\_<hhmmss±tz>” with a copy of the selected contents inside will appear in their FTP service. Note that in this case the resources are copied directly from their original location on the web portal to the mentioned folder in their FTP directory, without requiring the intermediate step of copying them to the user area of the repository.

Users can go back and forth between the FTP and the “My Folder” pages by means of the aforementioned “Explore your FTP space” and its dual “Go back to <user> folder in repo”. Moreover, a dialog with information on how to use the FTP area is revealed by clicking on the namesake button. Namely, connection details and a link to a guide with instructions to install a GUI-based FTP client and use it to connect to the FTP service are given (Figure 33).



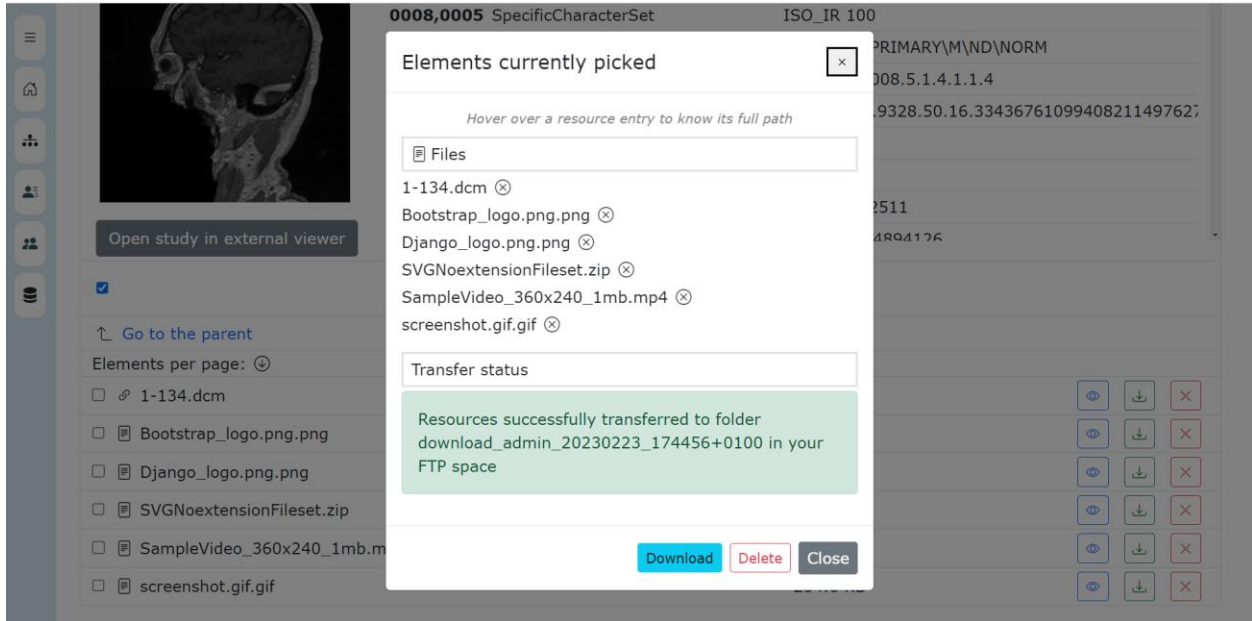


Figure 32: Screenshot of a successful transfer of files from a fileset to the user's FTP space

