



Deliverable No. 8.2

Prototype implementation of the CHIC repositories

Grant Agreement No.: 600841
Deliverable No.: D8.2
Deliverable Name: Prototype implementation of the CHIC repositories
Contractual Submission Date: 31/03/2015
Actual Submission Date: 30/04/2015

Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	X



COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	CHIC
Project Full Name:	Computational Horizons In Cancer (CHIC): Developing Meta- and Hyper-Multiscale Models and Repositories for <i>In Silico</i> Oncology
Deliverable No.:	D8.2
Document name:	Prototype implementation of the CHIC repositories
Nature (R, P, D, O) ¹	O
Dissemination Level (PU, PP, RE, CO) ²	CO
Version:	2
Actual Submission Date:	30/04/2015
Editor: Institution: E-Mail:	Philippe Büchler UBERN philippe.buechler@istb.unibe.ch

ABSTRACT:

The present document describes the prototype implementation of the CHIC repositories. More specifically, this report presents the prototype implementation of the following repositories:

- The clinical data repository, which will permanently host all the related medical data produced or collected by the CHIC project.
- The model/tool repository, which will host the hypomodels, the hypermodels, the data transformation tools, the linkers and other tools used in the construction and execution of hypermodels.
- The *in silico* trial repository, where the data related to *in silico* trials and *in silico* experiments will be stored.
- The RDF storage solution for semantic metadata.

KEYWORD LIST:

repositories, clinical data, models, *in silico* trial, semantics, metadata

¹ R=Report, P=Prototype, D=Demonstrator, O=Other

² PU=Public, PP=Restricted to other programme participants (including the Commission Services), RE=Restricted to a group specified by the consortium (including the Commission Services), CO=Confidential, only for members of the consortium (including the Commission Services)

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600841.

The author is solely responsible for its content, it does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of data appearing therein.

MODIFICATION CONTROL			
Version	Date	Status	Author
1.0	17/12/2014	Index Draft	Roman Niklaus, UBERN
1.1	26/02/2015	Draft	Roman Niklaus, UBERN
1.2	26/02/2015	Draft	Nikolaos Tousert, ICCS-NTUA
1.3	26/02/2015	Draft	Alexander Samuel, UCL
1.4	05/03/2015	Draft	Roman Niklaus, UBERN
1.5	13/03/2015	Draft	Roman Niklaus, UBERN
1.6	19/04/2015	Draft	Nikolaos Tousert, ICCS-NTUA
1.7	24/04/2015	Draft	Nikolaos Tousert, ICCS-NTUA
1.8	27/04/2015	Draft	Nikolaos Tousert, ICCS-NTUA
1.9	29/04/2015	Revision	Dimitra Dionysiou ICCS-NTUA, Georgios Stamatakis ICCS-NTUA
2.0	30/04/2015	Final	Philippe Büchler, UBERN

List of contributors

- Philippe Büchler, UBERN
- Michael Kistler, UBERN
- Roman Niklaus, UBERN
- Alexander Samuel, UCL
- Nikolaos Tousert, ICCS-NTUA
- Nikolaos Christodoulou, ICCS-NTUA

Contents

CONTENTS.....	5
FIGURES	6
TABLES	8
1 EXECUTIVE SUMMARY	9
2 INTRODUCTION.....	11
2.1 PURPOSE OF THIS DOCUMENT	11
2.2 STRUCTURE OF THE DELIVERABLE.....	11
3 GENERAL WORKFLOW FOR DATA UPLOAD	13
3.1 DATA TYPES AND STANDARDS.....	13
3.2 WORKFLOW	14
4 RDF STORAGE SOLUTION FOR SEMANTIC METADATA	17
4.1 INTRODUCTION	17
4.2 TOOLS	17
4.2.1 <i>Input and output of RICORDO tools</i>	18
4.3 THE CHIC SEMANTIC METADATA LIFECYCLE	18
4.4 WHAT CAN BE ANNOTATED?	19
5 CLINICAL DATA REPOSITORY	20
5.1 INTRODUCTION	20
5.2 AUTHENTICATION	20
5.3 USE CASES	21
5.3.1 <i>Actors</i>	21
5.3.2 <i>Overview</i>	22
5.3.3 <i>Upload & Versioning</i>	23
5.3.4 <i>Linking</i>	25
5.3.5 <i>Annotation & Search</i>	26
5.3.6 <i>Validation</i>	29
5.3.7 <i>Data organization</i>	30
5.3.8 <i>Download</i>	33
5.4 USER GUIDE	35
5.4.1 <i>CDR-UC 1: Upload a file</i>	35
5.4.2 <i>CDR-UC 2: Upload a new version of a file</i>	36
5.4.3 <i>CDR-UC 3: Link one object to another</i>	37
5.4.4 <i>CDR-UC 4: Manually annotate an object with an ontology term</i>	39
5.4.5 <i>CDR-UC 5: Automatically annotate objects with ontology terms</i>	40
5.4.6 <i>CDR-UC 6: Search objects based on ontology terms</i>	40
5.4.7 <i>CDR-UC 7: Review/validate new objects</i>	41
5.4.8 <i>CDR-UC 8: Create a folder</i>	46
5.4.9 <i>CDR-UC 9: Copy an object to a folder</i>	47
5.4.10 <i>CDR-UC 10: Modify a folder</i>	49
5.4.11 <i>CDR-UC 11: Share a folder with others</i>	52
5.4.12 <i>CDR-UC 12: Download objects or folders</i>	53
5.5 DEPLOYMENT	55
5.5.1 <i>Infrastructure</i>	55
5.5.2 <i>Installation</i>	56
5.6 CONCLUSION	67
6 MODEL/TOOL AND <i>IN SILICO</i> TRIAL REPOSITORIES	68
6.1 INTRODUCTION	68
6.2 MODEL/TOOL AND <i>IN SILICO</i> TRIAL REPOSITORIES ON THE CLOUD	68
6.3 SERVER CONFIGURATION SETTINGS AND DEPLOYMENT FOR THE MODEL/TOOL AND <i>IN SILICO</i> TRIAL REPOSITORIES.....	71

6.4	MODEL/TOOL AND IN SILICO TRIAL REPOSITORIES IN COMPLIANCE WITH DJANGO MVT PATTERN	73
6.5	AUTHENTICATION IN MODEL/TOOL AND IN SILICO TRIAL REPOSITORIES	85
6.6	MODEL/TOOL AND IN SILICO TRIAL REPOSITORIES ADMINISTRATOR SITE	89
6.7	GRAPHICAL USER INTERFACE OF MODEL/TOOL AND IN SILICO TRIAL REPOSITORIES	96
6.7.1	Graphical user interface of model/tool repository	97
6.7.2	Graphical user interface of the in silico trial repository	107
7	REFERENCES	115
	Appendix 1 – Abbreviations and acronyms	116

Figures

Figure 1: The general workflow for data upload	15
Figure 2: Brokered Authentication Flow with the clinical data repository	20
Figure 3: The use case diagram of the clinical data repository	22
Figure 4: The web-based user interface main view of the Clinical Data Repository	35
Figure 5: The clinical data repository in the CHIC cloud infrastructure	55
Figure 6: Adding the Web Server (IIS) role in the Server Manager on the Clinical Data Repository	56
Figure 7: Adding the .NET Framework feature in the Server Manager on the Clinical Data Repository	57
Figure 8: Microsoft SQL Server feature selection on the Clinical Data Repository	58
Figure 9: Creating a local user to be used by the Internet Information Services (IIS) in the Computer Management of the Clinical Data Repository	59
Figure 10: Creating an application pool in the Internet Information Services (IIS) Manager on the Clinical Data Repository	59
Figure 11: Creating the website and web service in the Internet Information Services (IIS) on the Clinical Data Repository	60
Figure 12: Shibboleth InProcess configuration on the Clinical Data Repository	60
Figure 13: Shibboleth RequestMapper configuration on the Clinical Data Repository	61
Figure 14: Shibboleth Identity Provider (IdP) configuration on the Clinical Data Repository	61
Figure 15: Shibboleth Attribute Map configuration on the Clinical Data Repository	62
Figure 16: Creating the database in the Microsoft SQL Server Management Studio on the Clinical Data Repository	63
Figure 17: Starting the SQL Server Agent automatically after booting the Clinical Data Repository	63
Figure 18: Creating a Maintenance Plan in the Microsoft SQL Server Management Studio for backup purposes on the Clinical Data Repository	64
Figure 19: Creating a publishing profile in Visual Studio 2013 for the Clinical Data Repository	65
Figure 20: Customizing the Web.config on the Clinical Data Repository	66
Figure 21: Setting the Local Security Policy to allow the creation of symbolic links on the Clinical Data Repository	66
Figure 22: Model/tool and <i>in silico</i> trial repositories deployed to the CHIC private cloud. For a brief description of the components see TABLE... ..	69
Figure 23: NginX is used to divide up requests between static media files and dynamic Django driven pages	71
Figure 24: NginX's IP-based virtual host configuration file	72
Figure 25: Apache's IP-based virtual host configuration file	73
Figure 26: Components of the MVT pattern	74

Figure 27: MTV pattern used in model/tool and <i>in silico</i> trial repositories	75
Figure 28: Entity-Relationship (ER) diagram of model/tool repository	77
Figure 29: Entity Relationship (ER) diagram of <i>in silico</i> trial repository.....	78
Figure 30: Entity Relationship (ER) diagram of tables associated with basic authentication and authorization	79
Figure 31: Model/tool and <i>in silico</i> trial applications inside the same Django project.....	80
Figure 32: Page for storing basic information of a model/tool.....	81
Figure 33: Django Template language for storing basic information of a model	82
Figure 34: The urls.py file, which belongs to model_app application (the application in charge of model/tool repository) is responsible for mapping the requested url to a specific view.....	83
Figure 35: Data model class and corresponding database table in model/tool repository.....	83
Figure 36: Django view function for storing basic information of a model	84
Figure 37: Brokered authentication for the model/tool repository web site.....	86
Figure 38: Brokered authentication for the <i>in silico</i> trial repository web site	87
Figure 39: Brokered authentication for the model/tool repository web services	88
Figure 40: Brokered authentication for the <i>in silico</i> trial repository web services.....	89
Figure 41: Use case diagram which overviews the usage requirements for model/tool and <i>in silico</i> trial repositories administrator site	90
Figure 42: Login page of the administrator site for model/tool and <i>in silico</i> trial repositories	91
Figure 43: Home page of the administrator site for model/tool and <i>in silico</i> trial repositories	92
Figure 44: Groups of users page of the administrator site for model/tool and <i>in silico</i> trial repositories	93
Figure 45: Administrator adds a user (modeller) to a group (modellers group).....	94
Figure 46: Administrator changes the information of a model's parameter.....	94
Figure 47: Administrator adds permissions to clinicians group.....	95
Figure 48: Administrator changes the information related to the users	95
Figure 49: Administrator stores an experiment through the administrator site	96
Figure 50: The researcher/modeller has read, create, change and delete permissions on model/tool repository	97
Figure 51: The clinician has read permission on model/tool repository	98
Figure 52: Page for storing the basic information of a new model	99
Figure 53: Page for storing a file associated with a specific model or tool.....	100
Figure 54: Page for storing information about a parameter associated with a specific model.....	101
Figure 55: Page for storing information about a reference (resource) associated with a specific model/tool.....	102
Figure 56: Page for storing a property and its description	103
Figure 57: Page for viewing, changing and deleting the basic information of a selected model	104
Figure 58: Page for viewing, changing and deleting the information of a selected parameter.....	105
Figure 59: Page for viewing, changing and deleting the information of a selected property.....	106
Figure 60: The researcher the modeller and the clinician have read, create, change and delete permissions on <i>in silico</i> trial repository	107
Figure 61: Page for storing information about a trial	108
Figure 62: Page for storing a new experiment.....	109
Figure 63: Page for storing a new subject.....	110
Figure 64: Page for storing a new file which is associated with a subject	111
Figure 65: Page for storing information about a reference (resource) associated with a specific trial	112
Figure 66: Page for viewing, changing and deleting the information of a subject	113
Figure 67: Page for viewing, changing and deleting the information of a reference stored in the <i>in silico</i> trial repository	114

Tables

Table 1: Data types and standards	13
Table 2: The different sources of the data types	14
Table 3: Parties involved in the general workflow for data upload	16
Table 3: Parties involved in the general workflow for data upload	16
Table 4: The general categories of users called actors for the Clinical Data Repository	21
Table 5: Template for all use cases	23
Table 6: The CHIC security framework vocabulary used in the Shibboleth Attribute Map	62
Table 7: Brief description of the components of Openstack Architecture	70

1 Executive Summary

The CHIC project aims at developing reproducible, interoperable and collaborative hypermodels in the cancer domain. This task requires the integration of models developed by several partners with clinical data, which will enable to evaluate the effect of the treatment during *in-silico* trials. The CHIC repositories constitute an important part of the computational infrastructure required to perform these analysis. The CHIC repositories are used to store the clinical data, the cancer models developed by the partners as well as to host the results of the *in-silico* trials. In addition, the semantic information used to annotate the datasets and objects will be store as RDF triples to enable advanced search mechanism.

This deliverable presents the prototype implementation of the repositories developed by WP8:

- A repository of multiscale data exploitable by the models, either by physically storing the data in the project's data repository, or by providing links to other, already existing, data repositories or warehouses
- A repository of cancer models, spanning from models of generic fundamental biomechanisms involved in cancer progression and treatment response, such as cell cycle and cell metabolism, to complex multiscale models of various types of cancer
- A repository to host the results of the *in silico* trials for various types of cancer
- A distributed RDF repository to store metadata from each partner, including the corresponding interfaces for annotating and querying.

In its prototype implementation, the clinical data repository is able to store medical images and clinical study data. However, due to this flexible design, the adaptation of the system for new data types such as histological data and genetic data will be easily performed. On the other hand, a specific import system is implemented for each type of data, in order to extract the relevant information from the uploaded file and to allow its annotation with the appropriate metadata. The prototype system has already been used to exchange data between clinicians and researchers thanks to the adoption of the EU-CHIC security framework at the very beginning of the prototype implementation. At this point the recommended brokered authentication mechanism introduced in deliverable “D5.2 - Security guidelines and initial version of security tools” has been fully integrated into the clinical data repository to support Single Sign-On.

In this deliverable we have presented possible use cases of the clinical data repository based on the general concepts introduced in the previous deliverable. Furthermore, the prototype implementation has been documented with a user guide as descriptive and simple as possible, so that it could also be provided to all end-users for training/instruction purposes. Last but not least, the clinical data repository has been successfully deployed to the private cloud infrastructure provided by FORTH allowing great flexibility in terms of computational power, storage space and networking resources. The deployed prototype implementation can be accessed by the following URL <https://cdr-chic.ics.forth.gr>. In addition, the semantic information used to annotate the datasets and objects will be stored as RDF triples to enable advanced search mechanisms. Special APIs, known collectively as RICORDO, have been developed to facilitate querying and creating such triples, including sophisticated ontology searches in order to make the triples as semantically meaningful as possible.

In its prototype implementation, the model/tool repository is able to store the models that will be developed (or are already developed) in the context of the CHIC project. It is also able to host tools such as linkers and data transformation tools necessary for the construction of hyper-models. Apart from models and tools, the prototype implementation of the model/tool repository is able to store descriptive information of models and tools, needed parameters (input and output) for the proper

linking between (hypo)models, and different versions of binaries and source code. In its prototype implementation, the *in silico* trial repository may be used for the persistent storage of simulation scenarios and *in silico* predictions. The *in silico* trial repository is already able to store the input data (the original state of the patient), the simulation scenario (the *in silico* treatment), and the output data (the state of the patient after *in silico* treatment).

2 Introduction

2.1 Purpose of this document

The present document, corresponding to the deliverable “D8.2: Prototype implementation of the CHIC repositories”, describes the design of:

- The data workflow to upload clinical data to the CHIC Clinical Data Repository.
- The RDF storage solution for semantic metadata and its integration with the RICCORDO framework used to query the RDF triples
- The implementation of the clinical data repository, which will permanently host all the related medical data produced or collected during the project.
- The model/tool repository, which will host hypomodels, hypermodels, data transformation tools, linkers and other tools used in the construction and execution of hypermodels.
- The *in silico* trial repository, where data related to *in silico* trials and *in silico* experiments will be stored.

2.2 Structure of the Deliverable

Before starting with the description of the data repositories themselves, this deliverable starts with the explanation of the workflow of clinical data from the hospital to the CHIC infrastructure (Chapter 2). The presented workflow results from discussions between partners that took place in December 2014 during a 2-days meeting in Homburg. It is important to describe the data upload process, since it conditions several technical implementations described later in the deliverable.

The storage of the semantic metadata generated for CHIC will be presented in Chapter 3. The description includes the RDF triplestore as well as the RICCORDO framework, which is a collection of APIs that facilitate the creation, storage, and reasoning over of metadata. These tools enable the semantic annotation of the CHIC objects as well as advanced search mechanisms.

The implementation of the Clinical Data Repository is detailed in Chapter 5. The description details the integration of the system with the CHIC authentication/authorization framework, definition of the use cases as well as the deployment on the CHIC infrastructure. In addition, a user guide for the system has been written to document the web-based user interface and all of its functionalities. Furthermore, the current state of the prototype is illustratively documented.

Finally, the model/tool and *in silico* trial repositories are presented in this deliverable in the same chapter (chapter 6) as they reside in the same virtual machine, use the same web servers and have been both developed using the same web application framework (Django). The way that the model/tool and *in silico* trial repositories use resources from the CHIC private cloud (chapter 6.2) is also described. Chapter 6.3 presents the essential web server configurations needed for the deployment of the model/tool and *in silico* trial repositories and chapter 6.4 outlines how these repositories make use of the MVT pattern. Moreover, chapter 6.5 presents the way in which the recommended brokered authentication mechanism introduced in deliverable “D5.2 - Security guidelines and initial version of security tools” is going to be integrated into the model/tool and *in silico* trial repositories. In addition, chapter 6.6 presents the administrator site for the aforementioned repositories which has been designed only for the administrators in order for them to control the content of the repositories and enrol system users. Last but not least, chapter 6.7 presents some screenshots of the graphical user interface of the model/tool and *in silico* trial repositories.

In this deliverable only the graphical user interfaces of the model/tool and *in silico* trial repositories are presented. The programmatic interfaces as well as the integration of the model/tool and *in silico* trial repositories with the CHIC security framework are going to be reported in deliverable “D8.3: Implementation of the interfaces of the CHIC repositories”.

3 General workflow for data upload

The general workflow for data upload is key in all subtasks of this work package. Therefore, a technical meeting has been organized in Homburg (11-12 Dec. 2014) where all involved parties have contributed to propose a solution that meets all the requirements listed below. This chapter describes the technical approach that has been defined during this meeting. This aspect is critical since it impacts the overall integration of all the components related to clinical data, but more specifically the clinical data repository and the semantic annotations.

The workflow must fulfil several aims, which required the collaboration of several teams and services; the clinicians providing the data, the semantic interface, pseudonymization service and the clinical data repository. The confidentiality of the patients' data is key, but required strong synergies between these tools to simultaneously ensure security and usability of the uploaded datasets. In summary, the aims and requirements of the workflow are given below:

Aim:

1. Define the data types and standards to be used.
2. Minimize the interaction after upload in the repositories.
3. The data of a single patient must be grouped or linked in the repository, even if the different information is uploaded at different time point.
4. Pseudonymization of the data must be ensured by the trusted third party
5. Metadata can be exported and used by RICORDO.

Requirements:

1. Clinical data must be annotated by the clinical staff before upload to the CHIC infrastructure
2. A unique identifier must be used for each patient to keep track of its related information on the clinical side (i.e. before upload and pseudonymization process).
3. Access rights to the data must be specified

3.1 Data types and standards

In order to provide efficient anonymization/pseudonymization of the data, standard file format should be used. This standardization is also important to enable proper extraction of the metadata information from the files by the clinical data repository and storage of this information in searchable tables. In deliverable "D8.1 – Design of the CHIC repositories" most data types were already briefly described. In Table 1 the various types of clinical data are listed in combination with the standard file format used during the general workflow for data upload. The agreement on the file format used for data exchange clinical information will simplify the development process for all involved parties.

Table 1: Data types and standards

Data type	Standard
Clinical data (pathological and also outcome)	CDISC ODM
Imaging data (post-processed, segmented, etc.)	DICOM, MetaImage, Nifti, Analyze

Genetic / Molecular data	SOFT, MINiML
--------------------------	--------------

For better understanding it is also important to know, where the data comes from. In Table 2 the different sources of the data types are listed. This insight is especially useful for parties not involved in the workflow prior to the data upload.

Table 2: The different sources of the data types

Data type	Source
Clinical data	ObTiMA (ontology-based clinical trial management system)
Imaging data	Local PACS (Picture Archiving and Communication System)
Genetic / Molecular data	Platform specific (e.g. Affymetrix) files as generated by the appropriate equipment.

3.2 Workflow

The intention of the workflow shortly described here is not to limit the additional workload on the clinical side, while providing all the relevant information with the data. The trial data collection on the clinical side is not covered. It is rather about contributing the collected data to the CHIC infrastructure. Once the data is uploaded to the CHIC infrastructure it must be impossible to know the origin of the data came from nor any of the patient information without being authorized to translate the used pseudonyms by the TTP. A good example for this workflow is a cryptographic hash function. A cryptographic hash function is considered practically impossible to invert if only the hash value is known. However, the same input data will result always in the same hash code. This one-way concept can be applied to the general workflow for data upload in the CHIC context.

With this concept in mind, it is straightforward to understand the problems and limitations of the data upload pipeline. A critical requirement to meet is to keep track of the patient throughout the anonymization process. Therefore, it is necessary that the same pseudonym is used for the same patient across different file formats. A unique patient identifier will ensure that the repository receiving the data is able to keep the links between datasets obtained from the same patient, even if the data are uploaded at different time points. The information is critical for the CHIC platform, since the modelling tool and other services rely on all the information collected on each individual patient. Another important aspect is the fact that datasets cannot be properly annotated once uploaded to the CHIC infrastructure, because the data uploader does not know where the dataset is actually stored. Due to the pseudonymization requirement, it is necessary to perform the annotation prior to data upload and to transfer this information together with the data file.

Therefore the system must find a compromise between the requirements linked to the data protection, limited time available by the clinician to process the data and the information necessary to run the *in-silico* trials in the CHIC infrastructure. The analysis of these constraints resulted in the following proposition for the data upload workflow:

- A special trial-patient-identifier will be used across all the datasets collected on each patient.
- The datasets will be annotated before upload in a way that ensures a reliable extraction of the meta-information by the repositories.

- The semantic annotations will be stored in a triplestore (providing generalized search functionalities) only after retrieval of the meta information by the clinical data repository.

The general workflow for data upload involves 6 distinct parties which are briefly described in Table 4 and the general workflow is illustrated in Figure 1.

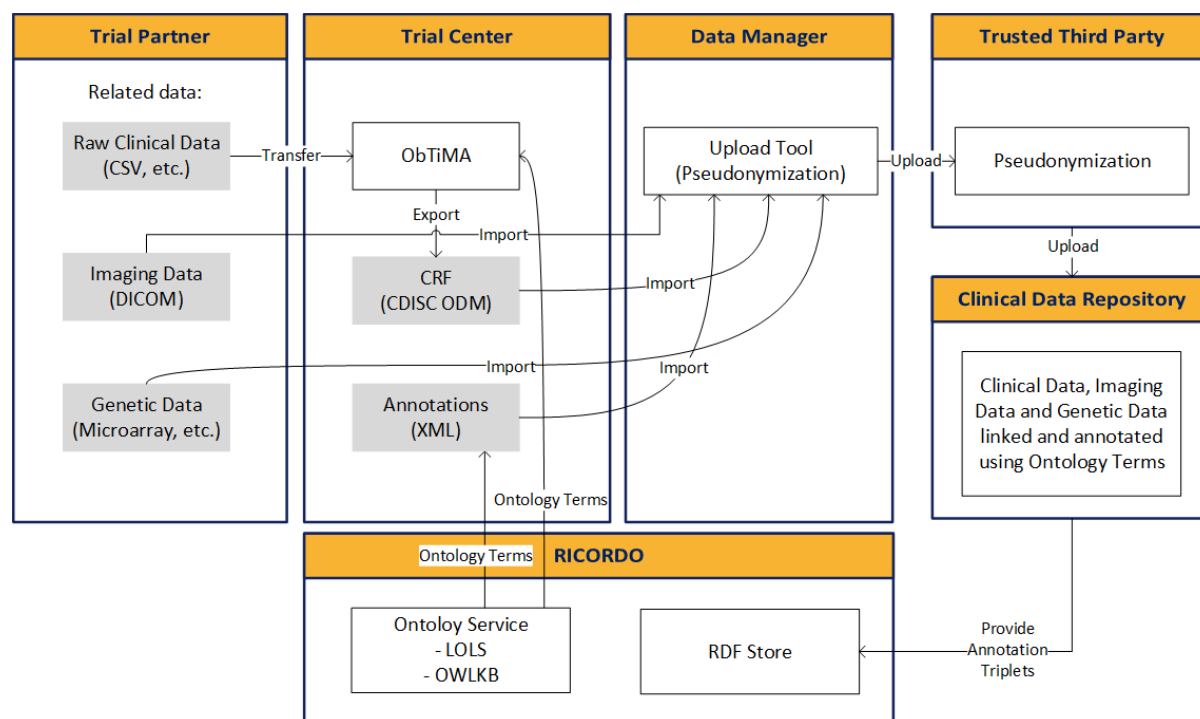


Figure 1: The general workflow for data upload

The steps required to store clinical data in the CHIC environment, including the related semantic annotation is the following:

1. The trial partner enters the patient in the trial center which generates the special trial-patient-identifier.
2. The trial partner provides the data to the trial center.
 - a. The trial partner enters the clinical study data available in raw format into the tool provided by the trial center (e.g. ObTiMA).
 - b. The trial partner provides the imaging data to the trial center.
 - c. The trial partner provides the genetic data to the trial center.
3. The trial center makes sure that the trial-patient-identifier is used accordingly.
 - a. The trial center exports the study data in standardized format (e.g. ObTiMA to CDISC ODM).
 - b. The trial center adds the trial-patient-identifier to the imaging data and creates the annotation file.
 - c. The trial center adds the trial-patient-identifier to the genetic data and creates the annotation file.

4. The trial center creates the special file containing annotations and other metadata.
5. The trial center imports the data in the upload tool.
6. The upload tool applies the first pseudonymization round.
7. The upload tool uploads the data to the trusted third party.
8. The trusted third party applies the second pseudonymization round.
9. The trusted third party uploads the data to the data repository.
10. The data repository extracts the annotations and provides them to the triplestore.