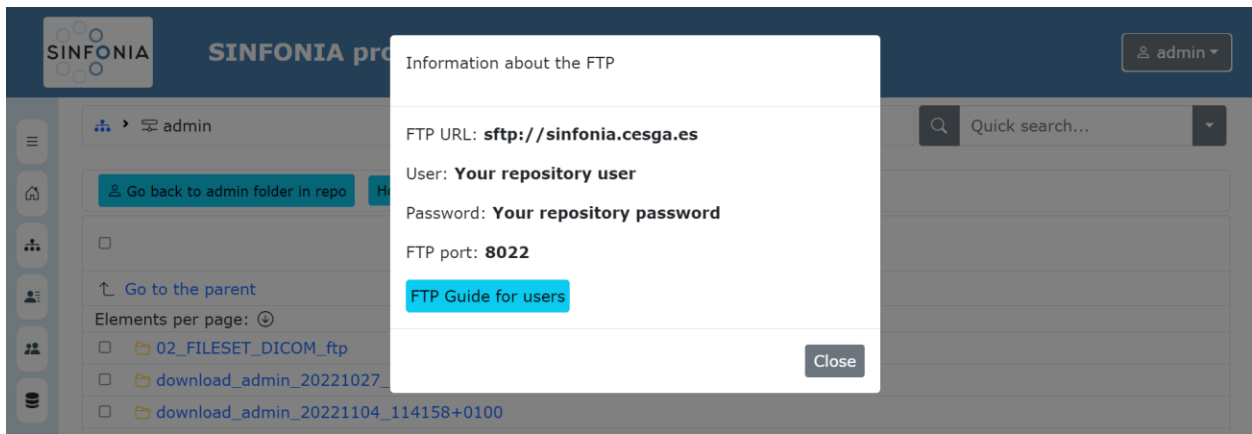


Figure 33: Screenshot of the dialog with information for using the FTP server

## 6. Y2 version validation

Approximately two weeks before the 3<sup>rd</sup> Consortium Meeting held on 27-28 October 2022, a preliminary version of the Y2 iteration of the repository was released for internal tests within WP5. Both in the final part of the development and in this brief period of internal validation, CESGA's development team detected a number of issues to resolve and possible improvements to implement. They span across the various layers of the repository's architecture, from details of the portal's graphical interface (e.g.: the behaviour of the drag-and-drop feature could be misleading for certain users) to major overhauls of parts of the Django web application (e.g.: single-step transfers between FTP server and repository folders, creation of a master account different to plain users with administrative powers and implications of that in ACLs, improvement of the SINFONIA ID search method). So, considering the varied scope of these issues and the possibility that they might clash with eventual proposals from the rest of the partners, we preferred to await their feedback and postpone the decision on implementing such changes and additions.

Moreover, the rest of WP5 partners conducted their own tests, focusing on features they anticipate using most frequently or for which they have proposed fixes and improvements in previous iterations. UoC reviewed the workflow designed for the exchange of files between the conventional repository storage and the FTP service, identifying as fundamental to overcome the limitation that users cannot directly copy hierarchies of resources between the FTP service and the repository without first passing them through the user folder. They also suggested to allow users to tag the resources to be copied from the FTP service with SINFONIA IDs in this step and raised the need to provide users with essential information on how to connect to the service. In the context of these usability tests of the FTP service, they also found a bug that were making the resource list pages of the portal to crash when dealing with files with non-Latin characters in their names. UoC also proposed a less restrictive alternative to their initial proposal of limiting the access to the repository to a safelist of IP addresses (see section 3.2). Instead, they devised a mechanism to kill the users session after they exit the web browser.

SERGAS carried out an exhaustive validation of file uploading and downloading operations, finding several bugs, and identifying interesting usability improvements. Namely, they detected that the web application was experiencing problems in completing both uploads and downloads of large files (with sizes in the range of GB). After several tests, it was found that these problems were caused by several factors: limitation in the NGINX proxy of the size of files accepted, duration of the user session shorter than the usual time for uploading files of such size range over HTTP, and exhaustion of the internal memory of the server when forwarding files to or getting them from Girder/Orthanc. Connected to this, SERGAS also missed the provision of information on the interface about the progress of these operations and the possibility to leave them running on background allowing users to browse the web portal in the meantime. Other interesting proposals were to ask users for organising DICOM files automatically just after upload in hierarchies of folders and filesets according to the "DICOM Model of the Real World" and implement a kind of "recycle bin" from which removed resources can be recovered or permanently wiped.

QAEUM provided detailed feedback on how to improve the user experience in the repository. They enriched their initial proposal of including a contact form instead of a link to the project support email address by suggesting the possibility of allowing users to attach images or other resources to support the queries to be sent through this form. They also detected a loophole in the interface design that users may get confused when trying to upload resources into the root of a collection as no upload icon or menu entry is shown in that point, as well as a bug when trying to preview Microsoft Office documents using the OnlyOffice plugin. They also miss an integrated user guidance throughout the repository in the form of tooltips. Last but not least, QAEUM also covered data anonymization and user privacy in their validation work. Regarding data anonymization, they offered their insights about a further implementation of an effective mechanism to

assess and/or verify if the uploaded data is anonymized (or at least to know to what extent), in the vein of selecting randomly different segments of the uploaded data and look for potential personal information leaks. This task would be performed regularly, and depending on the result of such anonymization assessment, actions like calling the users to remedy the information leak could be taken. With relation to user privacy, they detected a misconfiguration in the JavaScript plugin in charge of enforcing the opting-in policy for non-essential cookies.