Table 3: Parties involved in the general workflow for data upload

Party	Description
Trial Partner	The trial partner conducts the clinical trial and gathers all data to be stored in the CHIC infrastructure.
Trial Center	The trial center coordinates the clinical trial and ensures that the unique trial-patient-identifier is used across all supported file formats accordingly.
Data Manager	The data manager is responsible to upload compliant data provided by the trial center to the trusted third party after a first pseudonymization round.
Trusted Third Party	The trusted third party accepts data uploaded by the data manager and uploads it to the data repository after a second pseudonymization round.
Data Repository	The data repository stores clinical, imaging and genetic data. Related data is linked and annotated with ontology terms.
RICORDO	RICORDO provides services to search ontology terms, to store annotation triples, to conduct semantically driven search queries and to perform automated semantic reasoning.

Table 4: Parties involved in the general workflow for data upload

4 RDF storage solution for semantic metadata

4.1 Introduction

Semantic metadata refers to data which is specially formatted for greater machine readability, interoperability, and automated semantic reasoning. In the context of CHIC, models and their parts are given various annotations, and these annotations are stored as metadata. For the sake of interoperability and automated reasoning, these annotations refer to standard reference ontologies. For example, if we have an image of a lung, rather than annotate it with a bare string ("lung"), we might annotate it with an ontology term (such as FMA_7195: the "lung" entry in the Foundational Model of Anatomy, a widely-used anatomy ontology). This makes it possible to use the metadata easily, without having to program ad hoc machinery to do so. This also reduces ambiguity and reduces dependence on a particular spoken language (such as English).

4.2 Tools

CHIC uses RICORDO [1], a collection of APIs that facilitate the creation, storage, and reasoning over of metadata.

Individual annotations are stored in the form of RDF triples. RDF is the Resource Description Framework, a W3C recommendation which forms the backbone of the semantic web. An RDF triple consists of a subject, a predicate, and an object. For a concrete example, imagine a triple whose subject is (a formal name of) a particular photograph; whose object is (an ontology term for) "lung"; and whose predicate is (a formal name of) "image-of". This triple means that the image in question is an image of a lung. Crucially, by storing the fact in this form, the fact becomes amenable to automated reasoning, in an interoperable way.

Triples are stored in a so-called "triple-store". A triple-store is a database specially designed to contain linked metadata in triples form. Currently, the most appropriate triple store is Virtuoso, an open-source triple store with demonstrated performance and scalability. RICORDO provides (via its RDFSTORE program) a template API intended to facilitate querying a triple store with user-friendly forms.

To create annotations using ontology terms, it is necessary to search an ontology and find which term to use. RICORDO provides tools for doing this. RICORDO's Local Ontology Lookup Service provides an API for quickly searching for ontology terms.

RICORDO further provides, through its OWLKB program, an API for automated semantic reasoning, and can generate so-called "composite terms", semantically meaningful combinations of existing ontology terms. For example, suppose we wish to annotate a procedure with a date of occurrence, and suppose we have reference ontologies with terms for "date" and for "occurrence", but not for "date of occurrence". OWLKB can be used to generate a composite term for "date of occurrence" from the constituent terms "date" and "occurrence", and most importantly, the composite term is semantically meaningful: automated reasoners can understand what it means.

The RICORDO toolset and the Virtuoso triplestore have been deployed to a VPS (Virtual Private Server) hosted by CHIC. They provide service to other CHIC components via APIs over the HTTP protocol.

4.2.1 Input and output of RICORDO tools

RICORDO tools accept input through an API over the HTTP protocol, and send their responses as HTTP responses. For specific details about the API, and specific format of the input to RICORDO, see RICORDO's documentation.

Roughly speaking, RICORDO sends its output in JSON format. JSON stands for JavaScript Object Notation and it is the most appropriate format for use by web-based applications. We say, "roughly speaking", because there are a couple of caveats.

1. For legacy reasons, RICORDO's OWLKB by default sends its responses in an HTML format; in order to coax OWLKB into sending responses in JSON it is necessary to send a specific header one's API request, specifically the header "Accept: application/json".

2. RICORDO's RDFStore acts as middleman between CHIC application and triple-store: in particular, its output is simply the triplestore output. Most triplestores, including Virtuoso, offer JSON output; Rdfstore's documentation explains how to configure RDFStore to request JSON output from Virtuoso.

So in summary: RICORDO's tools accept input over an API; documentation has been provided for that input; and, with a couple minor caveats, RICORDO's tools send their output in JSON format.

4.3 The CHIC semantic metadata lifecycle

Semantic metadata in CHIC has a lifecycle consisting of three stages.

1. Creation

Some annotations will be created manually, but most will be automatically generated by various CHIC components. In order to facilitate this, CHIC's developers need APIs for querying the background reference ontologies in order to find which terms to use; RICORDO provides such APIs.

2. Insertion into triple store

Once generated, an annotation will be entered into the triplestore for storage and subsequent querying. Individual triples can be inserted one-by-one via SPARQL (or via more user-friendly forms generated from triples). But most triples will probably be bulk-loaded: multiple (possibly very many) automatically generated triples are first written to a file, in RDF format, and the file is then bulk-loaded into the Virtuoso triple store using its bulk-loading features. This allows Virtuoso to take advantage of multiple CPU cores, as well as to loading algorithms designed for bulk-loading, in order to add large amounts of triples to the store very quickly.

3. Querying

Once stored in the triplestore, triples are accessed by querying. The triplestore can be queried directly, using the SPARQL query language. This, however, will be invisible to the end-user: such queries will mostly be done programmatically "under the hood" by other programs in CHIC. (As an alternative to SPARQL, such other programs can also use RICORDO's RDFSTORE template system; this decision is up to the developers of the other components of CHIC.) In particular, our intention is that the ultimate end-user be able to query and use the metadata via friendly graphical front-ends requiring no knowledge of the underlying technology.

4.4 *What can be annotated?*

Flexibility is one of the key virtues of the RDF data format. Rather than referring to objects with machine-specific designators (filenames, memory locations, database indices, etc.), objects are referred to by IRI (Internationalized Resource Identifiers, another W3C standard), making RDF agnostic about technical implementation details. In short, anything that can be given a stable IRI, can be annotated.

5 Clinical Data Repository

5.1 Introduction

The clinical data repository will permanently host all the medical data produced or collected by the CHIC project. The data provided by the clinical environment will pass through de-identification and (pseudo)-anonymization processes, as described in chapter 3. Additionally, interfaces that will allow to import and export the contents of the clinical data repository will be developed. In this way the data can be sustained after the expiration of the project's lifetime and reused and exploited continuously within the limits allowed by the legal framework of the project. The export services that will be created will also assist in this direction, as many of the data sets to be gathered by the CHIC project will be reusable by future projects. The clinical data repository will contain for each patient all the relevant medical data including imaging data, clinical data, histological data and genetic data.

The focus in this deliverable is on the prototype implementation of the Clinical Data Repository. Foremost, the authentication mechanism is explained in chapter 5.2 and all general concepts introduced in deliverable D8.1 are translated to the use case format in chapter 5.3. Based on those use cases the prototype implementation will be described in the user guide chapter 5.4 and the last chapter concludes with the deployment scenario including a detailed description of the installation procedure.

5.2 Authentication

The clinical data repository makes use of the security framework introduced in Deliverable “D5.2 - Security guidelines and initial version of security tools”. Therefore, the users are not directly authenticated by the clinical data repository (Service Provider) itself but rather by the CHIC authentication broker (Identity Provider) to support Single Sign-On (SSO). This procedure is called brokered authentication and the flow is illustrated in Figure 2.

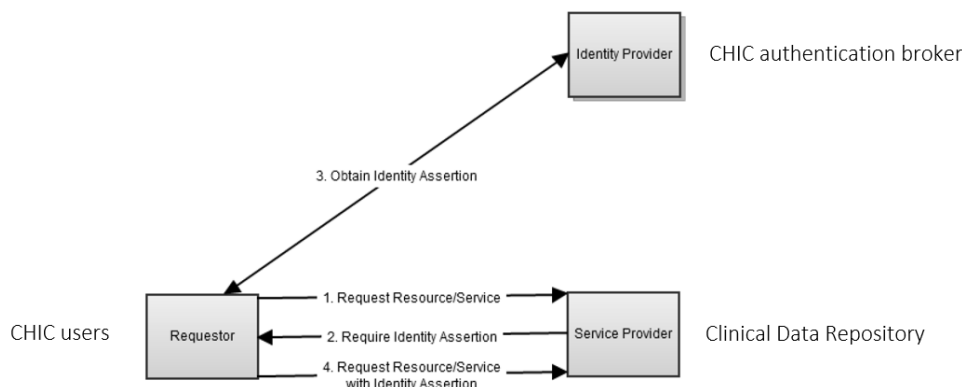


Figure 2: Brokered Authentication Flow with the clinical data repository

The CHIC security framework further distinguishes between brokered authentication for web services including REST and for web sites. As the clinical data repository provides complete access to the features of the database with the help of a REST interface, the Security Token Service (STS) provided by CHIC is fully integrated in the authentication process. Before calling the REST service of the clinical data repository the client needs to send a SOAP (Simple Object Access Protocol) request containing an RST (RequestSecurityToken) to the STS. The STS then returns the identity assertion as a SAML token, embedded in a RSTR (RequestSecurityTokenResponse). The SAML token can then be passed to the REST service through the HTTP authorization header.

The brokered authentication for the clinical data repository web site makes use of SAML Web Browser SSO Profile as suggested by the CHIC security framework. The SAML Web Browser SSO Profile is initiated by an end user who visits the protected clinical data repository web site, also called a Service Provider (SP). The SP redirects the user to the assertion provider (also called Identity Provider (IdP)) passing through an authentication request. The IdP will request the user to authenticate and upon successful authentication the IdP will issue an identity assertion for the user containing all information on the user needed by the SP to authenticate and authorise him. The assertion is then sent back to the SP that will use it to determine whether the user is allowed to access the requested resource.

5.3 Use Cases

This chapter describes the use cases of the clinical data repository corresponding to the recommended approach by Craig Larman [2]. The main goal is to bring the general concepts introduced in D8.1 “Design of the CHIC repositories” into the widely applied use case format.

5.3.1 Actors

Taking into account the CHIC Technical Annex, the deliverable D2.2 “Scenario based user needs and requirements”, the deliverable D5.1 “The CHIC technical architecture”, the deliverable D10.1 “The CHIC Portal” as well as the interaction taken place so far in the CHIC project, we expect the following general categories of users for the clinical data repository.

Table 5: The general categories of users called actors for the Clinical Data Repository

Actor	Remark
Data providers	The data providers are the end users who will be providing data into the clinical data repository. The main tasks of those users will be to de-identify, upload and annotate data for usage by other users or components of the repository.
Researchers	This user group is the main focus and purpose of the CHIC project. Researchers will use the clinical data repository to explore the available datasets, to link and combine them, to download them for further processing.
Clinicians	This user group which is regarded as a subset of the researchers group mentioned above will use the available datasets in order to find answers into specific clinical questions.
TTP	The Trusted Third Party is the mediator between the data providers and the clinical data repository. The main task of this system is to pseudonymize the data before providing it to the repository.
Data validators	The data validators are responsible to review the uploaded data before making it available to the CHIC community. This also involves setting

the appropriate permissions.

5.3.2 Overview

The use cases covered in this chapter are summarized in Figure 3. All primary actors are listed on the left side and all supporting actors are listed on the right side.

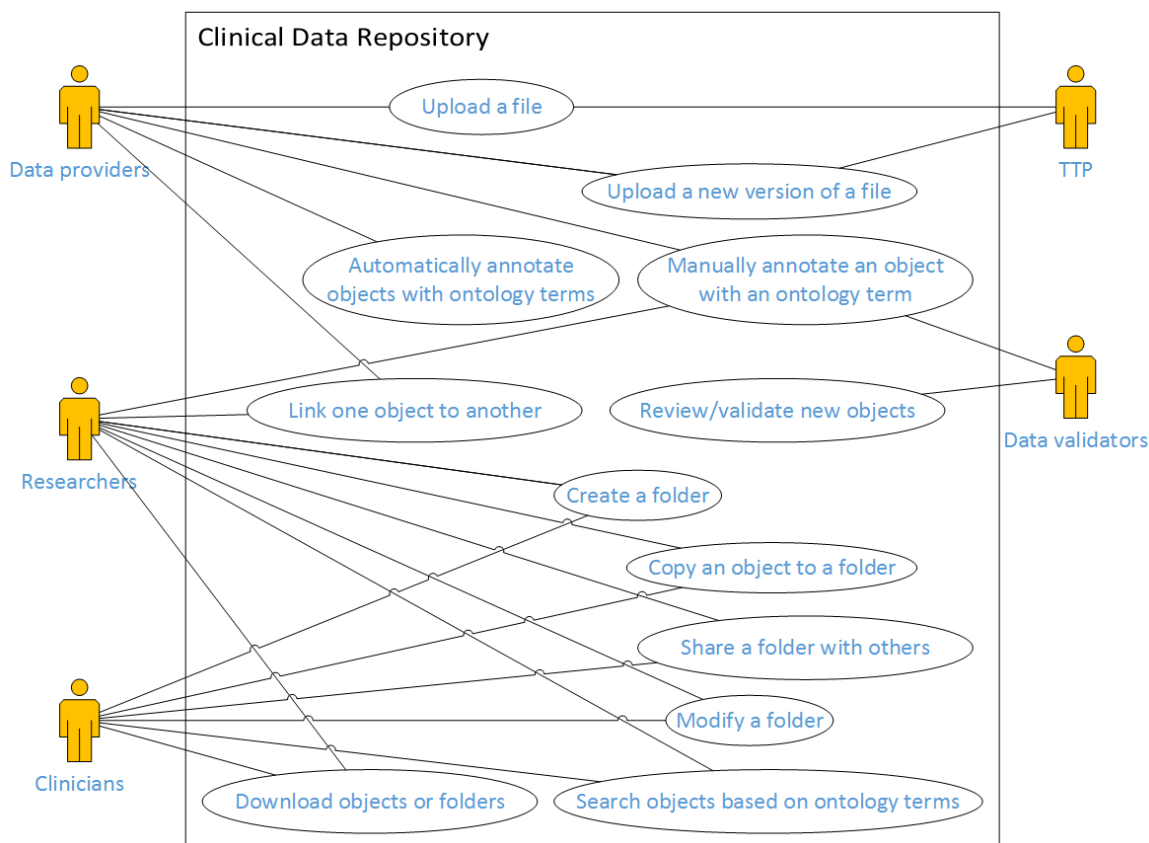


Figure 3: The use case diagram of the clinical data repository.

A summary of all use cases indicated in this figure is given below as well as the page reference of the detailed description:

CDR-UC 1: Upload a file	24
CDR-UC 2: Upload a new version of a file	25
CDR-UC 3: Link one object to another	26
CDR-UC 4: Manually annotate an object with an ontology term	27
CDR-UC 5: Automatically annotate objects with ontology terms	28
CDR-UC 6: Search objects based on ontology terms	29
CDR-UC 7: Review/validate new objects	30
CDR-UC 8: Create a folder	31
CDR-UC 9: Copy an object to a folder	32

CDR-UC 10: Modify a folder.....	33
CDR-UC 11: Share a folder with others.....	33
CDR-UC 12: Download objects or folders	34
The template represented by Table 5 will be used for all use cases. If a row in the template isn't needed for the use case it will be simply removed.	

Table 6: Template for all use cases

Use Case Id:	A (project-)unique identifier of the use case
Use Case Name:	A short but descriptive name for the use case.
Actors:	The actors of the use case.
Description:	The description of the goal to be achieved by the use case.
Pre-conditions:	Required conditions before the start of the use case.
Standard flow:	Describes the standard flow of the use case step-by-step.
Alternate flow:	Describes the alternate flow of the use case step-by-step.
Post-conditions:	Required conditions at the end of the use case.
Extensions:	Describes extensions to specific steps either in the standard or alternate flow of the use case.

5.3.3 Upload & Versioning

The clinical data repository is built around the concept of data objects (*ObjectVersion*), which constitute the basic component of the system. These data objects can be any type of image file, processed data, study data etc. This approach provides a large flexibility to the system in terms of data formats, data organization and data exchange [3].

The system has been designed to support versioning. Data uploaded to the system are never deleted, but multiple versions of an object can be stored in the database. This approach limits problems associated with accidental deletion of data, while maintaining the flexibility to keep updating data files. For example, the initial data of the clinical study concerning a patient can be uploaded before the final examinations. Once the last examination has been performed, a new version of the file is uploaded to the system, which enables modellers to have access to the latest information while keeping the ability to see the history of the modifications.

As described in chapter “3 General workflow for data upload” data providers do not directly upload to the clinical data repository. In a first step the data is uploaded from the hospital to the Trusted Third Party (TTP). In a second step the data is uploaded from the TTP to the clinical data repository.

Use Case Id:	CDR-UC 1
Use Case Name:	Upload a file
Actors:	Data providers, TTP
Description:	As a data provider/TTP I would like to upload a file to be stored in the clinical data repository.
Pre-conditions:	The user must be authenticated and authorized.
Standard flow:	<ol style="list-style-type: none"> 1. The data provider anonymizes the file. 2. The data provider uploads a file from the hospital to the TTP. 3. The TTP pseudonymizes the file. 4. The TTP uploads the file to the clinical data repository. 5. The clinical data repository stores the file and creates an object.
Post-conditions:	The object representing the uploaded file exists.
Extensions:	5. It depends on the file format whether a new object will be created or an existing one will be used to represent the uploaded file. This is because an object can represent multiple files (e.g. DICOM stack).

CDR-UC 1: Upload a file

The clinical data repository uses *GenericObjects* to extend the generic *ObjectVersion* concept. A new version of a dataset (*ObjectVersion*) will have the same *GenericObject* as the ancestor. If, for example, a clinical trial form has been update with new questions in the clinical trial software, a new version will be generated keeping the same unique identifier as the original object. Based on this identifier, a new version can be created in the clinical data repository. With this approach, the users will still have access to the initial data version (through the *GenericObject*), but the new version will be shown as the current version.

Use Case Id:	CDR-UC 2
Use Case Name:	Upload a new version of a file
Actors:	Data providers, TTP

Description:	As a data provider/TTP I would like to upload a new version of a file to be stored in the clinical data repository.
Pre-conditions:	The user must be authenticated and authorized. The previous version of the file must exist in the clinical data repository.
Standard flow:	<p>Same steps as in CDR-UC 1.</p> <p>5. The clinical data repository detects that a previous version of the new file already exists.</p> <p>6. The clinical data repository creates a new object pointing to the same <i>GenericObject</i> as the previous version.</p>
Post-conditions:	The object representing the new version of the file exists.
Extensions:	6. It depends on the file format whether a new object will be created or an existing one will be used to represent the uploaded file.

CDR-UC 2: Upload a new version of a file

5.3.4 Linking

Each new dataset can be linked with any object already present in the repository. For example anatomical structures can be segmented out of one or multiple medical images. Linking mechanisms ensure that an uploaded segmentation file is not only associated with the correct patient's data, but also that the original images used to perform the segmentation task can be identified by the users of the system. In the case of multimodal image segmentation, this implies that multiple links are created to relate the segmentation file with each of the multi-modal original images. If available, the system will make use of the meta-information stored in the files to automatically generate this linking.

Use Case Id:	CDR-UC 3
Use Case Name:	Link one object to another
Actors:	Data providers, researchers
Description:	As a data provider/researcher I would like to link one object to another because they are related.
Pre-conditions:	The user must be authenticated and authorized.

Standard flow:	<ol style="list-style-type: none"> 1. The user uploads the first file. 2. The system creates the first object. 3. The user uploads the second file. 4. The system creates the second object. 5. The system detects that the two objects are related and creates a link automatically.
Alternate flow:	<ol style="list-style-type: none"> 1. The user selects two or more existing objects. 2. The user creates a link between the selected objects. 3. The system stores the link with a description indicating that it has been created manually by the user.
Post-conditions:	A link indicating the relation between the objects exists. A description of the link can be retrieved by the user, indicating why the link has been created automatically or who has created the link manually.
Extensions:	<p>Standard flow:</p> <p>3.-5.: The system detects that the uploaded file is related with one or multiple objects already existing in the system and creates the links automatically.</p>

CDR-UC 3: Link one object to another

5.3.5 Annotation & Search

In addition to the imaging and clinical data, each data object can be annotated with multiple ontology terms. Initial investigations have been made to integrate an anatomical ontology; the Foundational Model of Anatomy (FMA) [4]. The FMA is a symbolic representation of the canonical, phenotypic structure of an organism; a spatial-structural ontology of anatomical entities and relations which form the physical organization of an organism at all salient levels of granularity. The ontology relies on a triplestore storage system and not in relations or tables. Therefore, a separate system will be used to store the semantic information. Web based queries based on SPARQL [5] will be used to retrieve the information from the ontology for annotation and semantic search. The approach is very flexible and allows to easily include multiple ontologies. In addition to the FMA, additional ontologies will be included in a second step by integrating the RICORDO system. Based on these annotations it will be possible to conduct semantically driven search queries to find datasets containing the required anatomical structures or other properties.

Use Case Id:	CDR-UC 4
Use Case Name:	Manually annotate an object with an ontology term
Actors:	Data providers, researchers, data validators

Description:	As a data provider/researcher/data validator I would like to manually annotate an object with an ontology term.
Pre-conditions:	The user must be authenticated and authorized. The object to be annotated and the ontology term must exist in the system.
Standard flow:	<ol style="list-style-type: none"> 1. The user selects the object to be annotated. 2. The user searches for the ontology term to be used. 3. The user annotates the object with the found ontology term. 4. The system stores the annotation. 5. The system exports the annotation triple to the external RDF store asynchronously in a deferred separate task.
Post-conditions:	The annotation triple exists in the external RDF store.
Extensions:	<ol style="list-style-type: none"> 2. The ontology term is either retrieved from the system itself or from the external service called LOLS (Local Ontology Lookup Service).

CDR-UC 4: Manually annotate an object with an ontology term

Use Case Id:	CDR-UC 5
Use Case Name:	Automatically annotate objects with ontology terms
Actors:	Data providers
Description:	As a data provider I would like that objects are annotated with ontology terms automatically.
Pre-conditions:	The user must be authenticated and authorized. The ontology terms must exist in the system.
Standard flow:	<ol style="list-style-type: none"> 1. The user uploads a file. 2. The system creates an object. 3. The system extracts ontology terms provided within the file. 4. The system annotates the object with the extracted ontology terms. 5. The system exports the annotation triples to the external RDF store asynchronously in a deferred separate task.

Post-conditions:	The annotation triples exist in the external RDF store.
------------------	---

CDR-UC 5: Automatically annotate objects with ontology terms

Use Case Id:	CDR-UC 6
Use Case Name:	Search objects based on ontology terms
Actors:	Researchers, clinicians
Description:	As a researcher/clinician I would like to search for objects based on a semantically driven query containing ontology terms.
Pre-conditions:	-
Standard flow:	<ol style="list-style-type: none"> 1. The user enters an ontology term to search for. 2. The system assembles the SPARQL query. 3. The system runs the SPARQL query against the RDF store. 4. The RDF store processes the query. 5. The RDF store returns the triples which fulfil the query. 6. The system displays all objects extracted from the triples received from the RDF store.
Post-conditions:	The search output displays all objects which fulfil the semantically driven query containing ontology terms.
Extensions:	<p>In order to access the data behind the found objects, the user must be authenticated and authorized.</p> <p>Standard flow:</p> <p>1.-2.:</p> <ul style="list-style-type: none"> • The user either fills a predefined template; or • The user assembles the SPARQL query manually.

CDR-UC 6: Search objects based on ontology terms

5.3.6 Validation

To ensure a high level of quality to the data stored in the repository, the system will support a multi-step validation process. During the validation process the user can review the metadata extracted from the data, include additional relevant information and finally publish the data object. Once published, the new data object is accessible by the other users of the system having the appropriate permissions.

Use Case Id:	CDR-UC 7
Use Case Name:	Review/validate new objects
Actors:	Data validators
Description:	As a data validator I would like to review/validate the uploaded data before making it available to the CHIC community.
Pre-conditions:	The user must be authenticated and authorized.
Standard flow:	<ol style="list-style-type: none"> 1. The user selects one or multiple objects to be validated. 2. The user reviews the metadata extracted from the data. 3. The user includes additional relevant information. 4. The user sets the permissions for other users and/or groups. 5. The user publishes the selected objects.
Alternate flow:	<ol style="list-style-type: none"> 1. The user selects one or multiple objects to be deleted. 2. The user hits the delete button. 3. The system deletes the selected objects.
Post-conditions:	<p>The published objects appear in the list view of validated objects and authorized users can access/organize/download them.</p> <p>The deleted objects no longer appear in the list view of unvalidated objects.</p>
Extensions:	The system validates new objects automatically, if configured to do so.

CDR-UC 7: Review/validate new objects

5.3.7 Data organization

The clinical data repository allows each user to freely organize the data into his/her desired folder structure for easy access to the data needed for his/her research. Hereby, data objects are not physically moved or duplicated, but the system creates a reference to the data object, retaining the original file permission and ownership. A folder structure created by one user can be directly shared to others. Modifications made by one user will immediately be visible in the folder of the other collaborators. The mechanism should allow efficient collaboration between modellers working on the same tumour model. To simplify the collaboration within a group, the system provides a default shared group folder, which is accessible and manageable by all members of the group.

Use Case Id:	CDR-UC 8
Use Case Name:	Create a folder
Actors:	Researchers, clinicians
Description:	As a researcher/clinician I would like to create a folder which can then be used to organize the datasets.
Pre-conditions:	The user must be authenticated and authorized.
Standard flow:	<ol style="list-style-type: none"> 1. The user selects an existing folder, wherein the new folder should be created. 2. The user hits the button to create a folder. 3. The user enters the name for the new folder. 4. The user hits the save button. 5. The system creates a new folder.
Post-conditions:	The folder appears in the parent folder.
Extensions:	5. The systems aborts the operation and raises an alert if a folder with the same name already exists in the parent folder.

CDR-UC 8: Create a folder

Use Case Id:	CDR-UC 9
Use Case Name:	Copy an object to a folder
Actors:	Researchers, clinicians
Description:	As a researcher/clinician I would like to copy an existing object to an existing folder.
Pre-conditions:	The user must be authenticated and authorized. The object to be copied and the folder must exist.
Standard flow:	<ol style="list-style-type: none"> 1. The user selects one or multiple objects to be copied. 2. The user hits the button to copy objects to a folder. 3. The user selects the folder.

	4. The user hits the save button. 5. The system creates a reference of the object in the selected folder.
Post-conditions:	The selected object appears in the selected folder.
Extensions:	5. The systems aborts the operation if a reference of the same object already exists in the selected folder.