A preliminary version was deployed in a timely manner in order to present it to the consortium in a live demonstration performed at a workshop within the 3<sup>rd</sup> SINFONIA yearly meeting in M26 (late October 2022). After the demonstration, the participants were given some free time to get in touch with the new version, give their first impressions and ask any questions they might have. The concerns expressed by WP5 colleagues about the usability of the FTP service were confirmed: the audience agreed on the usefulness of adding the FTP service for mass file transfers but requested additional help on how to connect and use the service on their own. They also found having to pass resources through the user folder to be a bit cumbersome, which reinforces the need identified by the development team on improving the user experience of the FTP service in further increments. Another unanimous suggestion already raised by WP5 partners in the internal validation process was the incorporation of a “recycle bin” to allow the recovery of accidentally deleted resources.

Looking ahead to the next iterations of the repository, several members also remarked on the need of building an AI algorithm execution platform as generic as possible, allowing to execute codes working with different libraries, packages and even languages (the latter is the case of SKANDION and its specific needs with MATLAB). In relation to this requirement of comprehensiveness, the Scientific Coordinator Prof. John Damilakis also expressed the importance of testing to what extent the chosen PACS (Orthanc) is able to process the huge number of DICOM file modalities defined in the standard, envisaging eventual extensions of the SINFONIA project to other types of tumours.

As for the Y1 prototype, partners were encouraged again to continue with their experiments on their own to get familiar with the new look-and-feel of the web portal, to learn more about the new features introduced, and to think about more suggestions or to detect and report eventual bugs. The workflow is the same as the one applied so far. CESGA, as leader of the development of the repository, remains attentive to any communication from the Consortium sent to [sinfonia@cesga.es](mailto:sinfonia@cesga.es), the mailbox dedicated for these issues. Proposals are first evaluated to distinguish whether they are (1) minor patches that could be applied on the fly over the running version, (2) affordable changes that would require some additional studies but could be part of a partial upgrade of the repository, and (3) major changes that would mean revising and extending the architecture, the implementation of its components, or both, all of which are processes that fit better in further increments of the repository.

### 6.1. Review of the delivered version

As we did for the Y1 prototype in D5.1, let us finish the validation section with a self-assessment about how, and up to which grade of fulfilment, the requirements and improvement proposals collected from the users are being addressed after the updates of M29 (January 2023) and M31 (March 2023), applied on top of the version introduced in the 3<sup>rd</sup> CM. A chart summarising this review is shown in Figure 34, with each requirement being marked in black, yellow, or green depending on being already covered in the Y1 prototype, partially fulfilled in the latest version available when preparing this document (M31, March 2023), or totally fulfilled, respectively.