CDR-UC 9: Copy an object to a folder

Use Case Id:	CDR-UC 10
Use Case Name:	Modify a folder
Actors:	Researchers, clinicians
Description:	As a researcher/clinician I would like to modify a folder.
Pre-conditions:	The user must be authenticated and authorized. The folder to be modified must exist.
Standard flow:	1. The user selects the folder to be renamed. 2. The user hits the button to rename the folder. 3. The user enters a new name for the folder. 4. The user hits the save button. 5. The system saves the modification.
Alternate flow 1:	1. The user selects the folder to be deleted. 2. The user hits the button to delete the folder. 3. The system deletes the folder.
Alternate flow 2:	1. The user selects the objects to be removed from the folder. 2. The user hits the button to remove objects from the folder. 3. The system removes the selected objects from the folder.
Post-conditions:	Standard flow: The folder appears with the new name. Alternate flow 1: The folder is deleted. Alternate flow 2: The removed objects no longer appear in the folder.
Extensions:	Standard flow:

	<p>5. The systems aborts the operation and raises an alert if a folder with the same name already exists in the parent folder.</p> <p>Alternate flow 1:</p> <p>3. The systems aborts the operation and raises an alert if the folder to be deleted is not empty.</p>
--	--

CDR-UC 10: Modify a folder

Use Case Id:	CDR-UC 11
Use Case Name:	Share a folder with others
Actors:	Researchers, clinicians
Description:	As a researcher/clinician I would like to share a folder with another user or group.
Pre-conditions:	The user must be authenticated and authorized. The folder to be shared must exist.
Standard flow:	<ol style="list-style-type: none"> 1. The user selects the folder to be shared. 2. The user hits the button to share the folder. 3. The user enters the name of the target user or group. 4. The user assigns the appropriate permissions. 5. The user hits the save button. 6. The system shares the folder.
Post-conditions:	The folder appears in the list view of the target user or group.

CDR-UC 11: Share a folder with others

5.3.8 Download

The system provides download functionality for available datasets for further processing. A single object or multiple objects can be downloaded simultaneously. Even a whole folder structure can be downloaded in a compressed container file, preserving the folder structure. To save bandwidth and time the files represented by the objects or folders are compressed on the fly for the download.

Use Case Id:	CDR-UC 12
--------------	-----------

Use Case Name:	Download objects or folders
Actors:	Researchers, clinicians
Description:	As a researcher/clinician I would like to download objects or folders.
Pre-conditions:	The user must be authenticated and authorized. The objects and the folders to be downloaded must exist.
Standard flow:	<ol style="list-style-type: none"> 1. The user selects the object to be downloaded. 2. The user hits the download button. 3. The system creates a compressed zip containing all files represented by the selected object. 4. The system sends the compressed zip to the user.
Alternate flow:	<ol style="list-style-type: none"> 1. The user selects the folder to be downloaded. 2. The user hits the download button. 3. The system creates a compressed zip containing all files represented by the objects in the selected folder. 4. The system sends the compressed zip to the user.
Post-conditions:	<p>The compressed zip file exists on the local file system of the user.</p> <p>Alternate flow:</p> <p>The compressed zip file contains the same folder structure as on the system.</p>

CDR-UC 12: Download objects or folders

5.4 User Guide

The user guide builds upon the uses cases described in the previous chapter and multiple objectives should be achieved with it at the same time. First of all, it should document the web-based user interface and all of its functionalities. Furthermore, the current state of the prototype is illustratively documented. In addition, this user guide should be as descriptive and simple as possible, so that it can be provided to all end-users for training/instruction purposes.

The prototype implementation of the web-based user interface offers a main view illustrated in Figure 4 which serves as entry point for almost all functionalities described throughout the user guide. On top, an input field enables the end-user to search for datasets (1). On the left side, the folder explorer enables the user to organize data (2). MyData is the location of the user's data; MyGroups is the default collaboration folder accessible to all group members; MyProjects are folders to organize data into personal projects; SharedFolder are folders of others which are shared to the user. In the middle of the main view, the toolbox enables the user to initiate batch commands for multiple objects or folders (3). A preview image assists the user to identify datasets (4). Several icons enable the user to display additional information about the corresponding dataset as requested (5). The file name introduced by the Clinical Data Repository is based on a constructed template (6). The template is updated if the information is available otherwise XX is used as a placeholder.

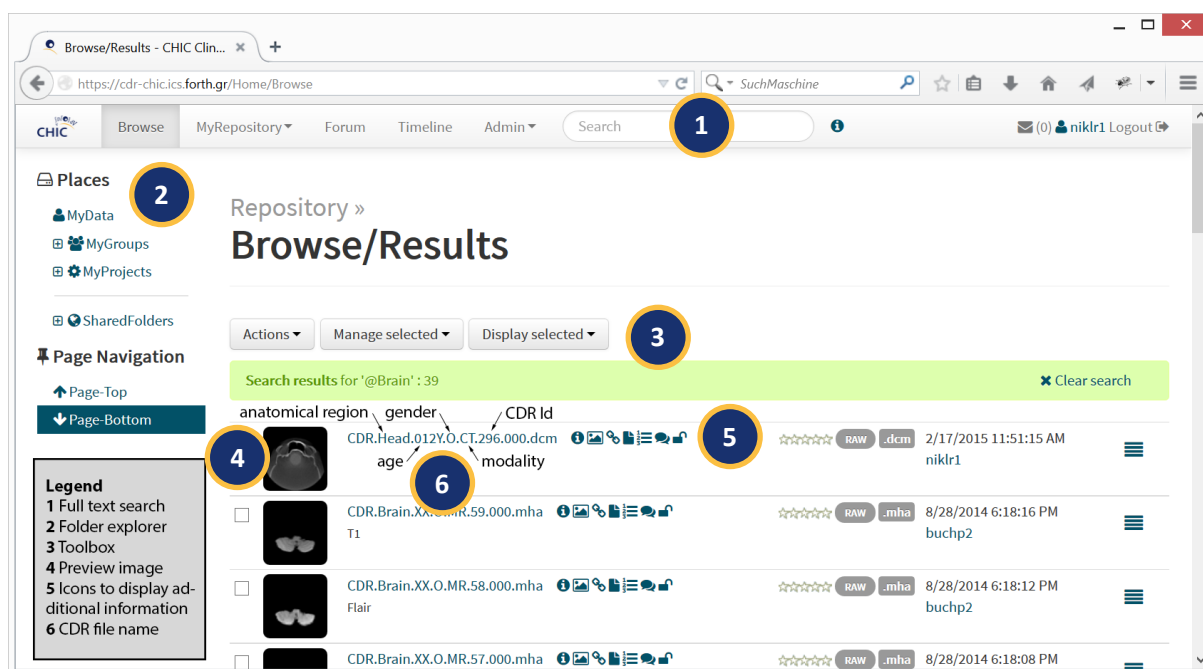


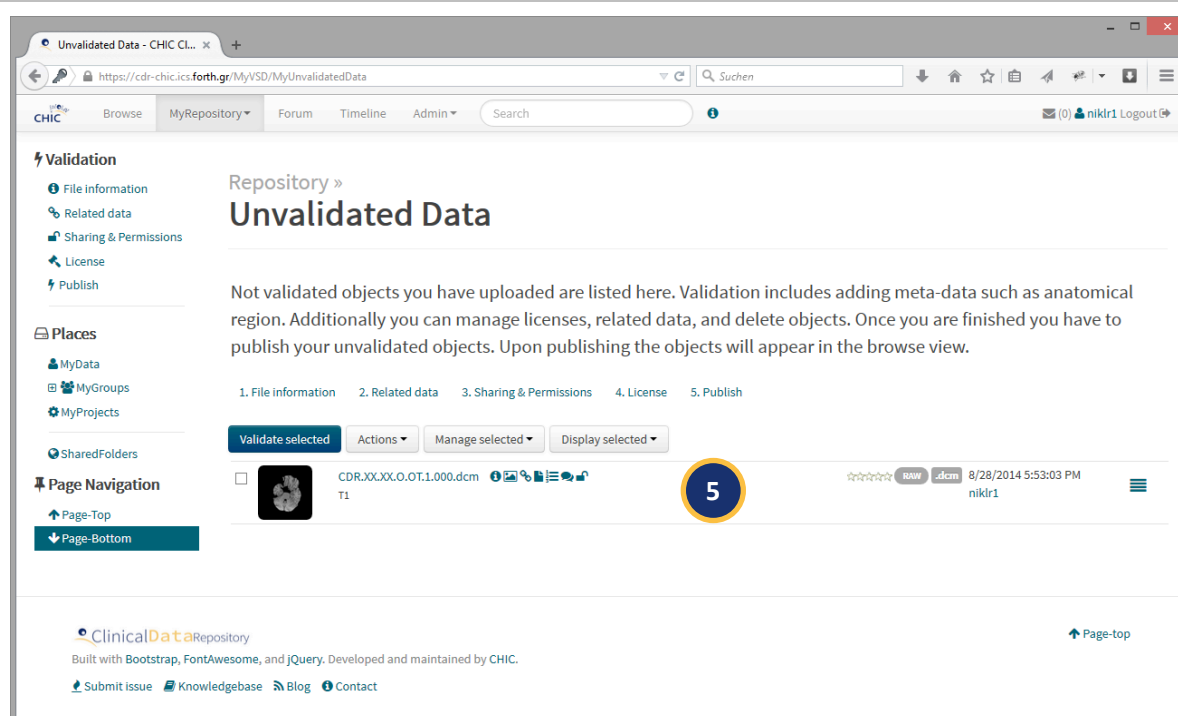
Figure 4: The web-based user interface main view of the Clinical Data Repository

5.4.1 CDR-UC 1: Upload a file

1	The data provider anonymizes the file.
2	The data provider uploads a file from the hospital to the TTP.
3	The TTP pseudonymizes the file.

4

The TTP uploads the file to the clinical data repository.



5

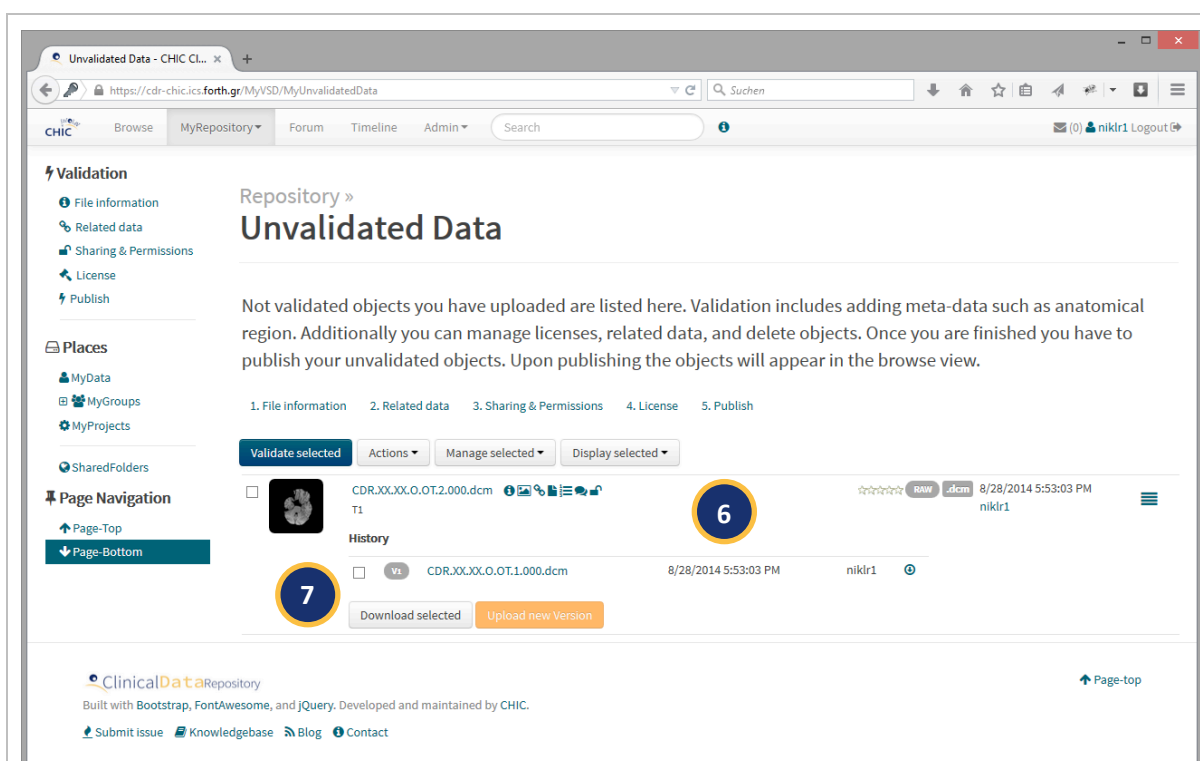
The clinical data repository stores the file and creates an object.

5.4.2 CDR-UC 2: Upload a new version of a file

Same steps as in CDR-UC 1.

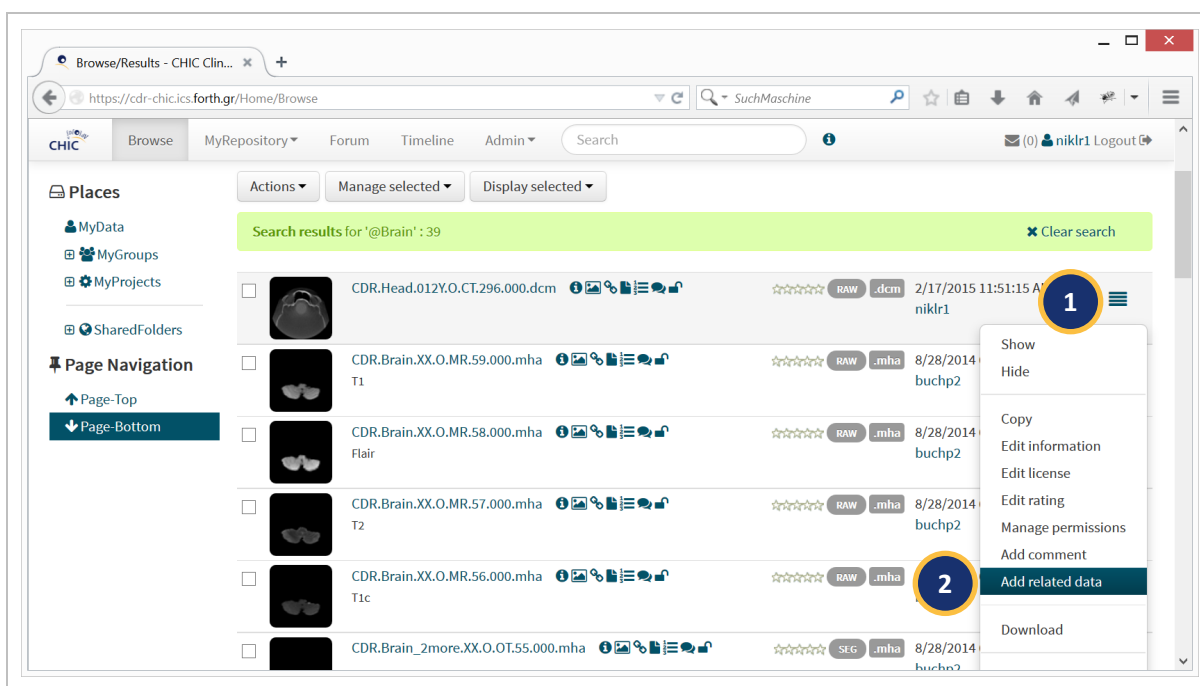
5

The clinical data repository detects that a previous version of the new file already exists.

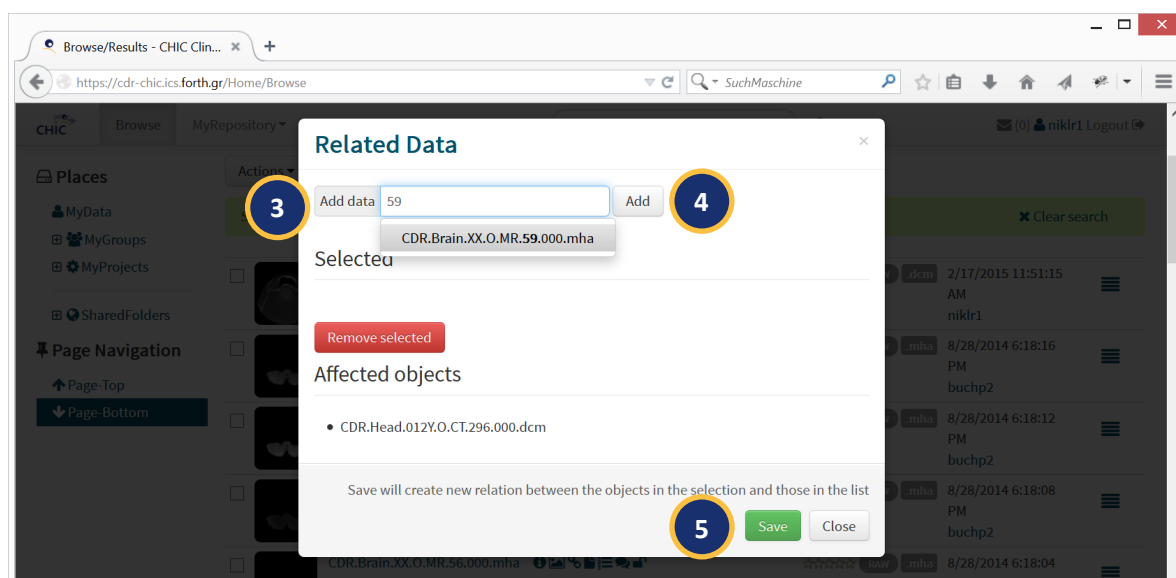


6	The clinical data repository creates a new object pointing to the same <i>GenericObject</i> as the previous version.
7	The previous version is listed in the history section.

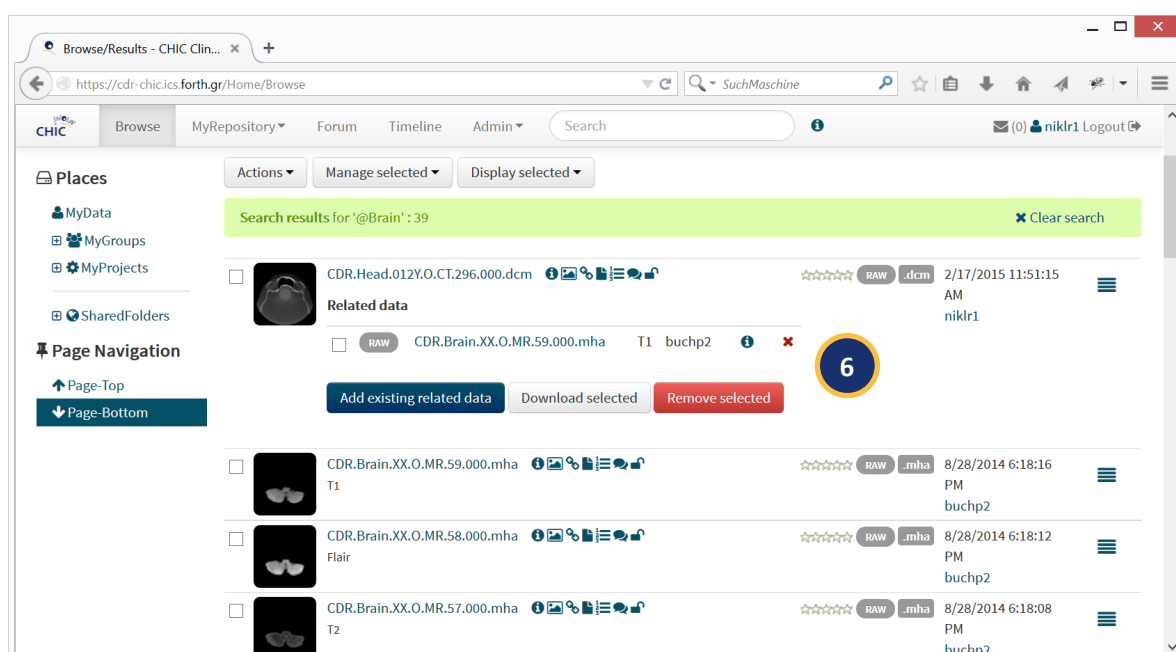
5.4.3 CDR-UC 3: Link one object to another



0	Select the object you want to link to another.
1	Hit the icon on the right side to open the options menu.
2	Choose the “Add related data” option.

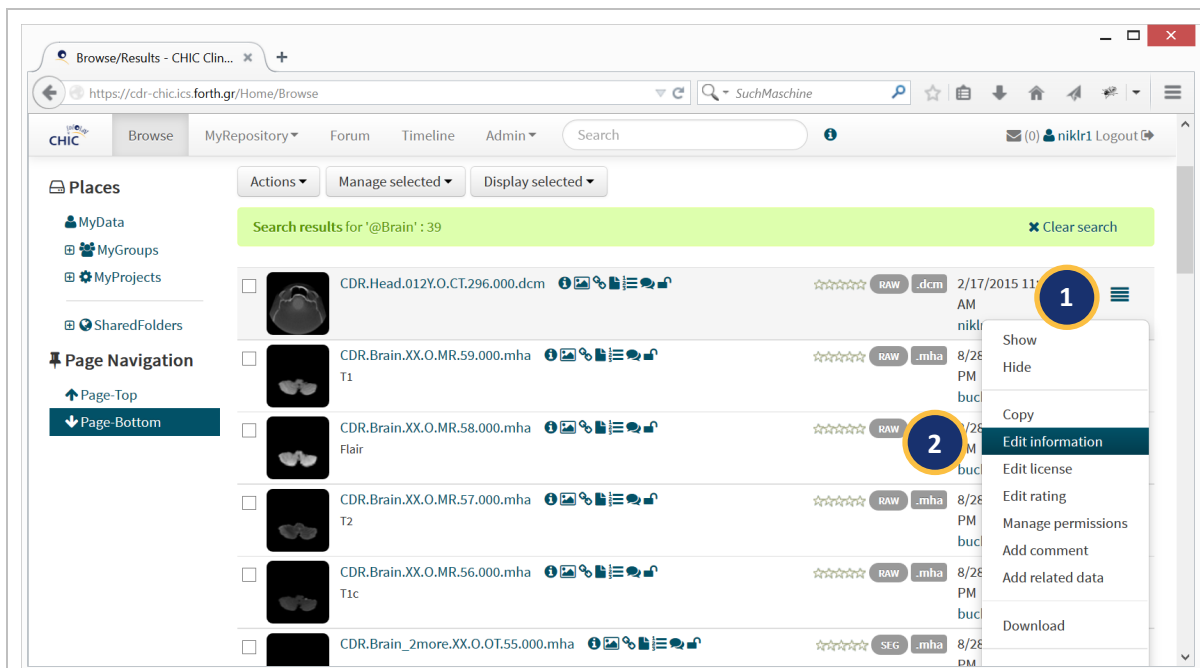


3	Enter the identifier of the object to be linked.
4	Hit the add button.
5	Hit the save button in order to create the link between the two objects.



6	The object link is listed in the related data section.
---	--

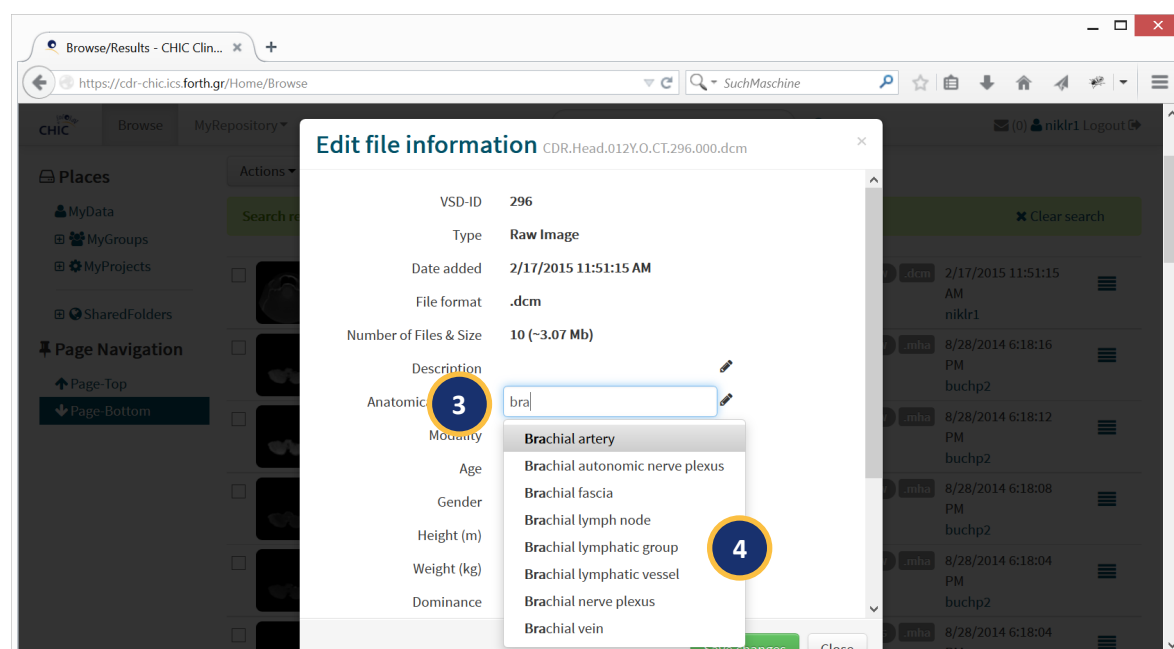
5.4.4 CDR-UC 4: Manually annotate an object with an ontology term



0 Select the object you want to manually annotate.

1 Hit the icon on the right side to open the options menu.

2 Choose the “Edit information” option.



3 Start by typing to search for the ontology term.

4 Select the appropriate ontology term and save the changes.

5.4.5 CDR-UC 5: Automatically annotate objects with ontology terms

1	The user uploads a file.
2	The system creates an object.
3	The system extracts ontology terms provided within the file.

The screenshot shows the CHIC web interface. On the left, there's a sidebar with 'Places' (MyData, MyGroups, MyProjects, SharedFolders) and 'Page Navigation' (Page-Top, Page-Bottom). The main area displays file information for 'CDR.Brain.XX.O.MR.59.000.mha'. The 'Information' section lists: VSD-ID 59, Type Raw Image, Date added 8/28/2014 6:18:16 PM, File format .mha, Number of Files & Size 1 (~1.23 Mb), Description T1, Anatomical Region Brain (highlighted with a blue circle and '4'), Modality MR, Age -, Gender Other, Height -, Weight -. The top navigation bar includes 'Browse', 'MyRepository', 'Forum', 'Timeline', 'Admin', and a search bar.

4	The system annotates the object with the extracted ontology terms.
5	The system exports the annotation triples to the external RD store.

5.4.6 CDR-UC 6: Search objects based on ontology terms

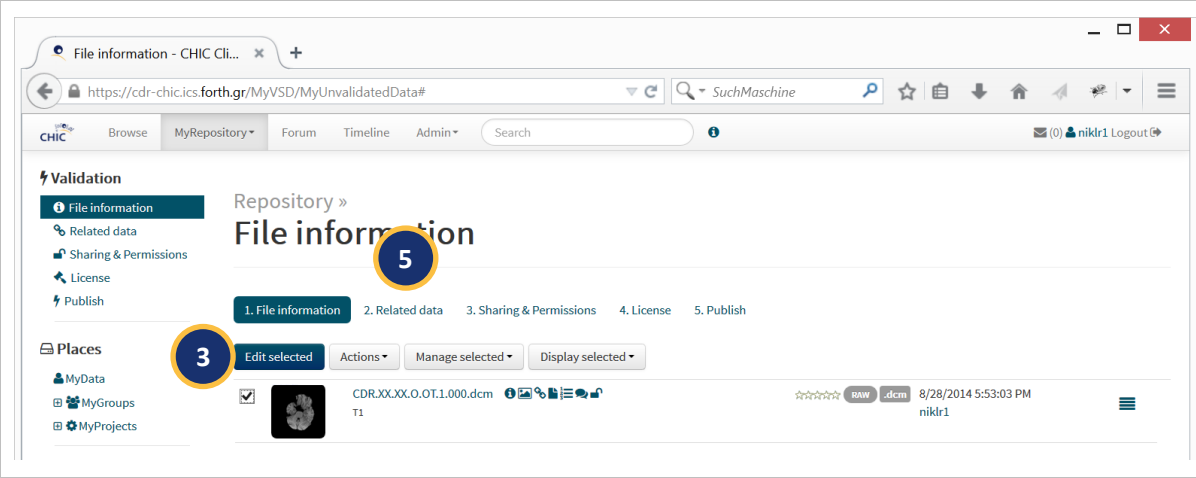
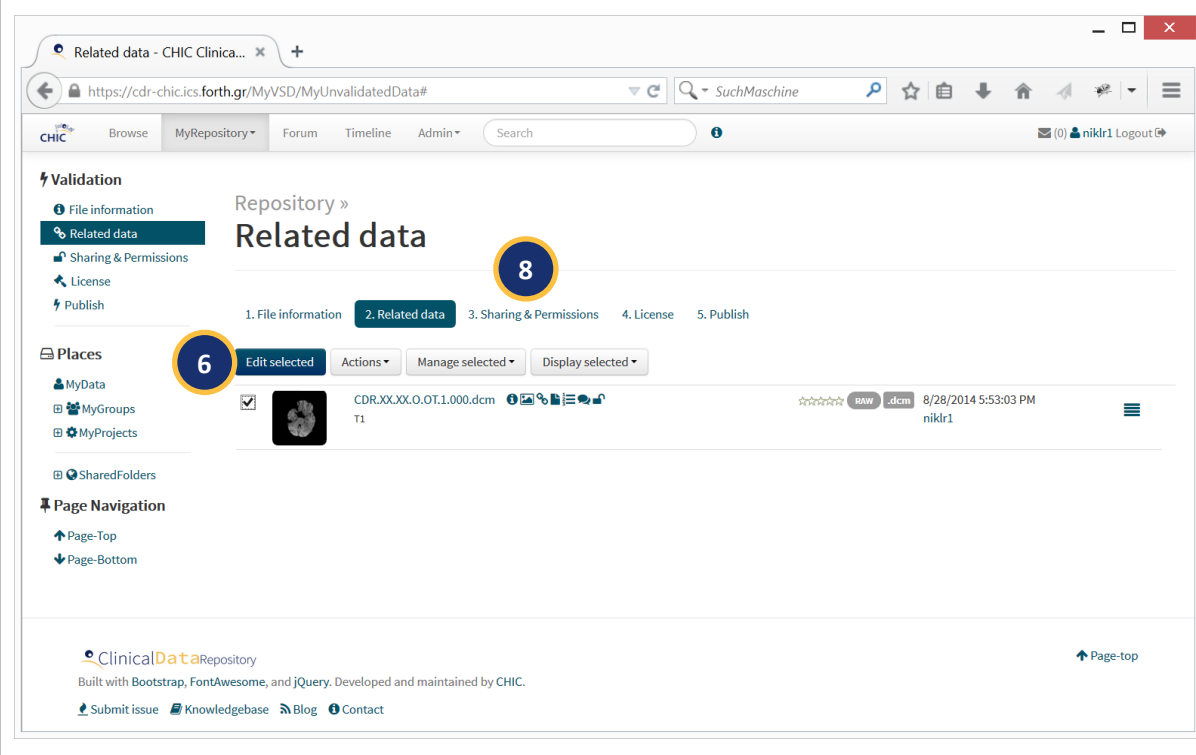
The screenshot shows the CHIC web interface with search results for '@Brain'. The top navigation bar has a search bar containing '@Brain' (highlighted with a blue circle and '1'). Below the navigation bar, the page title is 'Repository » Browse/Results'. There are buttons for 'Actions', 'Manage selected', and 'Display selected'. A green bar shows 'Search results for '@Brain': 39' (highlighted with a blue circle and '6'). Below this, a list of search results is shown, each with a checkbox, a thumbnail, the file name, and the user. The first result is 'CDR.Head.012Y.O.CT.296.000.dcm' by 'niklr1'. The second is 'CDR.Brain.XX.O.MR.59.000.mha T1' by 'buchp2'. The third is 'CDR.Brain.XX.O.MR.58.000.mha Flair' by 'buchp2'. The fourth is 'CDR.Brain.XX.O.MR.57.000.mha T2' by 'buchp2'. The fifth is 'CDR.Brain.XX.O.MR.56.000.mha' by 'buchp2'.

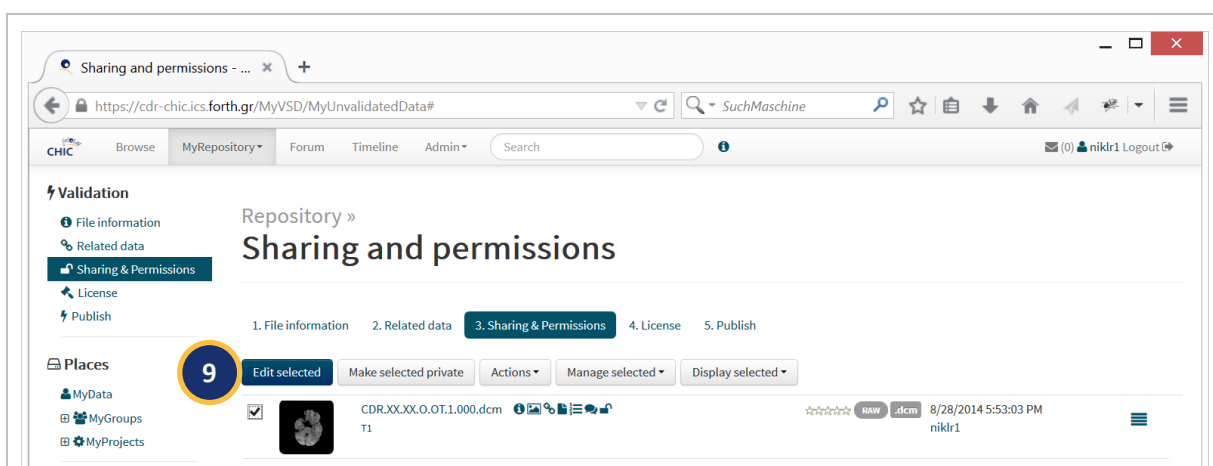
1	Enter the ontology term to be searched for.
2	The system assembles the SPARQL query.
3	The system runs the SPARQL query against the RDF store.
4	The RDF store processes the query.
5	The RDF store returns the triples which fulfil the query.
6	The system displays all objects extracted from the triples received from the RDF store.

5.4.7 CDR-UC 7: Review/validate new objects

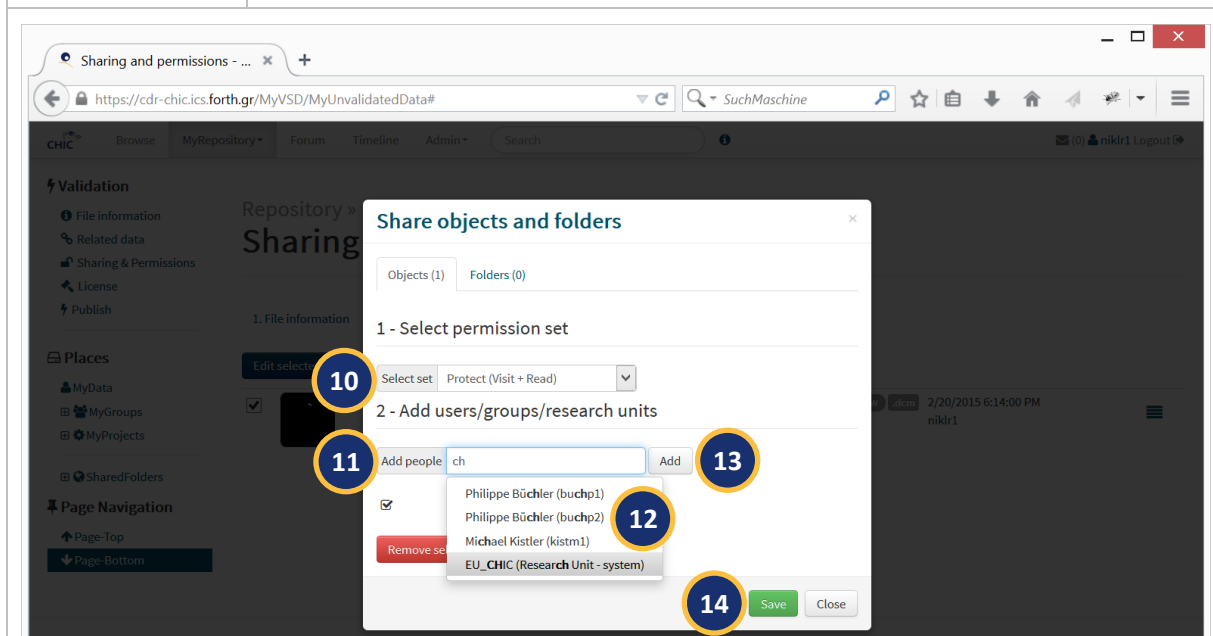
The screenshot shows the CHIC web interface. The browser address bar displays 'https://cdr-chicics.forth.gr/MyVSD/MyUnvalidatedData'. The page title is 'Unvalidated Data'. The main content area contains a description of the validation process and a table of unvalidated objects. The table has columns for 'File information', 'Related data', 'Sharing & Permissions', 'License', and 'Publish'. A 'Validate selected' button is highlighted with a red circle and the number 2. A red circle with the number 1 highlights the 'Page-Top' link in the sidebar.

1	Select one or multiple objects to be validated.
2	Hit the “Validate selected” button in order to start the validation procedure.

	
3	Hit the “Edit selected” button in order to edit the file/object information.
4	Enter a description, modality, age, gender, etc. or manually annotate the object with ontology terms as described in 5.4.4.
5	Hit the “Related data” button in order to proceed to the next validation step.
	
6	Hit the “Edit selected” button in order to edit the related data.
7	Link the selected objects with others as described in 5.4.3.
8	Hit the “Sharing & Permissions” button in order to proceed to the next validation step.



9 Hit the “Edit selected” button in order to edit the sharing & permissions.



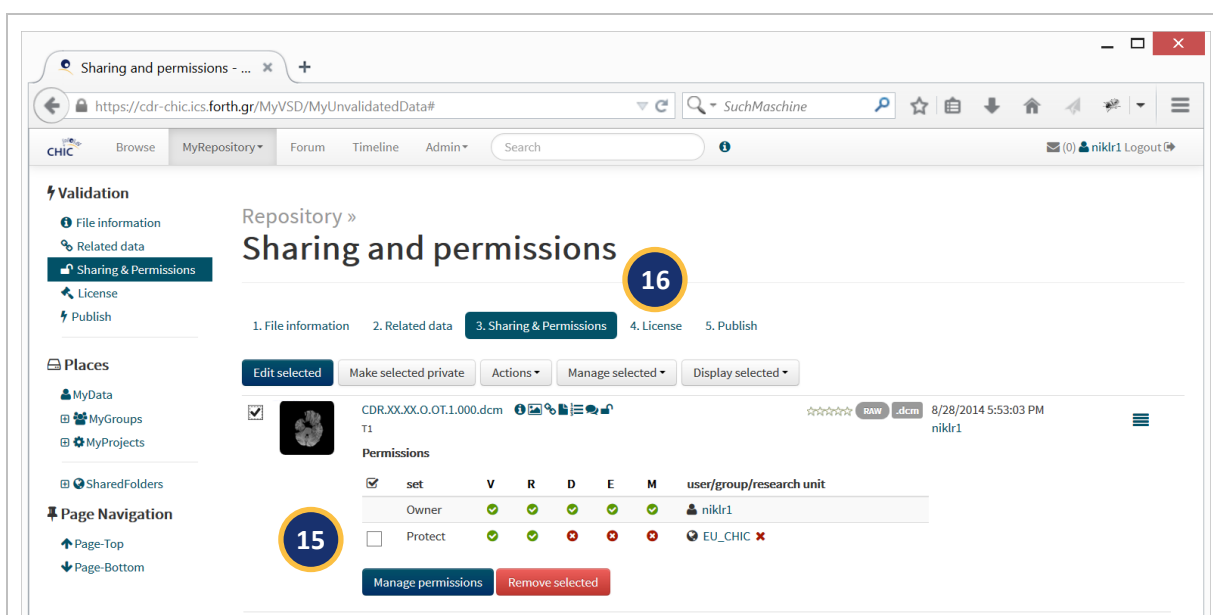
10 Select the appropriate permission set.

11 Start by typing to search for a user, group or research unit.

12 Select the appropriate user, group or research unit.

13 Hit the “Add” button.

14 Hit the “Save” button.

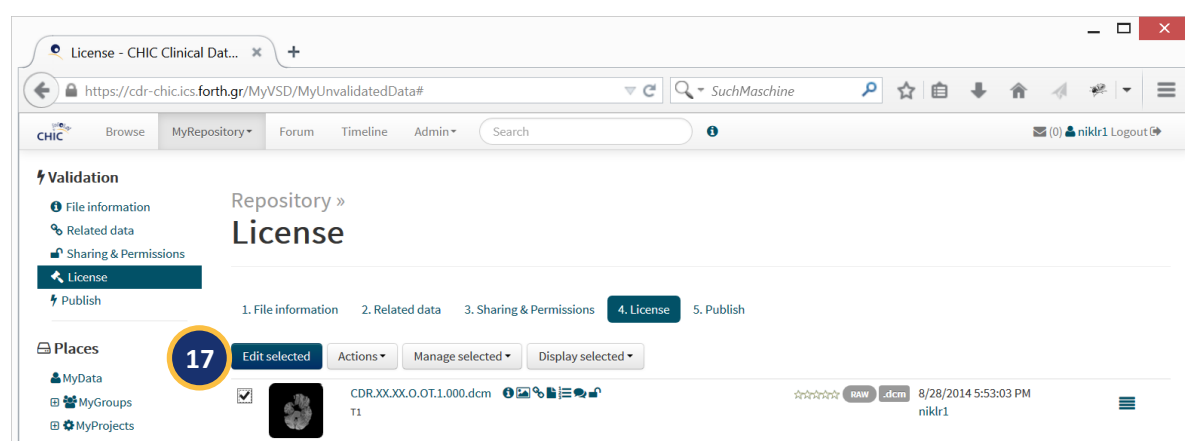


15

The saved sharing & permissions are listed in the permissions section.

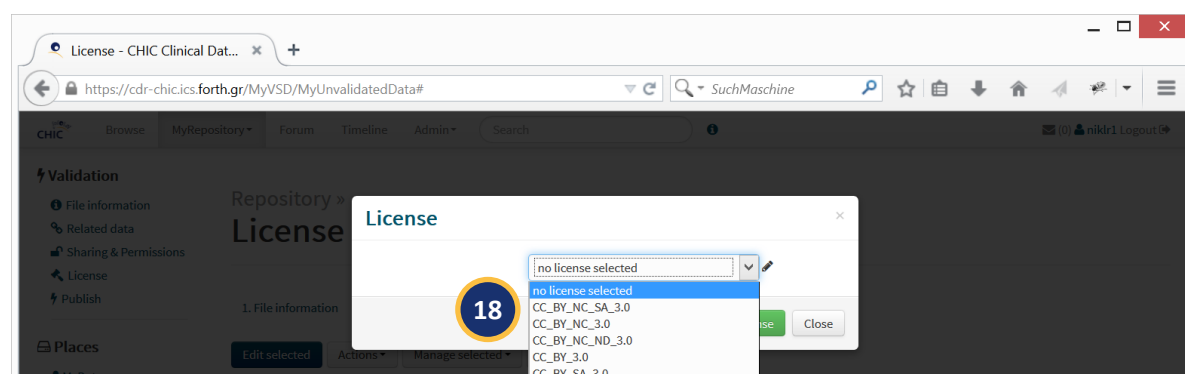
16

Hit the “License” button in order to proceed to the next validation step.



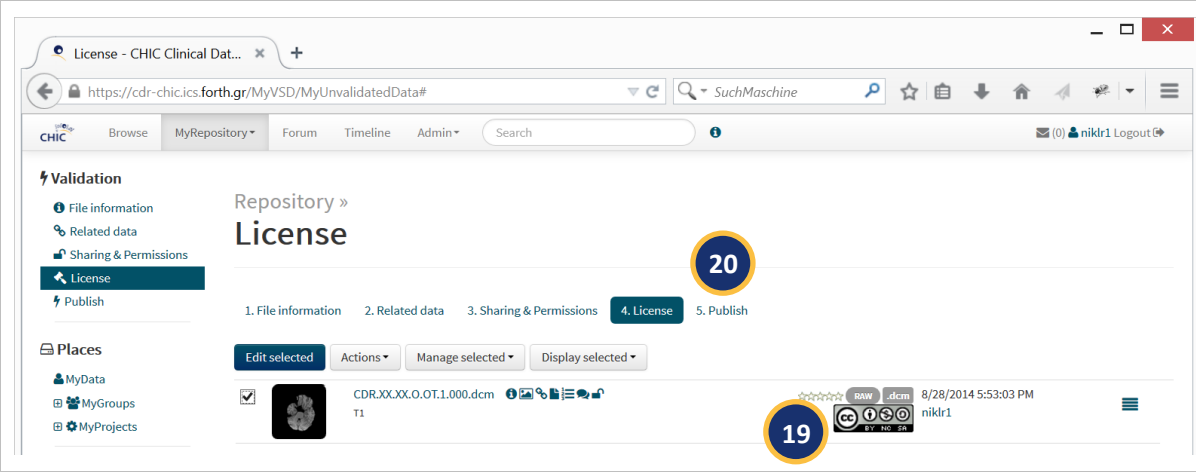
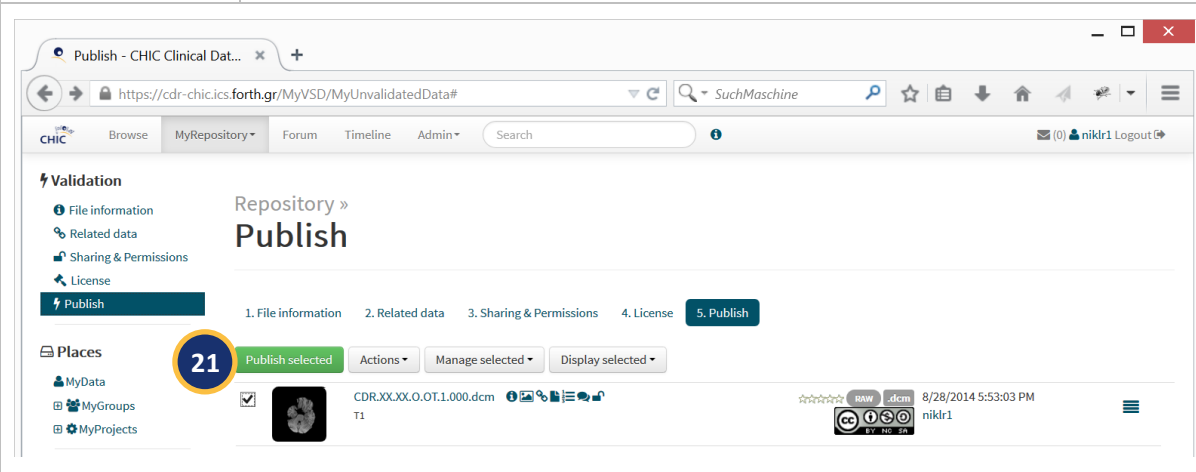
17

Hit the “Edit selected” button in order to edit the license.

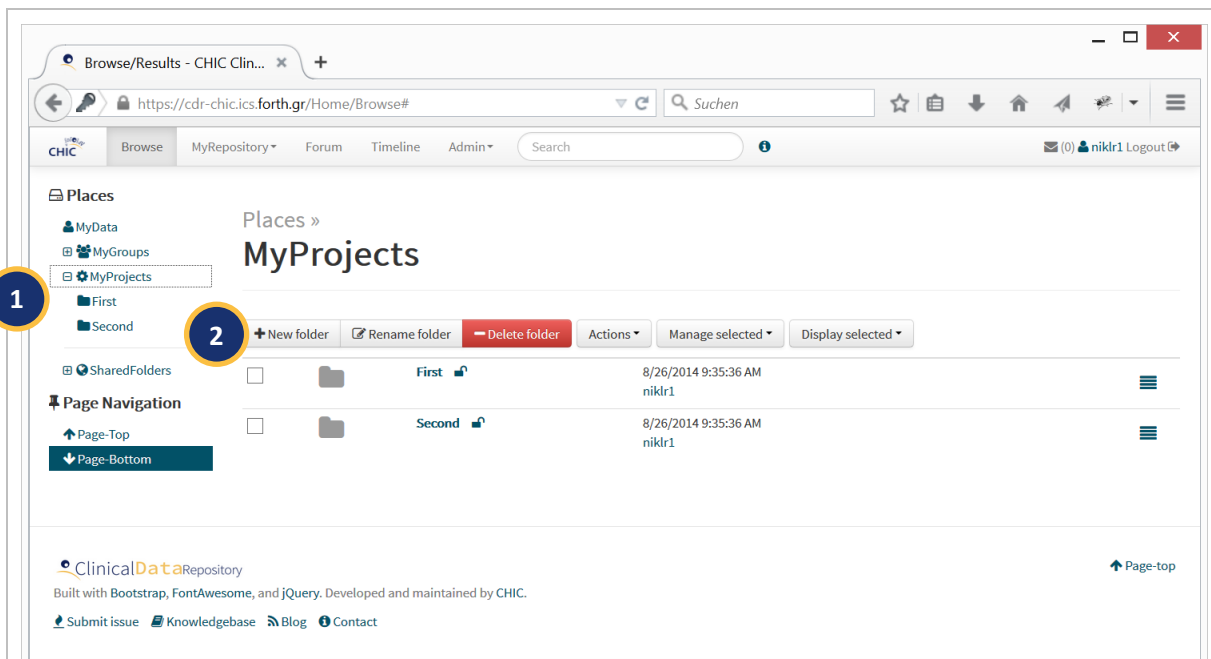


18

Select the appropriate license and save the changes.

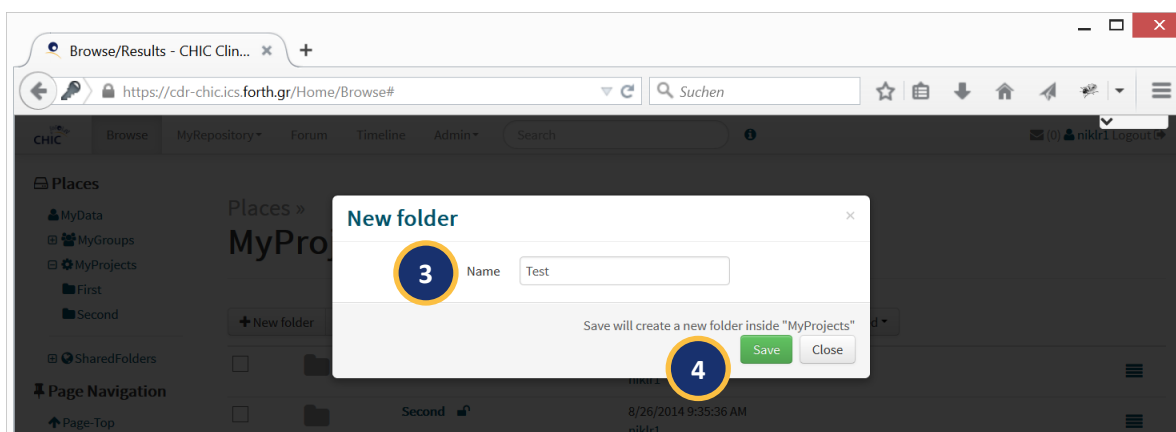
	
19	The selected license appears on the object.
20	Hit the “Publish” button in order to proceed to the last validation step.
	