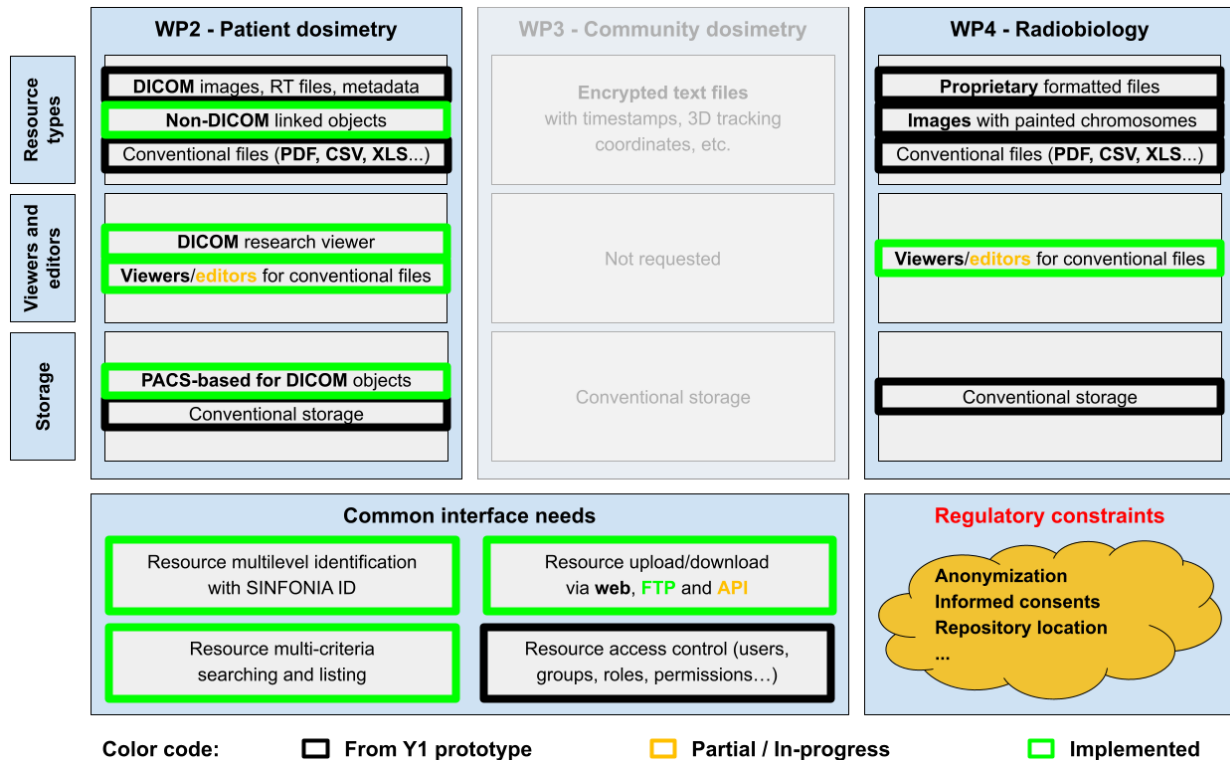


Figure 34: Summary chart of needs and constraints fulfilled by the latest Y2 increment

With respect to WP3 partners, let us remember that, as declared in D5.1, they declined to use the repository for their daily research work due to their very specific requirements, including real-time upload of massive amounts of tracking data. Nevertheless, they occasionally make use of their separate collection to share data with partners from other WPs. It was indeed this sporadic use to access data shared by WP2 which revealed some of the limitations of the repository regarding the upload and download of large files (in a GB range) both in the user experience and in the internal implementation of these operations.

In relation to the specific needs of WP4, this increment of the repository stills accept the upload of proprietary-formatted files or images with painted chromosomes. No special requests have been received for embedding viewers for proprietary-formatted files, but images with painted chromosomes should be able to be previewed while stored in conventional image formats, thanks to the general previewer embedded in the fileset pages. The general previewer also allows reviewing the directory hierarchy contained in a ZIP file, in response to SCO's request, as a first step towards integrating more complex remote operations with compressed datasets. Regarding other conventional files like text or PDFs reports or data spreadsheets, they can be previewed by means of the OnlyOffice viewer. The possibility of editing these file types, which was a demand expressed by some SU researchers, is still under study due to difficulties in connecting Girder and the editing part of OnlyOffice.

Most of the common interface features that were provided through the Girder web application have been replicated in this new version's interface, with the exception of uploading and downloading via API, which are no longer available through Girder's one until the repository's own API tailored to the needs of the AI algorithm execution platform is defined. Bugs affecting these features are mostly solved, yet finer solutions could be implemented in the Y3 version along with the other proposed improvements for them. Moreover, the FTP service for massive transfer of files has been implemented and is operative, and users are provided with connection details and basic instructions to install locally an FTP client tool to access this service. Resource identification using the SINFONIA scheme is now available for both new and existing resources, and

consequently the resource multi-criteria searching and listing is extended by including the SINFONIA ID criterion. Both the improvement of allowing free movement of resources between repository and FTP folders and the implementation of a “recycle bin” for deleted resources are already being considered for rapid incorporation into a forthcoming increase in the repository.