21	Hit the “Publish selected” button in order to finish the validation procedure.

5.4.8 CDR-UC 8: Create a folder



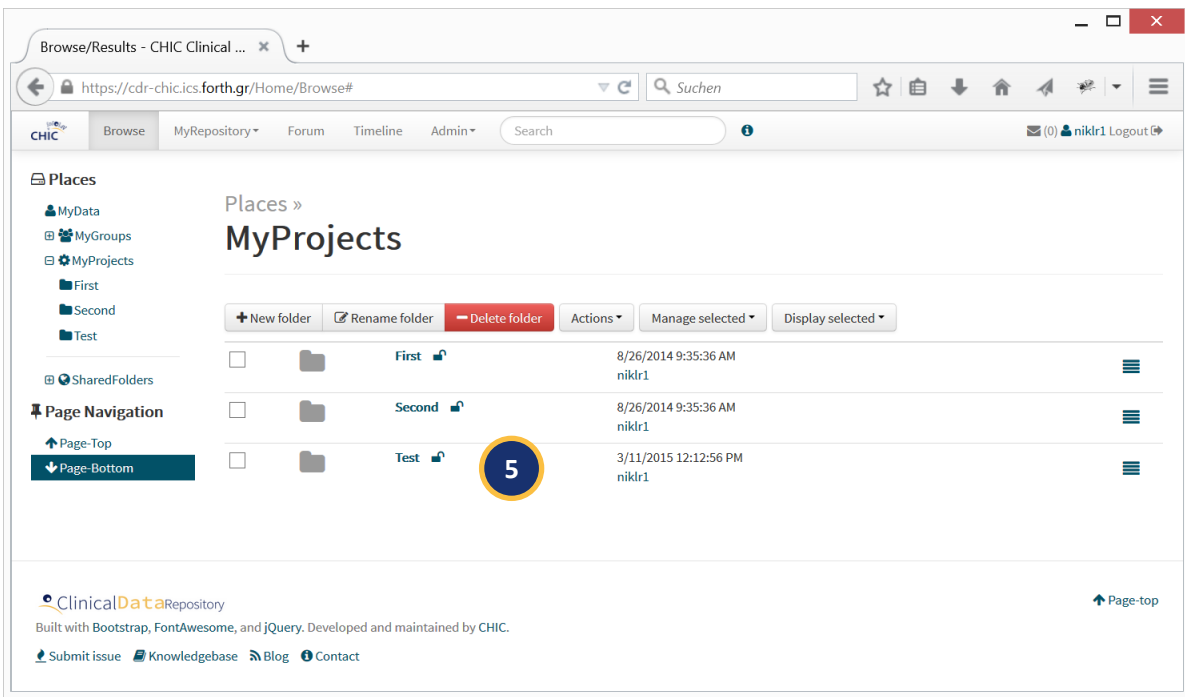
1	Select an existing folder, wherein the new folder should be created.
---	--

2	Hit the “New folder” button in order to create a folder.
---	--



3	Enter a name for the folder.
---	------------------------------

4	Hit the “Save” button.
---	------------------------



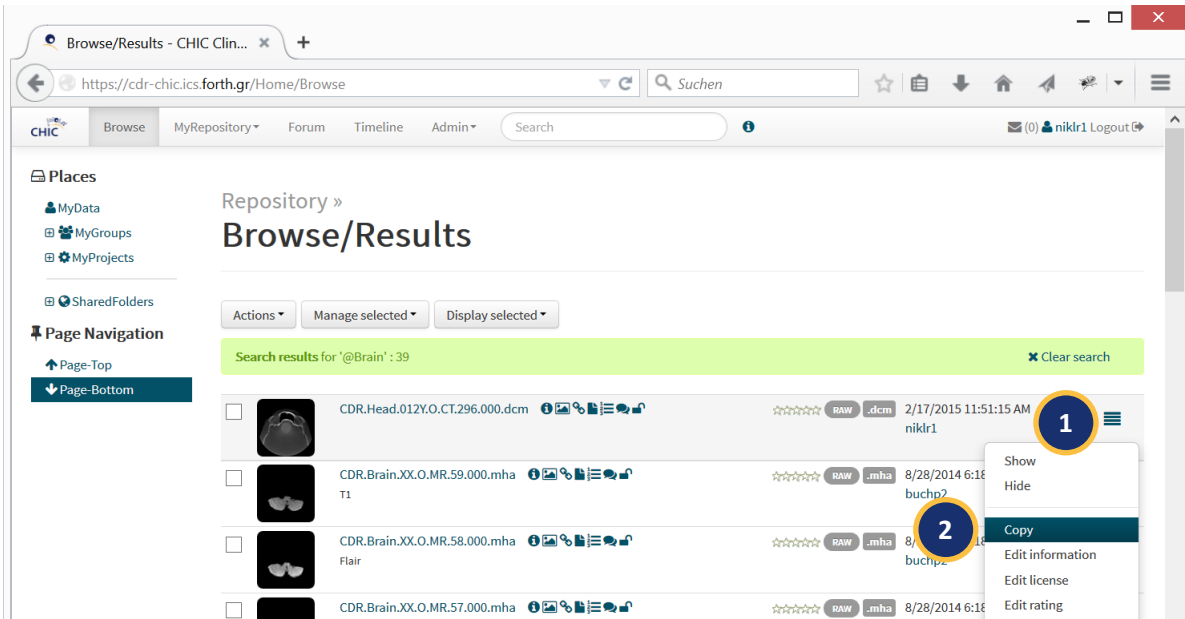
The screenshot shows the CHIC web application interface. The browser address bar displays 'https://cdr-chic.ics.forth.gr/Home/Browse#'. The page title is 'Browse/Results - CHIC Clinical ...'. The main content area is titled 'Places » MyProjects'. On the left sidebar, under 'Places', 'MyProjects' is selected, showing a list of folders: 'First', 'Second', and 'Test'. The 'Test' folder is highlighted with a blue circle and the number 5. The main area also shows a table of folders with columns for checkboxes, folder names, and timestamps. The 'Test' folder is listed with a timestamp of '3/11/2015 12:12:56 PM'.

Folder Name	Timestamp
First	8/26/2014 9:35:36 AM
Second	8/26/2014 9:35:36 AM
Test	3/11/2015 12:12:56 PM

5

The folder is created and listed as expected.

5.4.9 CDR-UC 9: Copy an object to a folder



The screenshot shows the CHIC web application interface. The browser address bar displays 'https://cdr-chic.ics.forth.gr/Home/Browse'. The page title is 'Browse/Results - CHIC Clin...'. The main content area is titled 'Repository » Browse/Results'. A search bar at the top shows 'Search results for '@Brain': 39'. Below the search bar, a list of search results is displayed. The first result, 'CDR.Head.012Y.O.CT.296.000.dcm', is highlighted with a blue circle and the number 1. A context menu is open over this result, with the 'Copy' option highlighted with a blue circle and the number 2.

Object Name	Timestamp
CDR.Head.012Y.O.CT.296.000.dcm	2/17/2015 11:51:15 AM
CDR.Brain.XX.O.MR.59.000.mha	8/28/2014 6:18
CDR.Brain.XX.O.MR.58.000.mha	8/28/2014 6:18
CDR.Brain.XX.O.MR.57.000.mha	8/28/2014 6:18

0

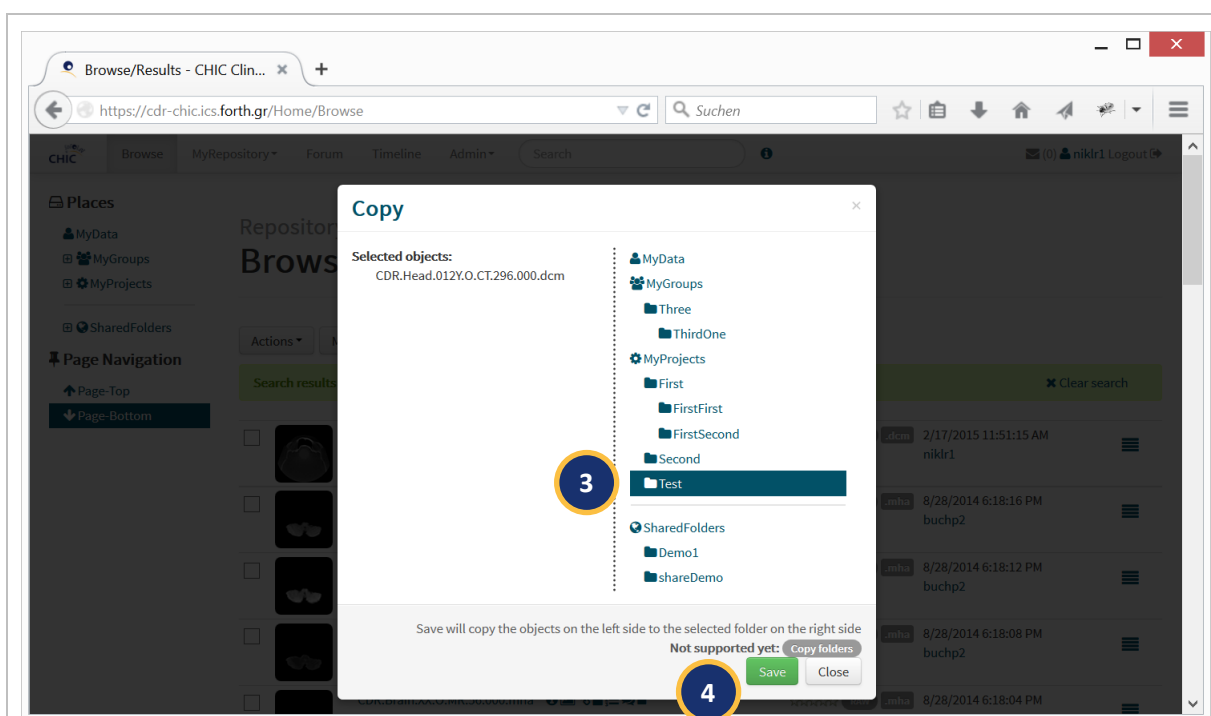
Select the object you want to copy to a folder.

1

Hit the icon on the right side to open the options menu.

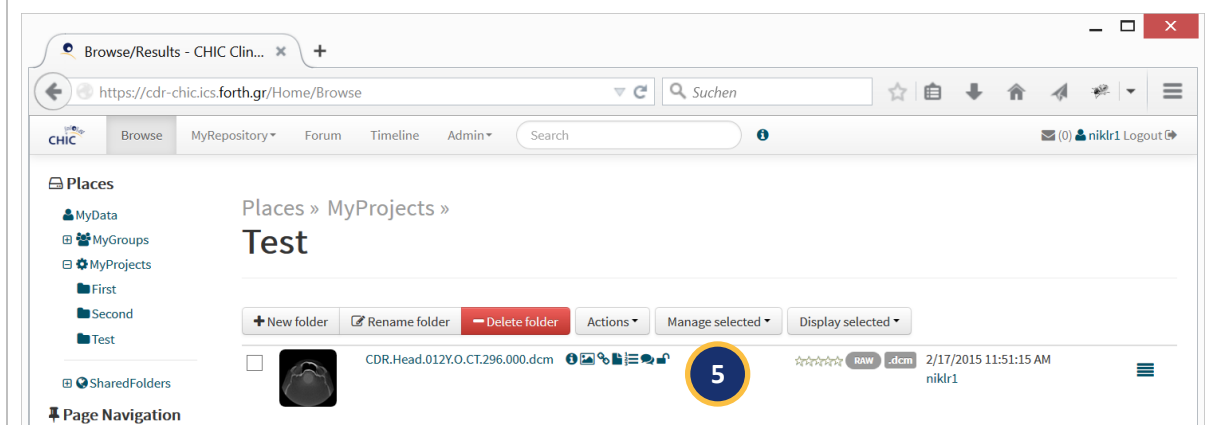
2

Choose the "Copy" option.



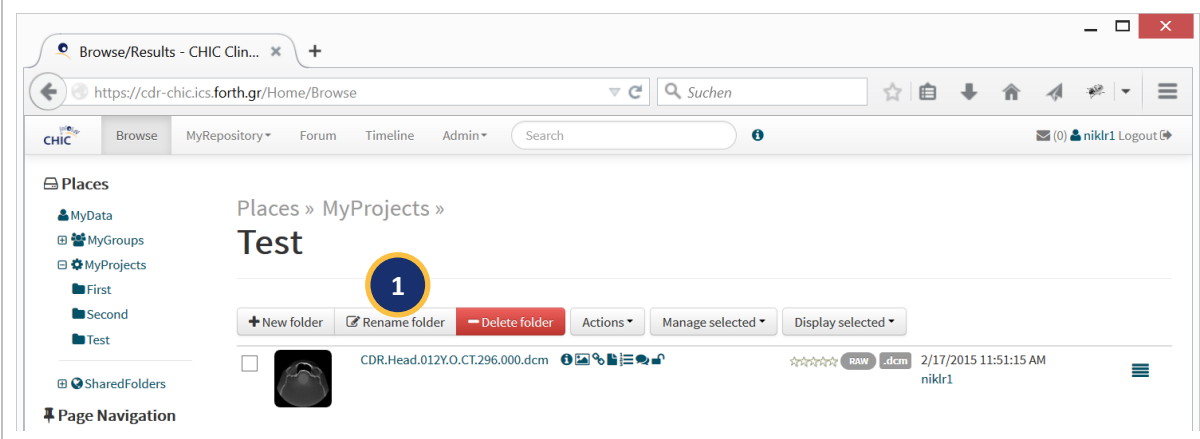
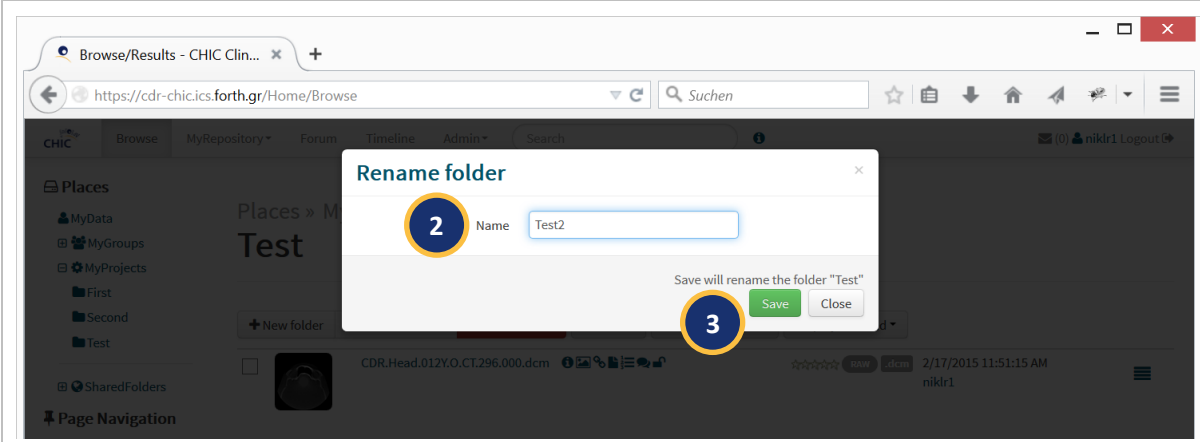
3 Select the target folder where the object should be copied to.

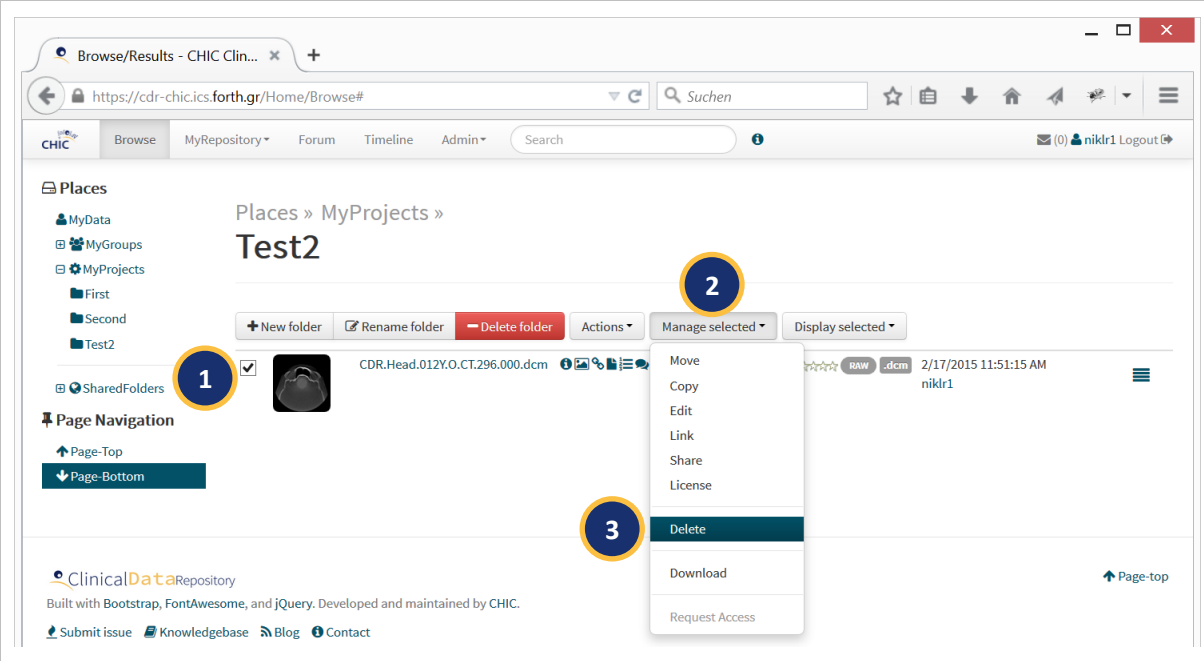
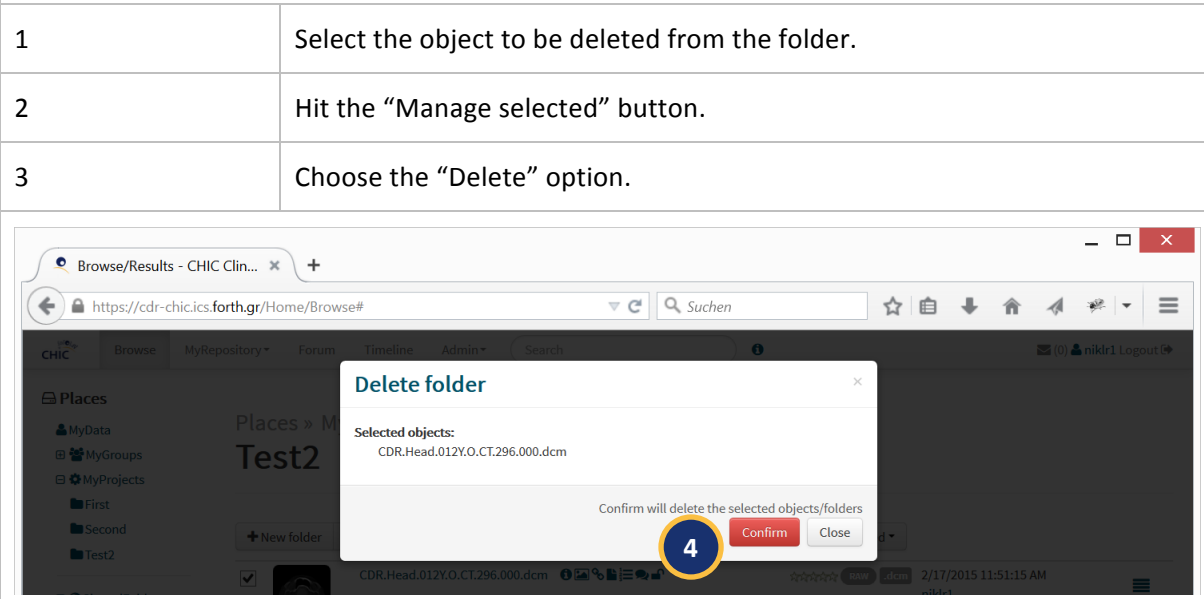
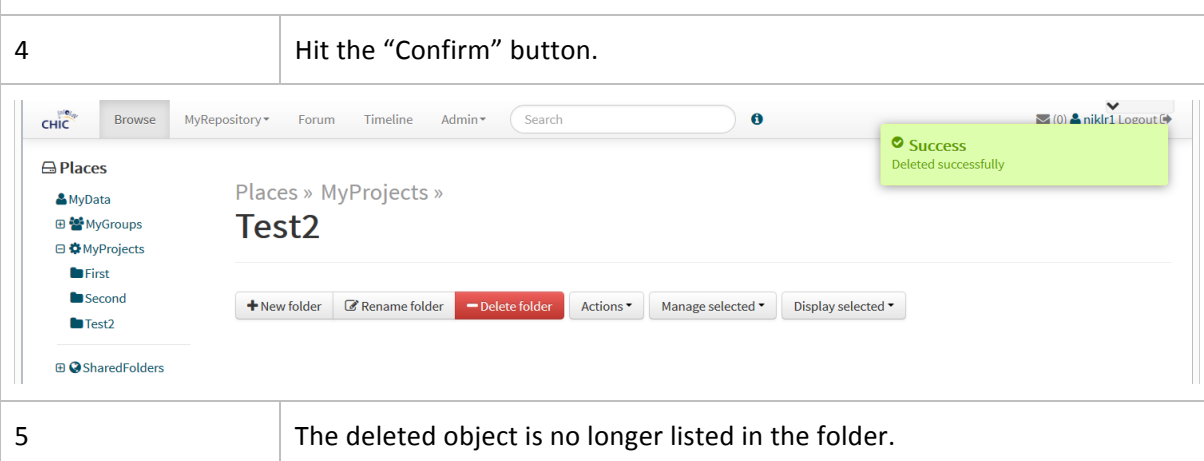
4 Hit the “Save” button.

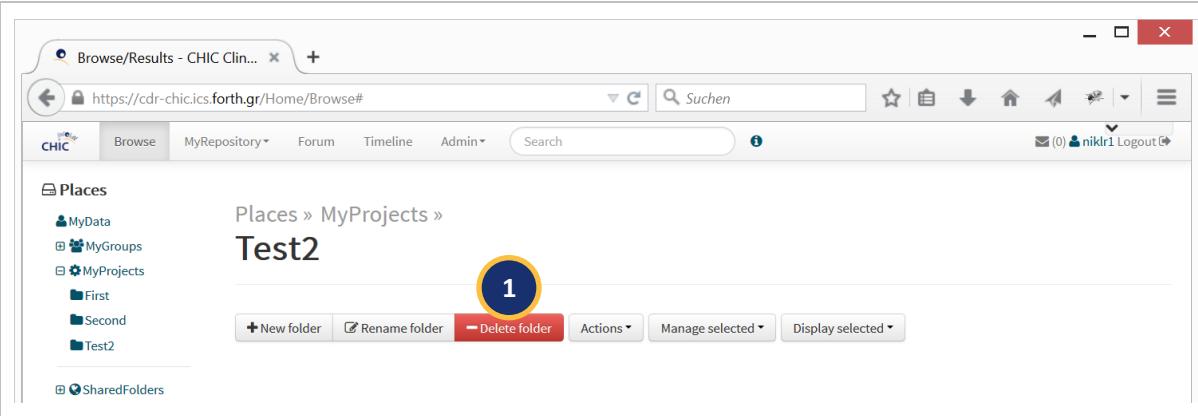
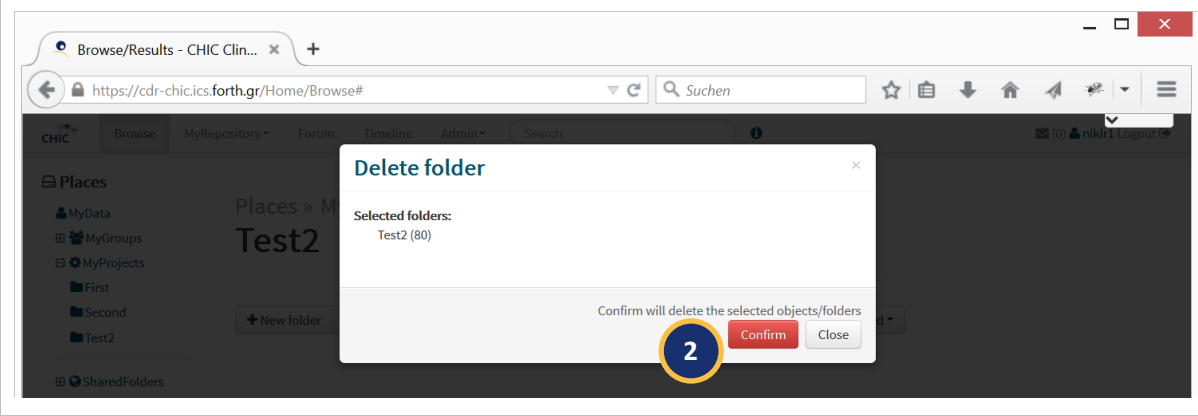
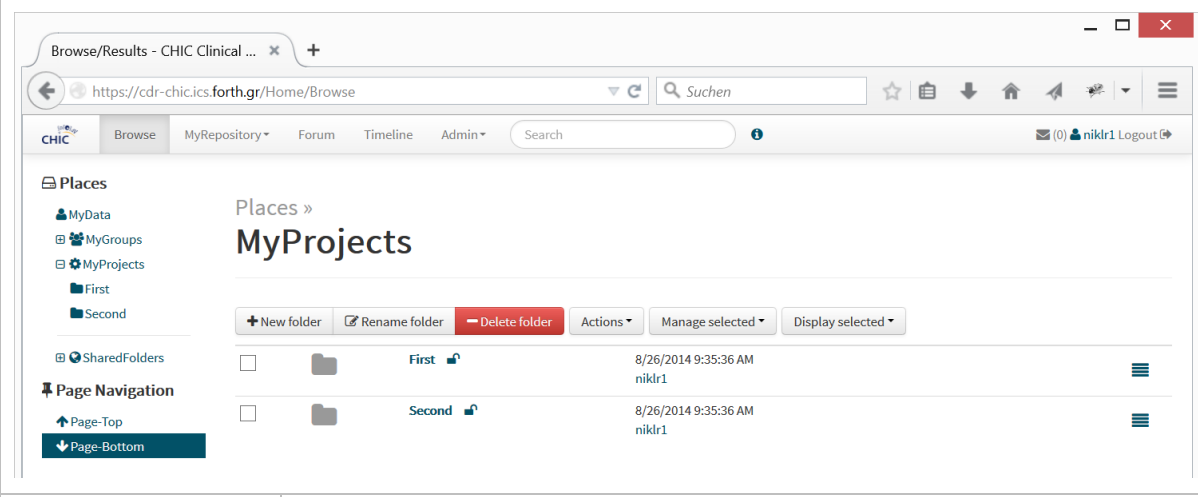


5 The object is listed in the folder.

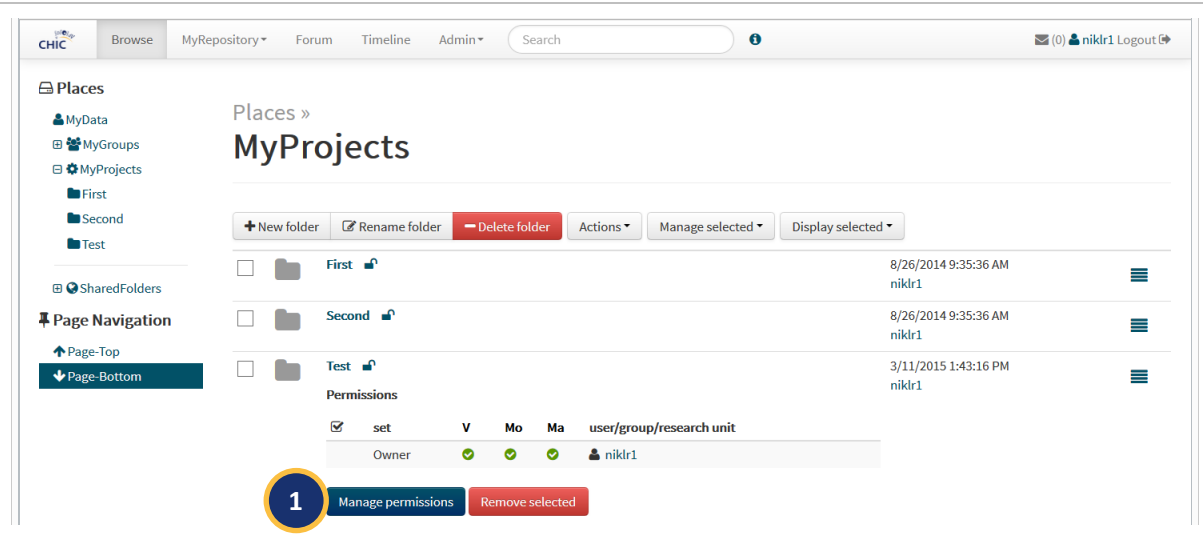
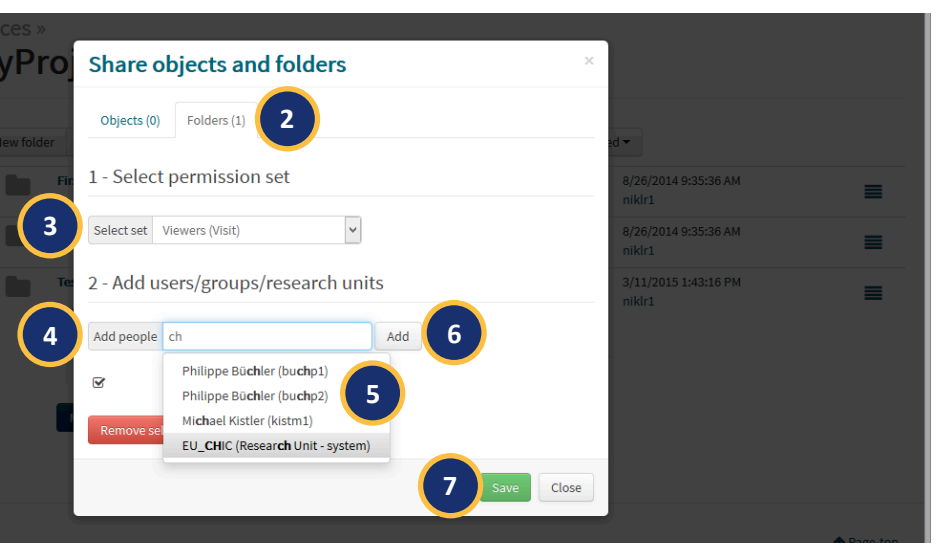
5.4.10CDR-UC 10: Modify a folder

	
0	Select the folder to be renamed.
1	Hit the “Rename folder” button.
	
2	Enter a new name for the folder.
3	Hit the “Save” button.

	<p>1 Select the object to be deleted from the folder.</p> <p>2 Hit the “Manage selected” button.</p> <p>3 Choose the “Delete” option.</p>
	<p>4 Hit the “Confirm” button.</p>
	<p>5 The deleted object is no longer listed in the folder.</p>

	
0	Select the folder to be deleted.
1	Hit the “Delete folder” button.
	
2	Hit the “Confirm” button.
	
3	The deleted folder is no longer listed.

5.4.11 CDR-UC 11: Share a folder with others

	
0	Select the folder to be shared.
1	Hit the “Manage permissions” button.
	
2	Select the “Folders” tab.
3	Select the appropriate permission set.
4	Start by typing to search for a user, group or research unit.
5	Select the appropriate user, group or research unit.
6	Hit the “Add” button.
7	Hit the “Save” button.