With relation to the regulatory constraints, we keep them in an in-progress status as we are still thinking out up to what extent (just assessment, actual modification of files...) and how (run locally before upload, use functions on PACS, both) to provide an anonymisation service for DICOM files. Meanwhile, the anonymization guidelines remain available for users. Privacy and security of research data is enforced by the same methods. Both the registration to SINFONIA members and the navigation through the repository is still restricted to logged-in users, and for having their accounts created, project members must explicitly accept first the SINFONIA Data Repository Users Privacy Policy. Moreover, users are explicitly informed of the repository's use of technical cookies. For the time being, performance and analytical cookies are not being used, but the cookie consent plugin is prepared for enforcing an opting-in policy for them. The alternative proposed for closing user sessions automatically instead of enforcing a safelist of IP addresses is still under consideration. Regarding the physical infrastructure of both repository servers and data storage, it is still hosted on CESGA's premises in the EU (Santiago de Compostela, Spain).

## 7. Conclusion

In this deliverable we have presented the SINFONIA repository platform in its Y2 version, initially presented at the 3<sup>rd</sup> CM meeting in M26 (October 2022) and enhanced with a number of improvements and bugfixes included after incremental updates in M29 (January 2023) and M31 (March 2023).

A brief reminder on the purpose of the SINFONIA project and the key objectives that the implementation of a data repository intends to fulfil have been provided as an introduction for the reader. Work packages relevant for those objectives and the tasks within these WPs have been recalled also, explaining which of them and to what extent they were covered by the previous deliverable D5.1 about the Y1 prototype, and which are addressed in this document.

An updated description of the software engineering process of the platform was presented, with the aim of highlighting the differences between the process carried out during Y1, dedicated to producing a working prototype in a reasonable time, and the one to be applied from now on, which should guide the annual production of platform versions and at the same time allow for a continuous improvement of its features.

The analysis stage of the Y2 version was based on two fundamental sources of information: one the one hand, the review of the users' requirements fulfilment reached by the Y1 prototype; on the other hand, the most relevant feedback received regarding Y1 prototype features. From this information we could identify a number of shortcomings, weak points and suggestions for improvements that were worth considering for incorporation into subsequent versions, and extend the platform architecture accordingly.

Before presenting the new extended architecture, the software engineering background that supported the initial architecture of the Y1 prototype was explored in depth. Once these concepts had been introduced, the new components to be incorporated into the architecture were described in order to extend it according to the needs updated in the previous stage.

While the innermost core of the Year 1 prototype was maintained, all components of the extended architecture, both new and evolved from the Y1 prototype, had to be implemented, deployed and properly connected. After describing this entire process, the main functionalities of the Y2 version were reviewed, covering both the changes with respect to the Y1 prototype and the new features implemented.

Finally, the validation process has been narrated. First both the CESGA development team and the rest of its partners within WP5 made a preliminary research work looking for undetected issues to solve and for improvements and additions in line with what the rest of the partners may require later on. The on-site demonstration at the 3<sup>rd</sup> CM allowed the partners to collectively assess the new version of the platform and to come up with new ideas for further iterations. As for the Y1 prototype, the validation process was concluded with a self-assessment about up to which extent the latest Y2 increment extended the coverage of both initial and newly identified needs and requirements.