The screenshot shows the CHIC web interface. On the left, a sidebar contains 'Places' (MyData, MyGroups, MyProjects, First, Second, Test, SharedFolders) and 'Page Navigation' (Page-Top, Page-Bottom). The main area is titled 'Places » MyProjects'. It features buttons for '+ New folder', 'Rename folder', 'Delete folder', 'Actions', 'Manage selected', and 'Display selected'. Below these are three folders: 'First', 'Second', and 'Test'. The 'Test' folder is selected, and its permissions are shown in a table below. A yellow circle with the number '8' highlights the permissions table, and a yellow circle with the number '9' highlights the 'Test' folder in the list.

	set	V	Mo	Ma	user/group/research unit
<input checked="" type="checkbox"/> Owner		✓	✓	✓	niklr1
<input type="checkbox"/> Viewers		✓	✗	✗	EU_CHIC ✗

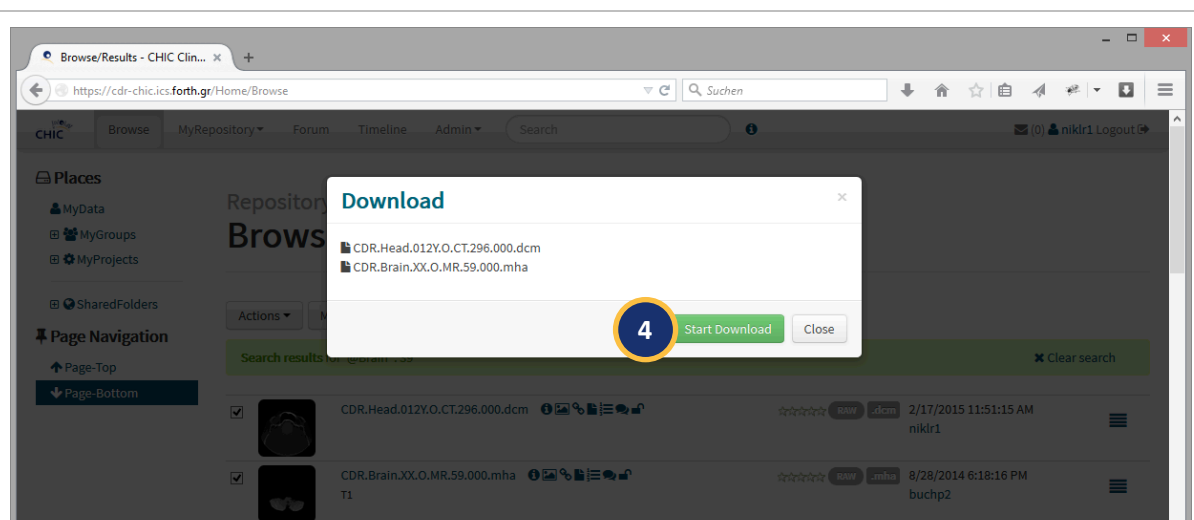
Buttons at the bottom of the permissions section are 'Manage permissions' and 'Remove selected'.

8	The saved sharing & permissions are listed in the permissions section.
9	The folder appears in the shared folders for authorized users, groups or research units.

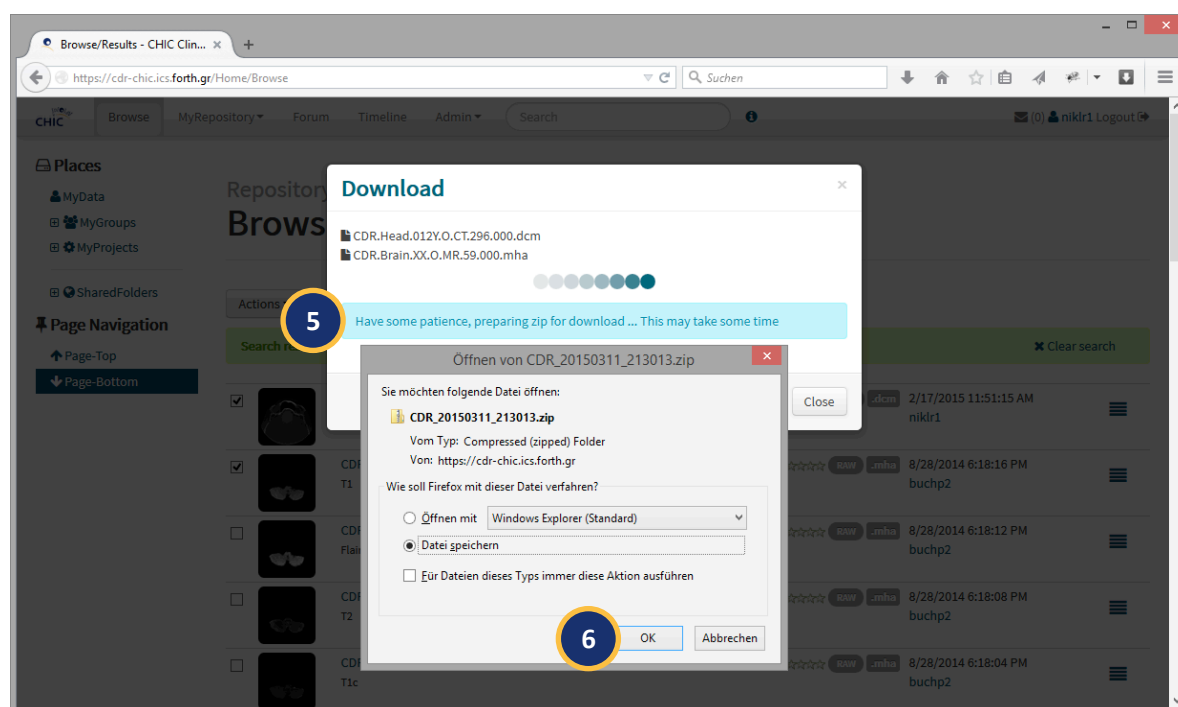
5.4.12CDR-UC 12: Download objects or folders

The screenshot shows the CHIC web interface at the 'Browse/Results' page. The sidebar is similar to the previous screenshot. The main area is titled 'Repository » Browse/Results'. It features buttons for 'Actions', 'Manage selected', and 'Display selected'. A search bar is present. Below the search bar is a list of objects. A yellow circle with the number '1' highlights the first object in the list. A yellow circle with the number '2' highlights the 'Manage selected' button. A yellow circle with the number '3' highlights the 'Download' option in the context menu that appears when the first object is selected.

1	Select the objects or folders to be downloaded.
2	Hit the “Manage selected” button.
3	Choose the “Download option.”



4 Hit the “Start download” button.



5 Wait until the compressed zip file is prepared.

6 Hit the “Ok” button in order to download the selected objects or folders to your local hard disk.

5.5 Deployment

The prototype of the clinical data repository is deployed to the CHIC private cloud infrastructure which makes use of the OpenStack open source cloud computing software as described in deliverable “D5.1.1 - The CHIC technical architecture”. In this chapter the specifications of the used virtual machine are described and what is needed to install, configure and run the prototype.

5.5.1 Infrastructure

The private cloud infrastructure provided by CHIC allows great flexibility in terms of computational power, storage, and networking resources. Figure 5 illustrates how the clinical data repository has been deployed (the second physical location is not functional at this prototype level). The objective of having a second physical location with mirrored resources is to strive for a high availability system with minimal downtime and no data loss. For automatic failover of the mirrored databases a witness server instance would be needed. The database owner can turn off the witness for a database at any time. Turning off the witness is equivalent to having no witness, and automatic failover cannot occur.

Virtual Machine 1 specifications:

- Operating system: Microsoft Windows Server 2012 Standard
- Computational power: 40 GB RAM, 4 x 2.9 GHz, 64-bit

Virtual Storage 1 specifications:

- Storage: 1TB++ (depending on the number of patients)

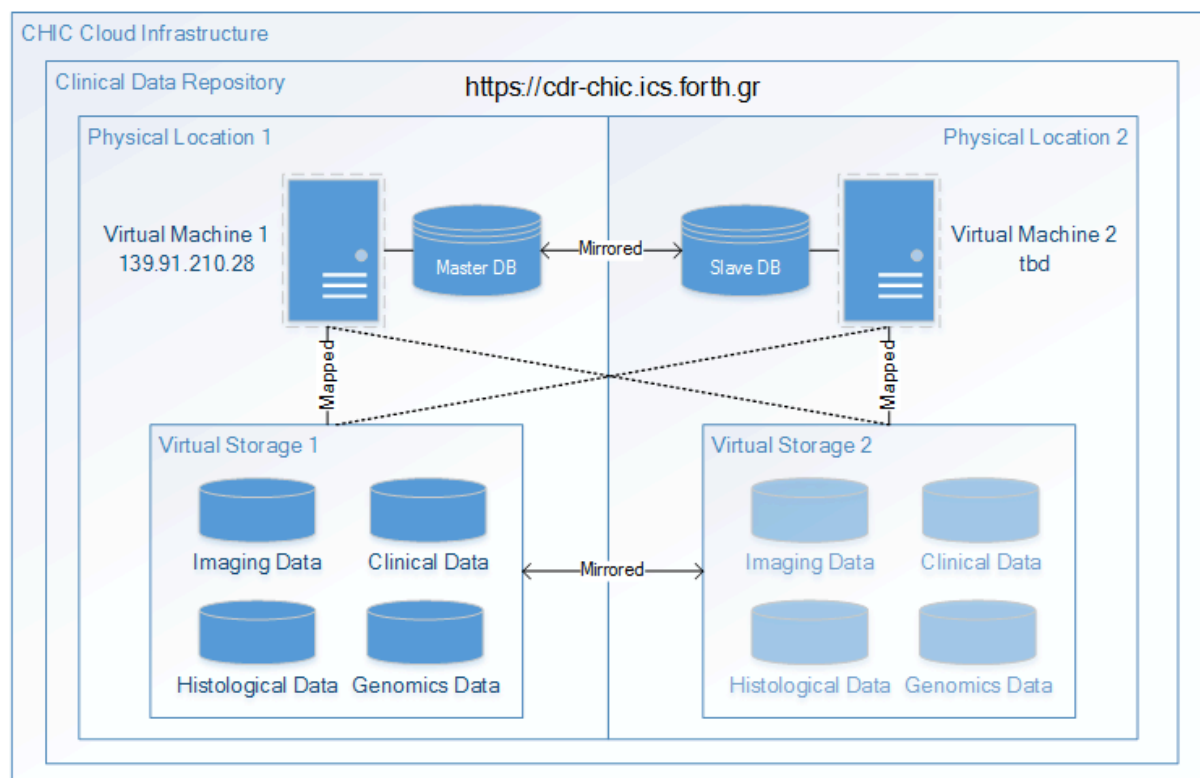


Figure 5: The clinical data repository in the CHIC cloud infrastructure

5.5.2 Installation

This chapter describes how to install and configure the prototype of the clinical data repository in the CHIC private cloud environment. The installation and configuration instructions assume a clean install of the operating system and administrative user rights. All necessary steps are summarized below:

1. Install Microsoft Internet Information Services (IIS)
 - a. Install Microsoft .NET Framework
2. Install Shibboleth Service Provider
3. Install Microsoft SQL Server
4. Configure Microsoft Internet Information Services (IIS)
5. Configure Shibboleth Service Provider
6. Configure Microsoft SQL Server
7. Publishing/deploying the Clinical Data Repository

5.5.2.1 Install Microsoft Internet Information Services (IIS) and Microsoft .NET Framework

First of all, the extensible web server called Internet Information Services (IIS) created by Microsoft needs to be installed. It is included with all versions of Microsoft's operating systems and can be installed through the graphical user interface (GUI) by using the Server Manager interface.

Server Manager provides a single dashboard to install or uninstall server roles and features. Server Manager also gives an overview of all currently installed roles and features. When IIS is chosen from the Server Manager, the basic components and services needed for IIS are automatically selected.

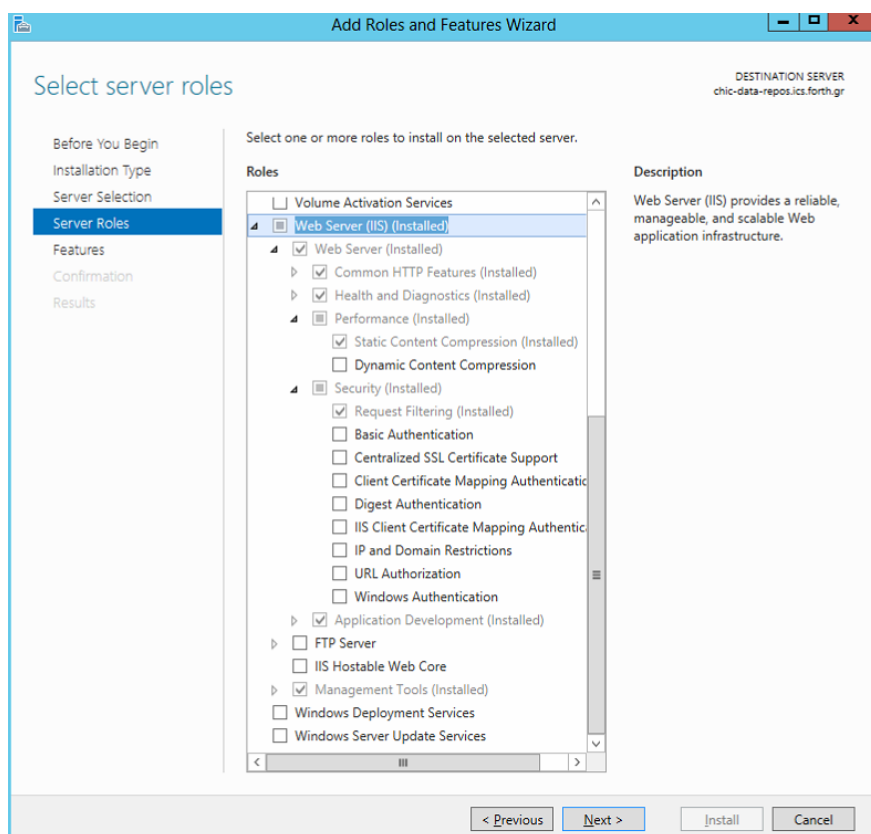


Figure 6: Adding the Web Server (IIS) role in the Server Manager on the Clinical Data Repository

The Microsoft .NET Framework can be installed right after selecting the Web Server (IIS) role. It includes a large class library known as Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for .NET Framework execute in a software environment (as contrasted to hardware environment), known as Common Language Runtime (CLR), an application virtual machine that provides services such as security, memory management, and exception handling. FCL and CLR together constitute .NET Framework.

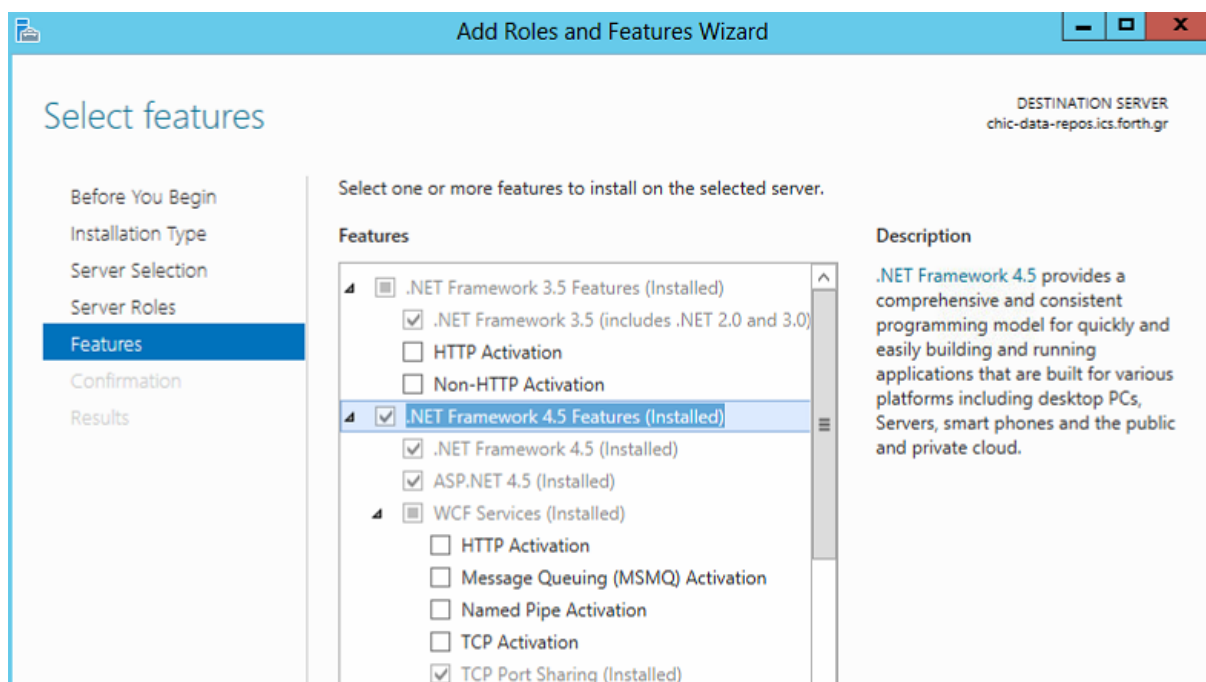


Figure 7: Adding the .NET Framework feature in the Server Manager on the Clinical Data Repository

5.5.2.2 Install Shibboleth Service Provider

The Shibboleth Service Provider needs to be installed in order to enable the brokered authentication flow described in chapter 5.2. Shibboleth is a web-based technology that implements the HTTP/POST, artifact, and attribute push profiles of SAML, including both Identity Provider (IdP) and Service Provider (SP) components. For the clinical data repository the SP component is sufficient.

1. Download the appropriate 32- or 64-bit .msi Shibboleth SP installer from the Shibboleth download site (<http://www.shibboleth.net/downloads/service-provider/latest>).
2. Run the installer. The installer will prompt for an install path, change default configuration files as appropriate for Windows, and set various environment variables. The shibd service will be installed automatically.

After rebooting, IIS should be configured for basic support. For an instant test, if the Shibboleth daemon and ISAPI Filter are working access with your browser the following URL.

<https://localhost/Shibboleth.sso/Metadata>

This URL should return the dynamically generated XML metadata of this Service Provider.

Source 1: <https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPWindowsIIS7Installer>

Source 2: <https://www.switch.ch/aai/docs/shibboleth/SWITCH/2.4/sp/deployment/windows-iis.html>

5.5.2.3 Install Microsoft SQL Server

Microsoft SQL Server is a relational database management system developed by Microsoft. As a database, it is a software product whose primary function is to store and retrieve data as requested by other software applications, be it those on the same computer or those running on another computer across a network.

This topic provides a step-by-step procedure for installing a new instance of SQL Server using the SQL Server setup installation wizard. The SQL Server Installation Wizard provides a single feature tree for installation of all SQL Server components so that they do not have to be installed individually.

1. Insert the SQL Server installation media. From the root folder, double-click Setup.exe.
2. The Installation Wizard runs the SQL Server Installation Center. To create a new installation of SQL Server, click Installation in the left-hand navigation area, and then click New SQL Server stand-alone installation or add features to an existing installation.
3. On the Setup Role page, select SQL Server Feature Installation.
4. On the Feature Selection page, select the components as illustrated in Figure 8.

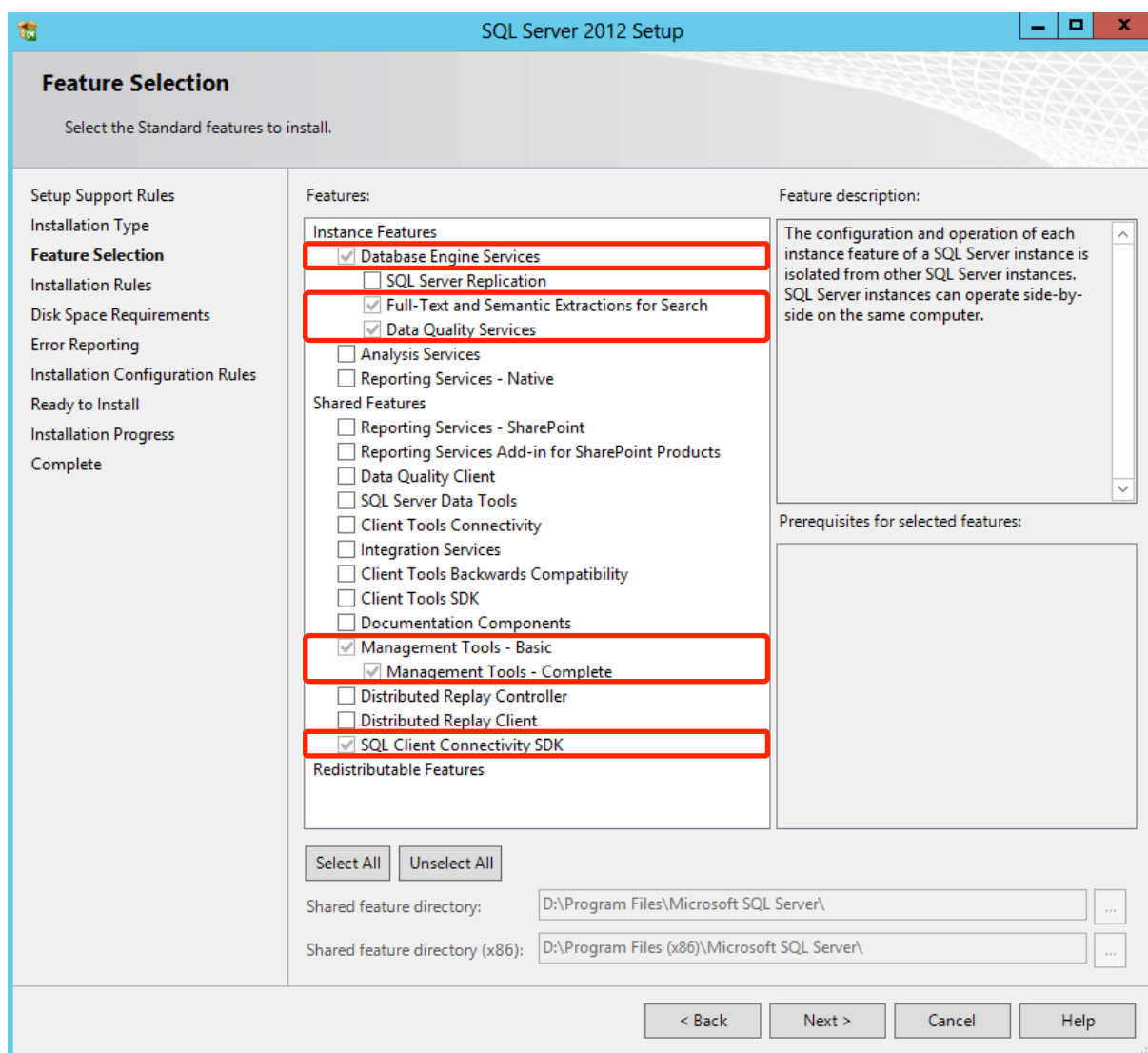


Figure 8: Microsoft SQL Server feature selection on the Clinical Data Repository

5.5.2.4 Configure Microsoft Internet Information Services (IIS)

Before configuring the Microsoft Internet Information Services a local user has to be created in the Computer Management as shown in Figure 9. The name of the user is not relevant. However, the user should be created with the following settings:

- User cannot change password
- Password never expires
- Member of Users group

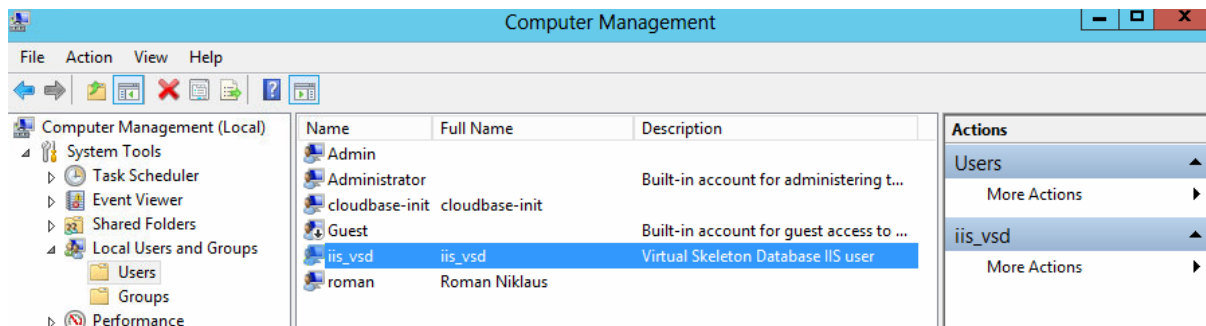


Figure 9: Creating a local user to be used by the Internet Information Services (IIS) in the Computer Management of the Clinical Data Repository

Next, an application pool has to be created in the Internet Information Services (IIS) Manager as shown in Figure 10. The previously created user is used as identity to run the application pool. The name of the application pool is not relevant. However, the application pool should be created with the following settings:

- .NET Framework Version: v4.0
- Enable 32-Bit Applications: False
- Managed Pipeline Mode: Integrated

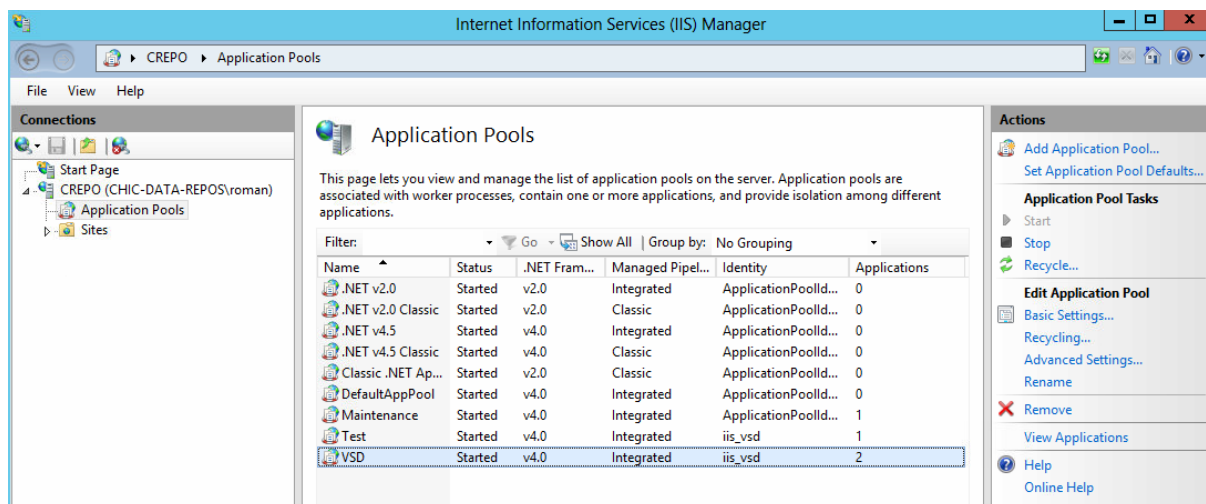


Figure 10: Creating an application pool in the Internet Information Services (IIS) Manager on the Clinical Data Repository

Once the application pool exists, the website and web service can be created. As illustrated in Figure 11 the website is represented by the Default Web Site entry and the web service is represented by the api entry. Both entries point to different physical locations but use the same application pool.