## References

- [1] Wikipedia contributors, "Multitier architecture," Wikipedia, The Free Encyclopedia, [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Multitier\\_architecture&oldid=1131409326](https://en.wikipedia.org/w/index.php?title=Multitier_architecture&oldid=1131409326). [Accessed 02 03 2023].
- [2] V. Deiwakh, "Vahid Deiwakh's website," 16 04 2021. [Online]. Available: <https://vahid.blog/post/2021-04-16-understanding-the-model-view-controller-mvc-pattern/>. [Accessed 02 03 2023].
- [3] DICOM Standards Committee, "DICOM PS3.4 2021d - Service Class Specifications: C6.1.1 Patient Root Query/Retrieve Information Model," NEMA, 2021. [Online]. Available: [http://dicom.nema.org/medical/dicom/current/output/html/part04.html#figure\\_C.6-1](http://dicom.nema.org/medical/dicom/current/output/html/part04.html#figure_C.6-1). [Accessed 28 10 2021].
- [4] M. O'Gara, "Ben Golub, Who Sold Gluster to Red Hat, Now Running dotCloud," SYS-CON Media, 23 07 2013. [Online]. Available: <https://web.archive.org/web/20190913100835/http://maureenogara.sys-con.com/node/2747331>. [Accessed 02 03 2023].
- [5] Docker, Inc., "Docker frequently asked questions (FAQ)," [Online]. Available: <https://docs.docker.com/engine/faq/#what-does-docker-technology-add-to-just-plain-lxc>. [Accessed 02 03 2023].
- [6] C. K, "poweruser.blog," 01 07 2019. [Online]. Available: <https://poweruser.blog/lightweight-windows-containers-using-docker-process-isolation-in-windows-10-62519be76c8c>. [Accessed 02 03 2023].
- [7] Docker, Inc., "What is a Container?," [Online]. Available: <https://www.docker.com/resources/what-container/>. [Accessed 02 03 2023].
- [8] Docker, Inc., "Dockerfile Reference. Docker documentation.," [Online]. Available: <https://docs.docker.com/engine/reference/builder/>. [Accessed 02 03 2023].
- [9] Docker, Inc., "Docker Registry. Docker documentation.," [Online]. Available: <https://docs.docker.com/registry/#what-it-is>. [Accessed 02 03 2023].
- [10] A. Mazy, "Orthanc Server 1.11.3 DICOM Conformance of Statement," 03 02 2023. [Online]. Available: <https://hg.orthanc-server.com/orthanc/file/Orthanc-1.11.3/OrthancServer/Resources/DicomConformanceStatement.txt>. [Accessed 02 03 2023].
- [11] Orthanc community, "Model of the real world. Understanding DICOM with Orthanc. Orthanc Book.," [Online]. Available: <https://book.orthanc-server.com/dicom-guide.html?highlight=real%20world#model-of-the-real-world>. [Accessed 02 03 2023].
- [12] Orthanc community, "How does Orthanc store its database? Orthanc Book.," [Online]. Available: <https://book.orthanc-server.com/faq/orthanc-storage.html>. [Accessed 03 02 2023].

- [13] Orthanc community, "PostgreSQL plugins. Orthanc Book.," [Online]. Available: <https://book.orthanc-server.com/plugins/postgresql.html>. [Accessed 03 02 2023].
- [14] Orthanc community, "MySQL/MariaDB plugins. Orthanc Book.," [Online]. Available: <https://book.orthanc-server.com/plugins/mysql.html>. [Accessed 02 03 2023].
- [15] Django Software Foundation, "Overview. Django Project.," [Online]. Available: <https://www.djangoproject.com/start/overview/>. [Accessed 02 03 2023].
- [16] A. Holovaty and J. Kaplan-Moss, "Chapter 5: Models. The Django Book.," [Online]. Available: <https://web.archive.org/web/20160902130823/http://www.djangobook.com/en/2.0/chapter05.html#the-mtv-or-mvc-development-pattern>. [Accessed 02 03 2023].
- [17] MongoDB, "BSON (Binary JSON) Serialization," 2009. [Online]. Available: <https://bsonspec.org/>. [Accessed 02 03 2023].
- [18] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," *Computer*, vol. 43, no. 2, pp. 12-14, 2010.
- [19] A. Yigal, "Sqlite vs. MySQL vs. PostgreSQL: A Comparison of Relational Databases," logz.io, [Online]. Available: <https://logz.io/blog/relational-database-comparison/>. [Accessed 03 02 2023].
- [20] A. S. Gillis, "Secure File Transfer Protocol (SSH File Transfer Protocol)," TechTarget, [Online]. Available: <https://www.techtarget.com/searchcontentmanagement/definition/Secure-File-Transfer-Protocol-SSH-File-Transfer-Protocol>. [Accessed 03 02 2023].
- [21] Rebex.net, "SFTP Standards. SFTP Protocol Info Site for Developers.," [Online]. Available: <https://www.sftp.net/specification>. [Accessed 02 03 2023].
- [22] Canonical Ltd., "Network User Authentication with SSSD. Ubuntu Server Docs.," [Online]. Available: <https://ubuntu.com/server/docs/service-sssd>. [Accessed 02 03 2023].
- [23] Network Working Group, "RFC 4511," IETF, 06 2006. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc4511.txt>. [Accessed 02 03 2023].
- [24] OpenLDAP, "Introduction to OpenLDAP Directory Services," [Online]. Available: <https://www.openldap.org/doc/admin24/intro.html>. [Accessed 02 03 2023].