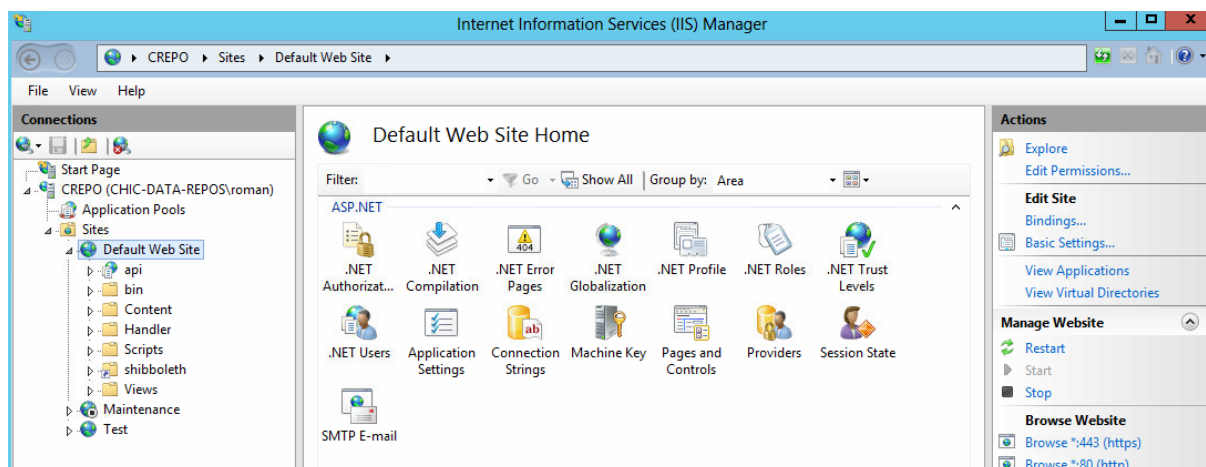


Figure 11: Creating the website and web service in the Internet Information Services (IIS) on the Clinical Data Repository

5.5.2.5 Configure Shibboleth Service Provider

A certificate is needed to encrypt the communication between the Service Provider (SP) and the Identity Provider (IdP). For this purpose a self-signed certificate has to be created or a certificate from a well-known Certificate Authority (CA) is available to be used. The main configuration of the Shibboleth Service Provider is done in C:\opt\shibboleth-sp\etc\shibboleth\shibboleth2.xml. Please verify that the site id within InProcess corresponds to the instance ID of the IIS configuration.

```
<!--
The InProcess section contains settings affecting web server modules.
Required for IIS, but can be removed when using other web servers.
-->
<InProcess logger="native.logger">
  <ISAPI normalizeRequest="true" safeHeaderNames="true">
    <!--
    Maps IIS Instance ID values to the host scheme/name/port. The name is
    required so that the proper <Host> in the request map above is found without
    having to cover every possible DNS/IP combination the user might enter.
    -->
    <Site id="1" name="139.91.210.28" scheme="https" port="443"/>
    <!--
    When the port and scheme are omitted, the HTTP request's port and scheme are used.
    If these are wrong because of virtualization, they can be explicitly set here to
    ensure proper redirect generation.
    -->
    <!--
    <Site id="42" name="virtual.example.org" scheme="https" port="443"/>
    -->
  </ISAPI>
</InProcess>
```

Figure 12: Shibboleth InProcess configuration on the Clinical Data Repository

In the RequestMapper section of the shibboleth2.xml file, the behaviour for specific resources on IIS can be customized. As illustrated in Figure 13 the following URLs will redirect the user to the IdP.

- <https://139.91.210.28/Account/LogOnShibboleth>
- <https://139.91.210.28/Account/LogOnShibbolethCallback>

```
<RequestMapper type="Native">
  <RequestMap>
    <!--
      The example requires a session for documents in /secure on the containing host with http and
      https on the default ports. Note that the name and port in the <Host> elements MUST match
      Apache's ServerName and Port directives or the IIS Site name in the <ISAPI> element above.
    -->
    <Host name="139.91.210.28">
      <Path name="Account">
        <Path name="LogOnShibboleth" authType="shibboleth" requireSession="true"/>
        <Path name="LogOnShibbolethCallback" authType="shibboleth" requireSession="true"/>
      </Path>
    </Host>
  </RequestMap>
</RequestMapper>
```

Figure 13: Shibboleth RequestMapper configuration on the Clinical Data Repository

Last but not least, the default Identity Provider (IdP) and the MetadataProvider are configured in the ApplicationDefaults section. To allow for >1 IdP, remove the entityID property and adjust discoveryURL to point to a discovery service. (Set discoveryProtocol to "WAYF" for legacy Shibboleth WAYF support.) The file defined in the MetadataProvider element must be provided by the IdP and in this case stored in the same location as the shibboleth2.xml file.

```
<!--
The ApplicationDefaults element is where most of Shibboleth's SAML bits are defined.
Resource requests are mapped by the RequestMapper to an applicationId that
points into to this section (or to the defaults here).
-->
<ApplicationDefaults entityID="https://139.91.210.28/shibboleth"
  REMOTE_USER="eppn persistent-id targeted-id">

  <Sessions lifetime="28800" timeout="3600" relayState="ss:mem"
    checkAddress="false" handlerSSL="true" cookieProps="https">

    <SSO entityID="https://ciam-dev-chic.custodix.com/idp/shibboleth">
      SAML2 SAML1
    </SSO>

    <!-- SAML and local-only logout. -->
    <Logout>SAML2 Local</Logout>

    <!-- Extension service that generates "approximate" metadata based on SP configuration. -->
    <Handler type="MetadataGenerator" Location="/Metadata" signing="false"/>

    <!-- Status reporting service. -->
    <Handler type="Status" Location="/Status" acl="127.0.0.1 ::1"/>

    <!-- Session diagnostic service. -->
    <Handler type="Session" Location="/Session" showAttributeValues="false"/>

    <!-- JSON feed of discovery information. -->
    <Handler type="DiscoveryFeed" Location="/DiscoFeed"/>
  </Sessions>

  <MetadataProvider type="XML" file="ciam-dev-chic-metadata.xml"/>
</ApplicationDefaults>
```

Figure 14: Shibboleth Identity Provider (IdP) configuration on the Clinical Data Repository

Another important configuration file is attribute-map.xml. By default, the SP ignores SAML information about users provided by the IdP unless it is defined specifically how and what to process. A variety of example mappings are supplied by default, but most of them are commented out to save processing time. For efficiency, it's suggested to enable or define mappings for only the information the application care about.

```
<Attribute name="urn:custodix:ciam:1.0:principal:uuid" id="uuid"/>
<Attribute name="urn:custodix:ciam:1.0:principal:uuid:alias" id="alternativeUuid"/>
<Attribute name="urn:oid:0.9.2342.19200300.100.1.1" id="userid"/>
<Attribute name="urn:custodix:ciam:1.0:domain:uuid" id="domainUuid"/>
<Attribute name="urn:custodix:ciam:1.0:domain:name" id="domainName"/>
<Attribute name="urn:oid:2.5.4.4" id="sn"/>
<Attribute name="urn:oid:2.5.4.42" id="givenName"/>
<Attribute name="urn:oid:0.9.2342.19200300.100.1.3" id="mail"/>
```

Figure 15: Shibboleth Attribute Map configuration on the Clinical Data Repository

The vocabulary used in the Shibboleth Attribute Map is defined by the CHIC security framework and listed in Table 6. This list is a temporary snapshot and subject to change.

Table 7: The CHIC security framework vocabulary used in the Shibboleth Attribute Map

Uniform Resource Name (URN)	Description
urn:custodix:ciam:1.0:principal:uuid	The federated ID of the user
urn:custodix:ciam:1.0:principal:uuid:alias	The alternative federated ID's of the user (after account merging)
urn:oid:0.9.2342.19200300.100.1.1	Contains a human readable identifier: the username of the user.
urn:custodix:ciam:1.0:domain:uuid	The security domain uuid this user is member of.
urn:custodix:ciam:1.0:domain:name	The security domains name this user is member of.
urn:custodix:ciam:1.0:principal:email urn:oid:0.9.2342.19200300.100.1.3	Registered email address
urn:oid:2.5.4.12	Title of the user
urn:oid:2.5.4.42	First name of the user
urn:oid:2.5.4.4	Last name of the user
urn:oid:2.5.4.10	Name of the organisation the user is part of

urn:oid:2.5.6.8

CHIC role

5.5.2.6 Configure Microsoft SQL Server

The database can be created in the Microsoft SQL Server Management Studio. The name of the database is not relevant as long as the same name is used in the configuration file of the web site and in the SQL scripts listed consecutively. Once the database is created, a login for the previously created user can be added. This login should be part of the public server role and mapped to the previously created database with db_datareader, db_datawriter and public role memberships as illustrated in Figure 16.

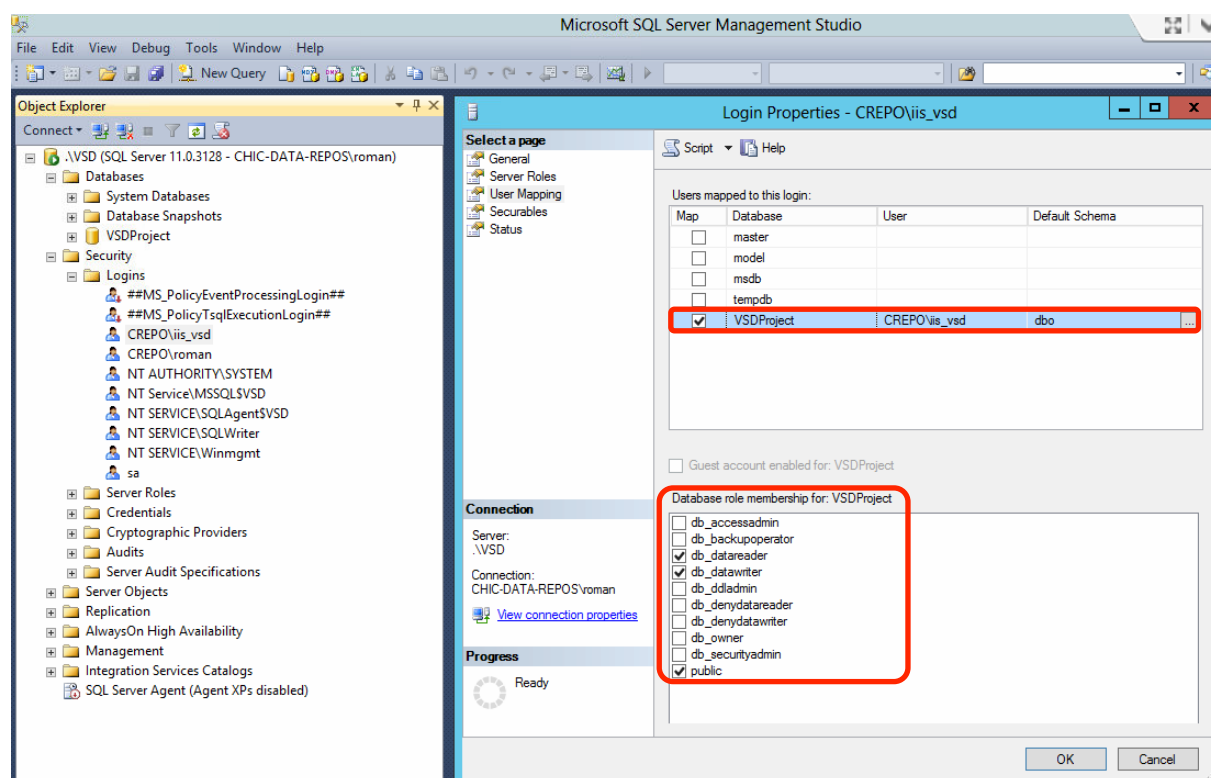


Figure 16: Creating the database in the Microsoft SQL Server Management Studio on the Clinical Data Repository

Next, several SQL scripts have to be executed in a predefined order to initialize the database. They will be provided in combination with the source code of the Clinical Data Repository.

Before creating a Maintenance Plan for backup purposes, the SQL Server Agent needs to be started. By default the SQL Server Agent needs to be started manually. This behaviour must be changed so that the service will be started after booting the operating system as illustrated in Figure 17.

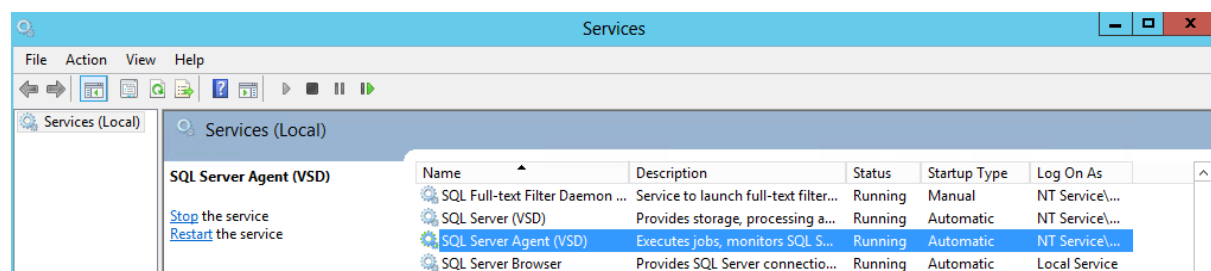


Figure 17: Starting the SQL Server Agent automatically after booting the Clinical Data Repository

In the Microsoft SQL Server Management Studio the Maintenance Plan can be created with the following sub plans. Each sub plan is scheduled differently but runs with the same SQL Server Agent service account as illustrated in Figure 18.

A full backup of the database is taken every week on Sunday at 12:00 AM. This sub plan is complemented by a differential backup occurring every day at 12:00 AM and a transaction log backup occurring every day every 1 hour. Furthermore, to reduce used disk space the differential and transaction log backups will be deleted every week on Sunday at 12:00 AM if older than 4 weeks. With this maintenance plan it is possible to restore the database at any given point in the past 4 weeks. If the given point in time exceeds the 4 weeks window, the database can be restored to a weekly restore point using a full backup.

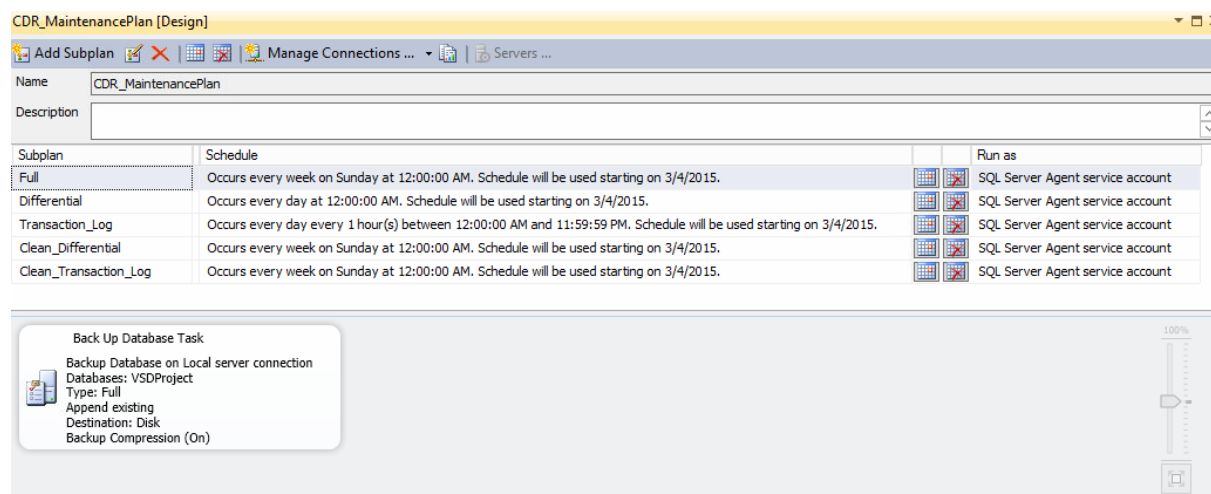


Figure 18: Creating a Maintenance Plan in the Microsoft SQL Server Management Studio for backup purposes on the Clinical Data Repository

5.5.2.7 Deploying the Clinical Data Repository

The deploying procedure described in this chapter makes use of the publish function provided by Visual Studio 2013. For this purpose a publishing profile can be created for the website and for the web service. Using the publishing profile illustrated in Figure 19 the website and web service will be published to the file system in the defined target location. From there, the folder can be copied to the virtual machine.

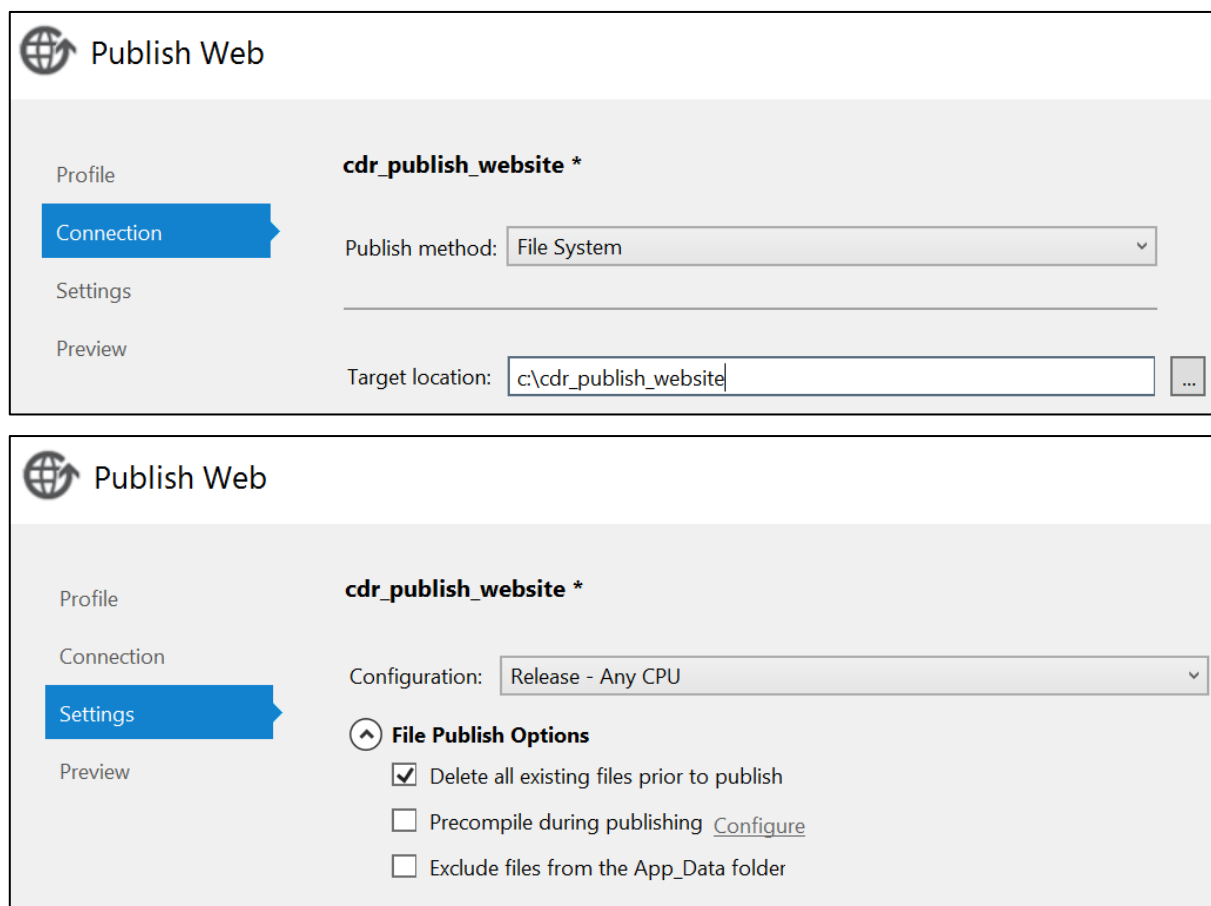


Figure 19: Creating a publishing profile in Visual Studio 2013 for the Clinical Data Repository

Once copied to the appropriate folder as configured in the Internet Information Services (IIS) on the virtual machine, several one-time modifications are necessary such as customizing the Web.config of the website and web service, adding a system path variable and enabling the support for symbolic links on the file system.

Web.config is the main file containing settings and configuration for an ASP.NET web application. The file is based on XML and stores the information about how the web application will act. It contains information that controls module loading, security configuration, session state configuration, and application language and compilation settings. Furthermore, it can also contain application specific items such as database connection strings.

First, the folder configuration entries should be customized to map the actual file system. In most cases it is sufficient to change the drive letter. Then, the connection string pointing to the SQL database should be adapted as illustrated in Figure 20.

```
<VSDSettingsGroup>
  <VSDSettings>
    <folders>
      <add key="cache" path="D:\www\VSD\Data\Cache" />
      <add key="data" path="D:\www\VSD\Data\Binary" />
      <add key="images" path="D:\www\VSD\Data\Images" />
      <add key="download" path="D:\www\VSD\Data\Download" />
      <add key="template" path="D:\www\VSD\Data\Template" />
      <add key="temp" path="D:\www\VSD\Data\Temp\" />
      <add key="challenges" path="D:\www\VSD\Data\Challenges\" />
      <add key="apps" path="D:\www\VSD\Data\Apps\" />
      <add key="TempUploadFolder" path="D:\www\VSD\Data\TempUpload" />
    </folders>
  </VSDSettings>
</VSDSettingsGroup>
<connectionStrings>
  <add name="VSDContext" connectionString="..." providerName="System.Data.EntityClient" />
</connectionStrings>
```

Figure 20: Customizing the Web.config on the Clinical Data Repository

The Clinical Data Repository makes use of the external HDF5DotNet - C++/CLI Wrapper of the HDF5 Library to support the Hierarchical Data Format. The installation instructions of this library require to add a reference to HDF5DotNet.dll to the Visual Studio project. For running the application, in addition to HDF5DotNet.dll, several native Dynamic Link Libraries (DLLs) are required. These native DLLs must be present in a directory that is part of the PATH environment variable on the virtual machine.

Last but not least, the Clinical Data Repository makes use of so called symbolic links (also symlink or soft link). A symbolic link is a special type of file that contains a reference to another file or directory in the form of an absolute or relative path and that affects pathname resolution. For this purpose the user created in chapter 5.5.2.4 must be added in the Local Security Policy to create symbolic links as illustrated in Figure 21.

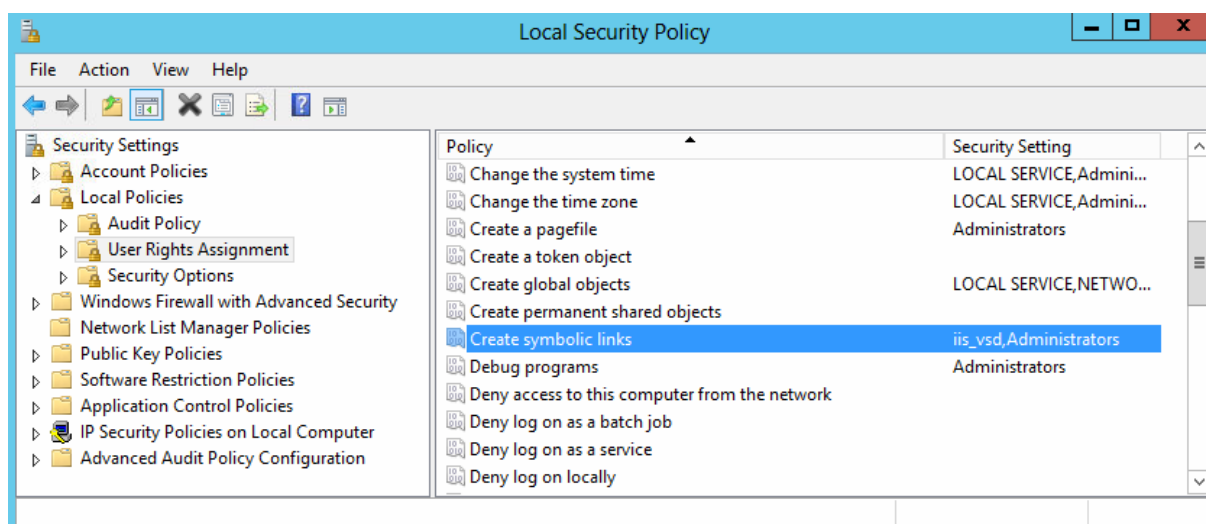


Figure 21: Setting the Local Security Policy to allow the creation of symbolic links on the Clinical Data Repository

5.6 Conclusion

The design of the clinical data repository is based on the concept of a generic object to easily accommodate different types of data. In its prototype implementation, the system is able to store medical images and clinical study data. However, due to this flexible design, the adaptation of the system for new data types such as histological data and genetic data will be easily performed. On the other hand, a specific import system is implemented for each type of data, in order to extract the relevant information from the uploaded file and to allow its annotation with the appropriate metadata.

First data exchange data between clinicians and researchers could already be successfully conducted due to adoption of the EU-CHIC security framework at the very beginning of the prototype implementation. At this point the recommended brokered authentication mechanism introduced in deliverable “D5.2 - Security guidelines and initial version of security tools” has been fully integrated into the clinical data repository to support Single Sign-On.

In this deliverable we have presented possible use cases of the clinical data repository based on the general concepts introduced in the previous deliverable. Furthermore, the prototype implementation has been illustratively documented with a user guide as descriptive and simple as possible, so that it could also be provided to all end-users for training/instruction purposes. Last but not least, the clinical data repository has been successfully deployed to the private cloud infrastructure provided by FORTH allowing great flexibility in terms of computational power, storage space and networking resources. The deployed prototype implementation can be accessed by the following URL <https://cdr-chic.ics.forth.gr>

The programmatic interface of the clinical data repository will be reported in “D8.3 Implementation of the interfaces of the CHIC repositories”.

6 Model/tool and *in silico* trial repositories

6.1 Introduction

The model/tool repository is the technological platform where hypomodels, hypermodels, data transformation tools, linkers and other tools used in the construction and execution of hypermodels are stored. With the help of graphical user interface, authorized end users (researchers, modellers, clinicians etc.) may access the stored models/tools, store new ones, delete existing models/tools and update them. In addition, a web-based interface will expose the contents of the model/tool repository to other tools developed in the CHIC project, such as the hypermodelling editor.

The *in silico* trial repository is the technological platform where the data related to *in silico* trials and *in silico* experiments are stored. The *in silico* trial repository is going to be used for the persistent storage of simulation scenarios and *in silico* predictions. Similar to the model/tool repository, end users and other technological components may access the *in silico* trial repository in order to prepare new *in silico* trials or store the results of the conducted *in silico* experiments.

Popular, open source and cutting edge tools that encourage clean and pragmatic design have been used for the implementation of both repositories. More specifically, Apache http server has been chosen as an application server, Django has been chosen as a web application framework and MySQL has been chosen as a database server. The usage of the above tools is going to be described in the next chapters.

6.2 Model/tool and *in silico* trial repositories on the cloud

The Model/tool and *in silico* trial repositories are deployed to the CHIC private cloud infrastructure which makes use of the OpenStack open source cloud computing software. The Openstack cloud operating system offers on-demand computing resources, by provisioning and managing large networks of virtual machines. Computing resources are accessible via APIs for developers building cloud applications and via web services for administrators and users. Openstack is architected to provide flexibility as you design your cloud and the ability to integrate with legacy systems and third party technologies. It is designed to manage and automate pools of computing resources and can work with widely available virtualization technologies, as well as bare metal and high-performance computing (HPC) configurations [1]. More information about the OpenStack cloud computing software is provided in deliverable “D5.3 – Techniques to build the cloud infrastructure available to the community”.

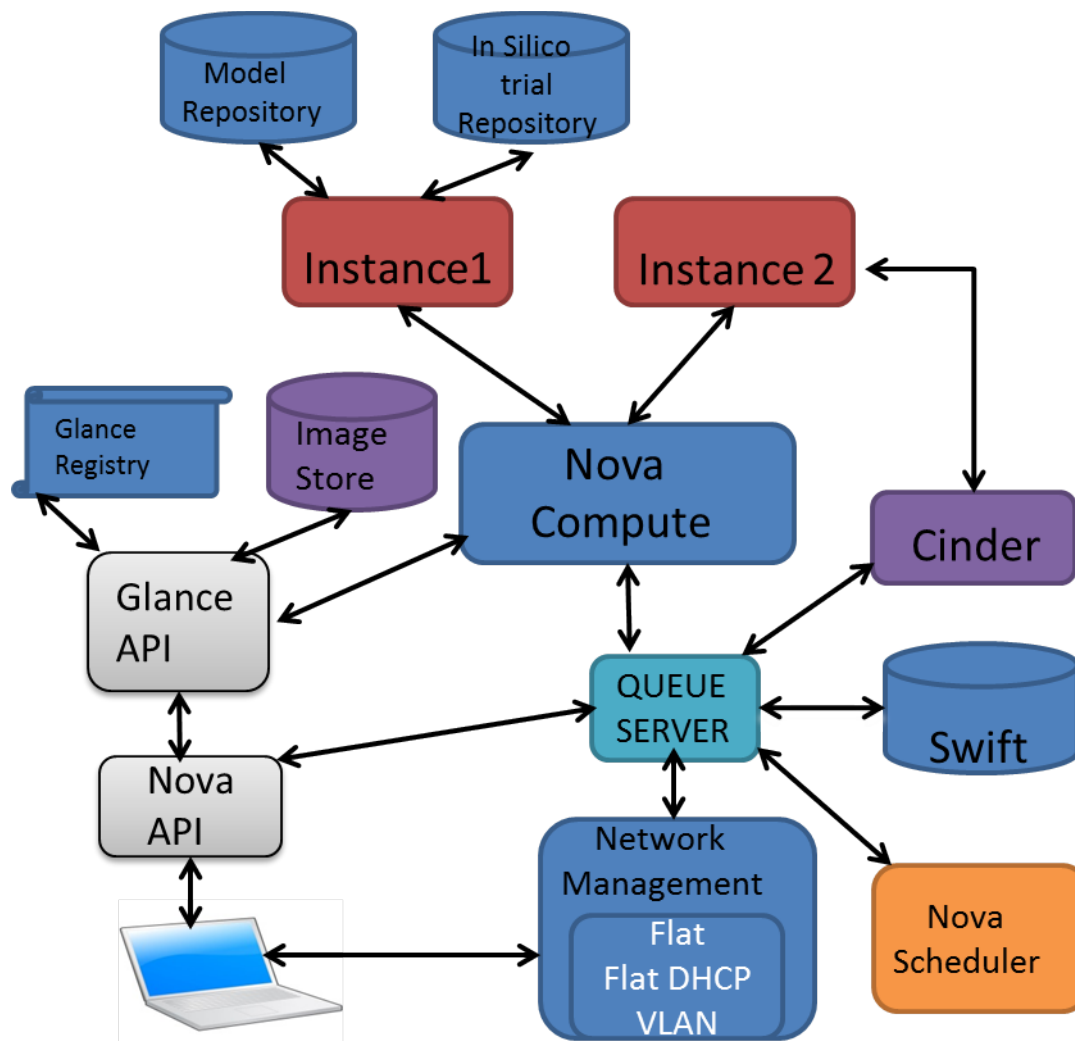
In the context of CHIC, a virtual machine has been provided for the deployment of the model/tool and *in silico* trial repositories, with the following specifications:

Virtual Machine Specifications:

- Operating system: Ubuntu 14.04.2 LTS
- CPU: Intel Xeon E312xx (Sandy Bridge) 2 GHz (size and capacity), 64 bits width
- DIMM RAM: 8GB, 64 bits

- Disk 107,4 GB, 209715200 sectors, 1638400 cylinders, sector size 512 Bytes

Figure 22 shows the components of the Openstack architecture and depicts how the virtual machine, in which the model/tool and *in silico* trial repositories are deployed, uses resources from the CHIC private cloud. Table 7 provides a brief description of the components of Openstack architecture [1].



3

Figure 22: Model/tool and *in silico* trial repositories deployed to the CHIC private cloud. For a brief description of the components see TABLE 7

Table 8: Brief description of the components of Openstack Architecture

Components of Openstack architecture	
Component	Brief Description
Network Management	Ensures instances have the right kind of networking setup.
Nova Scheduler	Dispatches compute and volume requests.
Swift	Provides object storage infrastructure
Queue Server	Sits between any 2 nova components and allows them to communicate in a loosely coupled fashion by using remote procedure calls.
Cinder	Block storage service
Nova Compute	Service responsible for handling and providing resources to virtual machines
Instance	Virtual machine that runs inside the cloud. Model/tool and <i>in silico</i> trial repositories have been deployed to a virtual machine
Nova API	Can be used for launching server instances, creating images, assigning metadata to instances and images and creating containers and objects
Glance API	A RESTful API that allows querying of VM image metadata as well as retrieval of the actual image

6.3 Server configuration settings and deployment for the model/tool and *in silico* trial repositories

This chapter describes how the code written for the implementation of the model/tool and *in silico* trial repositories has been deployed into production. A lot of developers regard the deployment stage of a web project as the final step in order to expose their software projects to the rest of the internet community and they leave it until the very end [7]. But as deployment, just like development, is fraught with the unexpected and the unanticipated, in most cases, specifically for substantial projects, this is not the most efficient approach. Much like testing, in a best-case scenario, the project should be deployed to a production environment right from the start. In this context, it has been decided that it is of great importance to deploy the model/tool and *in silico* trial repositories to the production environment from the very beginning.

The model/tool and *in silico* trial repositories have been structured in such a way that they can scale horizontally, in order to be able to operate across several machines. One of the reasons that the web application framework “Django” has been chosen for the implementation of the repositories is that Django lends itself very well to being deployed across lots of machines [7]. The functions of the model/tool and *in silico* trial repositories have been modularized so as to be able to scale with much less effort and not require the aid of very expensive server machines. Because of this modularity it will be very easy in the future, when needed, to configure a web server to route requests to different machines on a network. These machines could be a model/tool database server; an *in silico* trial database server; a memcached server; a static media server for CSS, JavaScript and graphic files; and an application server for Django dynamic pages. So far, we did not follow this approach and we deployed all of our model/tool and *in silico* trial repository code to a single virtual machine provided by FORTH.

A NginX web server has been set up in order to serve as the front-end web server serving requests for static media, such as style sheets and JavaScript files of the model/tool and *in silico* trial repositories, and forwarding requests for repository dynamic pages to the Apache server which will interface, using mod_wsgi, with the Django repository project. NginX is a fully functional web server that can be better suited for serving static media than Apache [7]. NginX has been configured to act as the public – facing web server, accepting all incoming requests for model/tool and *in silico* trial repositories. It will then divide up these requests between those for dynamic Django driven pages and those for static media files, forwarding the requests for dynamic pages to the Apache server and handling any requests for static media without involving Django project at all. The architecture that has been adopted looks much alike figure 23.

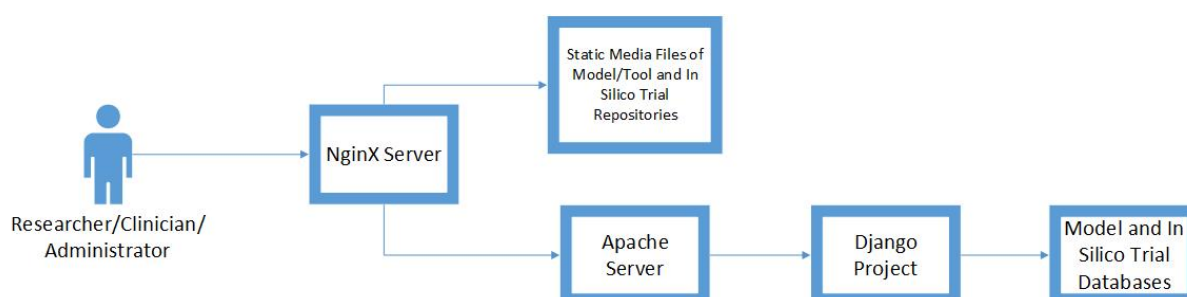


Figure 23: NginX is used to divide up requests between static media files and dynamic Django driven pages

In order for the NginX server to divide up the requests and forward those for dynamic pages to the Apache server, it is essential to create an IP-based virtual host for the NginX server. The term Virtual Host refers to the practice of running more than one web site on a single machine. Virtual hosts can be “IP-based”, meaning that there is a different IP address for every web site, or “named-based”, meaning that there are multiple names running on each IP address. In the case of model/tool and *in silico* trial repositories, an IP based virtual host is needed, since the repositories need a single IP address in order to be accessed by the researchers, clinicians and administrators. For the proper configuration of the NginX’s IP-based virtual host, a configuration file has been created which is depicted in the following figure:

```
server {
listen 80;
server_name 139.91.210.27
location / {
access_log /var/log/nginx/localhost.access.log;
proxy_pass http://127.0.0.1:8080;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
location /static {
root /home/isog/workspace/chic;
}
}
```

Listen on incoming requests on port 80

The IP of the Model and In Silico Trial web application

As it is not a static request, is will be forwarded to Apache server along with the IP address of the request

It is a static request and it will be served by NginX server

Figure 24: NginX’s IP-based virtual host configuration file

In the way that NginX has been configured (fig. 24), NginX server will listen for incoming requests on port 80. Any request that comes in to the NginX web server with a url path that starts with /static will be served by NginX server. In this case NginX will search for the requested static file (javascript, css, image) in the root directory of the Django project (/home/isog/workspace/chic) which holds all the files of the model/tool and *in silico* trial web application.

However any request with a url path that doesn’t begin with /static, is a request for a Django dynamic page and is going to be forwarded to the Apache server. However, as the two servers are installed on the same virtual machine provided by FORTH, all of the requests that get forwarded by NginX to Apache server will appear to have come from the IP address 127.0.0.1, which is localhost. In order to fix this, a reverse proxy add forward module for Apache, known in shorthand as mod_rpaf, has been installed.

Now that NginX server has been configured, Apache server should be configured in order to accept requests for repository dynamic pages from NginX server. Consequently, it is essential to create an IP-based virtual host for the Apache server. For the proper configuration of the Apache’s IP based virtual host, a configuration file has been created which is depicted in the following figure:

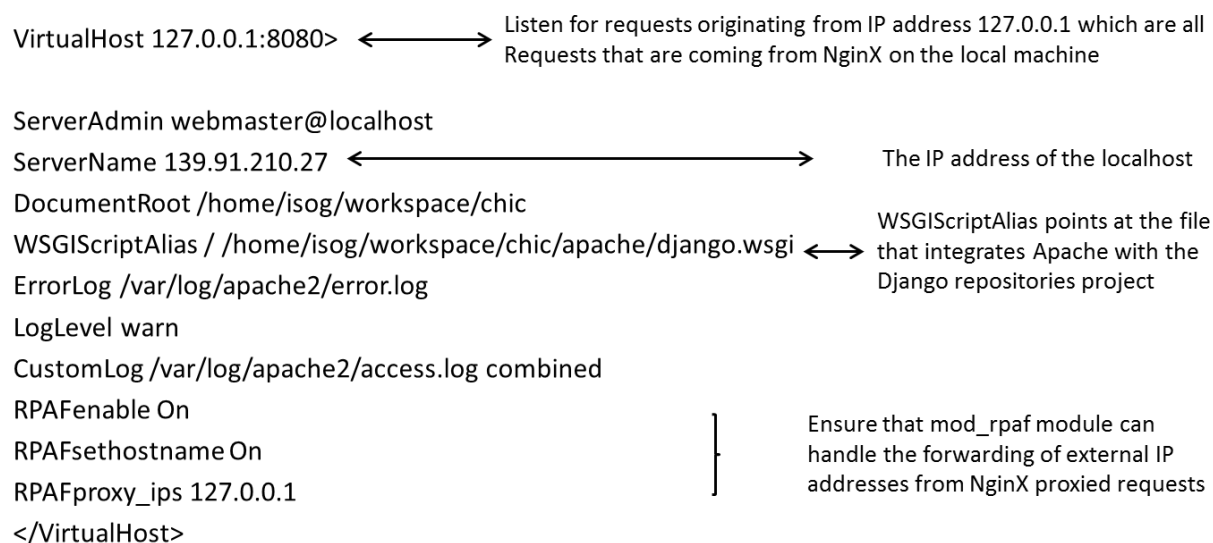


Figure 25: Apache's IP-based virtual host configuration file

In the way that Apache server has been configured (fig. 25), Apache will listen for requests originating from IP 127.0.0.1 which are all requests that are coming from NginX server. Furthermore in order to integrate Apache with the Django repositories project, the Web Server Gateway Interface (WSGI) has been installed which is an interface between web server and web applications for the python programming language.

After all this configuration, it has been ensured that when there is a request from the client (researcher, clinician etc.) for static files (CSS, javascript, images, etc.) only NginX server is going to respond, whereas when there is a request for dynamic pages, only Apache server is going to respond. In this way, web traffic has been divided between two servers.

6.4 Model/tool and in silico trial repositories in compliance with Django MVT pattern

The Django project implements the web-based interface which allows users (researchers, clinicians, administrators, etc.) to interact with the model/tool and *in silico* trial repositories. Django is a high-level Python Web Framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It is reassuringly secure, exceedingly scalable and it is free and open source.

Django project makes use of the MVT pattern. This software engineering pattern separates data (model), user interface (template), and data handling logic (view), so that one can be changed without affecting the others [8]. The benefits of this pattern are that designers can work on the interface without worrying about data storage or management and developers are able to program the logic of data handling without getting into the details of presentation. Figure 26 shows how each of the components of the MVT pattern interact with each other to serve a user request:

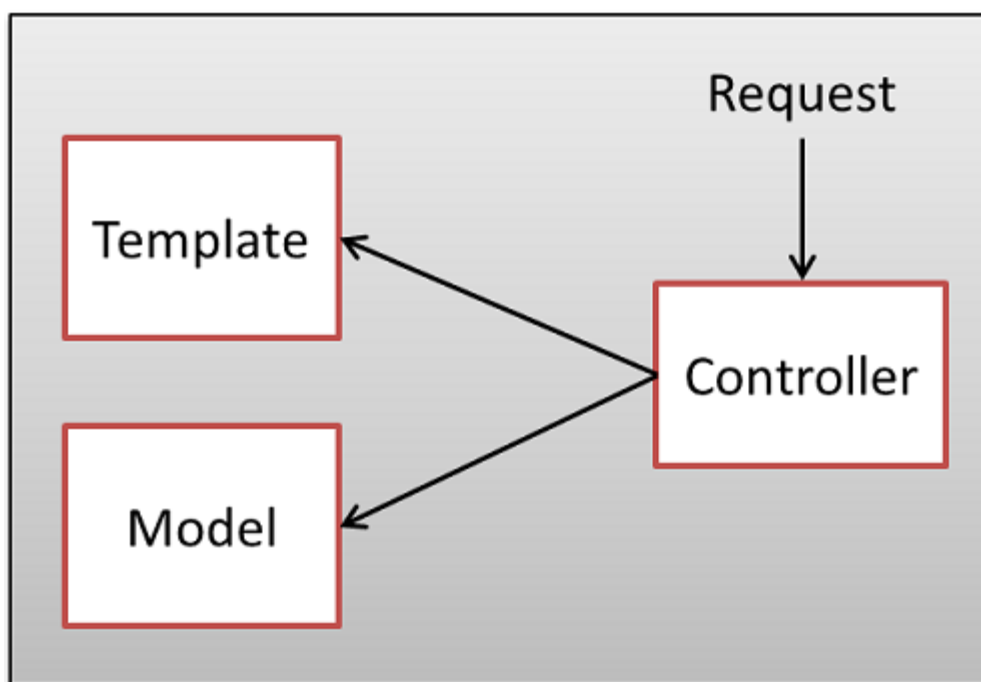


Figure 26: Components of the MVT pattern

A brief overview of the components depicted in figure 26 is the following [9]:

- The model is generally responsible for managing data and core business logic. Django models provide easy access to an underlying data storage mechanism, and can also encapsulate any core business logic, which much always remain in effect, regardless of which application is using it. Models exist independent of the rest of the system, and are designed to be used by any application that has access to them. The database manipulation methods that are available on model instances can be utilized even from the interactive interpreter, without loading a web server or any application-specific logic.
- A view function, or view for short, is simply a Python function that takes a Web request and returns a Web response. This response can be the html contents of a Web page, a redirect, a 404 error, an xml document, an image, etc. The view itself contains whatever arbitrary logic is necessary to return that response. The code of the view can live anywhere, as long as it is on the machine's Python path.
- While views are technically responsible for presenting data to the user, the task of how that data is presented is generally delegated to templates, which are an important part of Django development to be considered a separate layer entirely. Templates handle all the presentational details the user will see. Django provides a simple template language for this purpose, so that template designers don't need to learn python just to work with templates. Django's template language is not dependent on any particular presentation language. It is primarily used for HTML but can be used to generate any text-based format. However this template engine is just one tool that views can use to render a display for a user.

Even though the previous figure depicts the components of the MVT pattern and the connection between them, it does not depict the way model/tool and *in silico* trial repositories make use of MVT pattern. Figure 27 presents the component architecture of the repositories, which consists of the templates, the url dispatcher, the views, the data models, the MySQL driver and the databases:

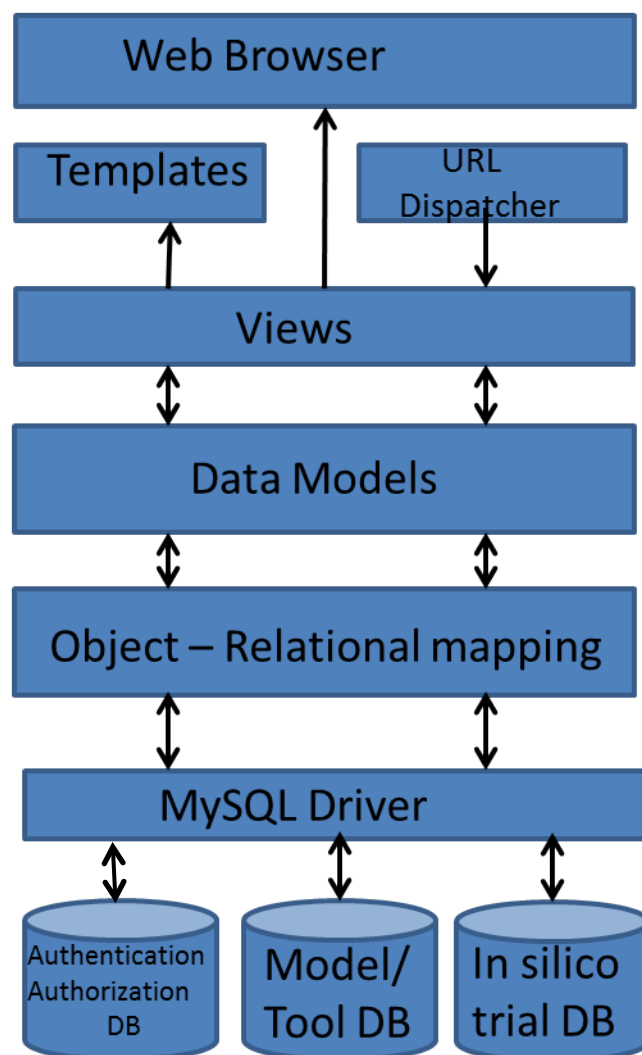


Figure 27: MTV pattern used in model/tool and *in silico* trial repositories

A brief description of the components and the control flow in figure 27 is the following:

- The url dispatcher maps the requested url to a view function and calls it. If caching is enabled, the view function can check to see if a cached version of the page exists and bypass all further steps, returning the cached version instead.
- The view function performs the requested function, which typically involves reading or writing to the database. It may include other tasks as well.
- The data model defines the data in python and interacts with it.
- After performing any requested tasks, the view returns an http response object (usually after passing the data through a template) to the web browser. Optionally the view can save a version of the http response object in the caching system for a specific length of time.
- Templates typically return html pages. The Django template language offers html authors a simple-to-learn syntax while providing all the power needed for presentation logic.

- Object relational mapping is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a “virtual object database” that can be used from within the programming language.
- In the lowest level, there are three MySQL databases, which define how data is stored. These three databases are the model/tool database, the *in silico* trial database and the authentication-authorization database whose basic principles are described in the deliverable “D8.1 Design of the CHIC repositories”.

The entity relational diagrams of these three databases, which have been retrieved from deliverable 8.1, are depicted, in the sake of completeness, in figures 28-30:

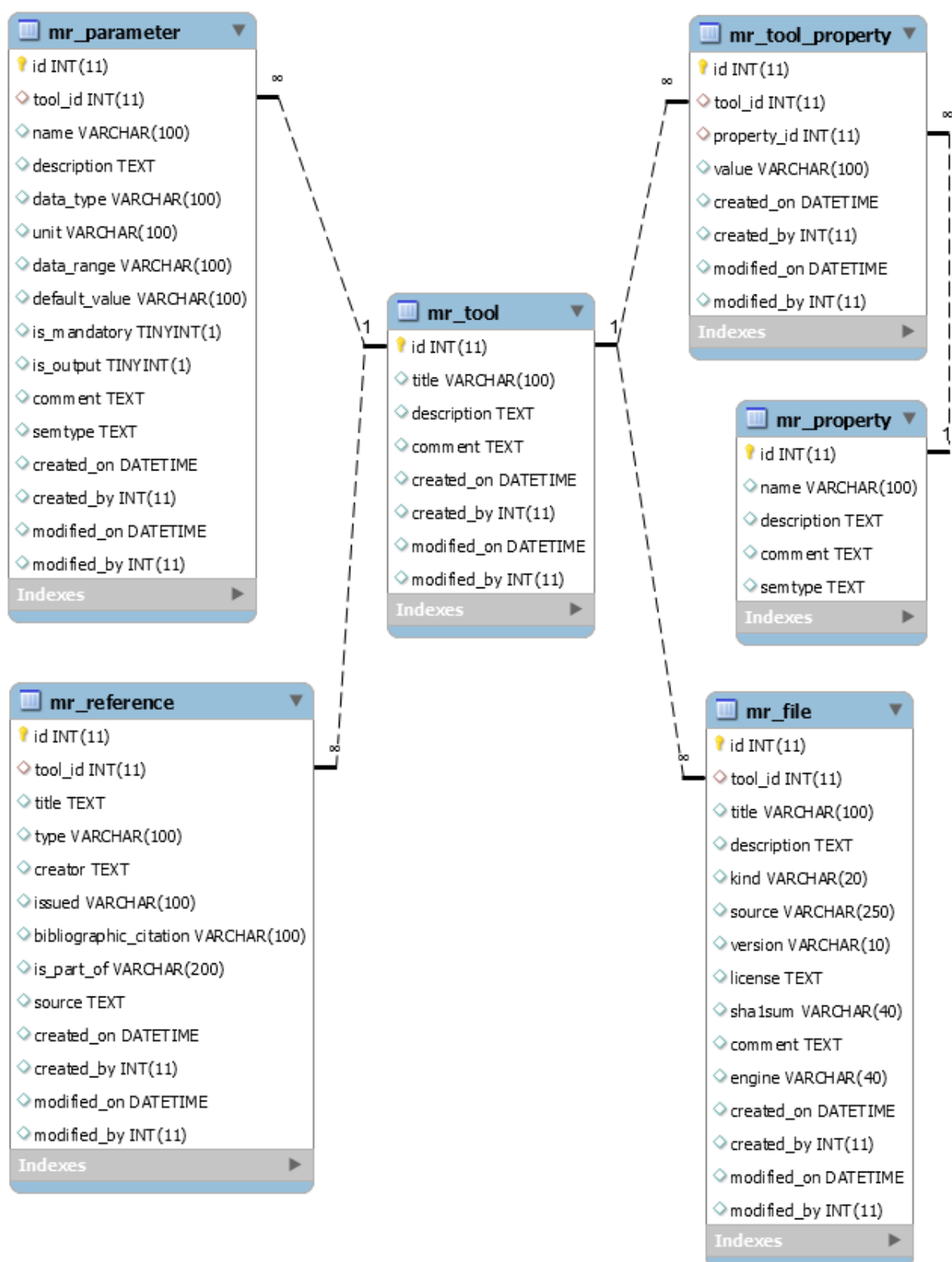


Figure 28: Entity-Relationship (ER) diagram of model/tool repository

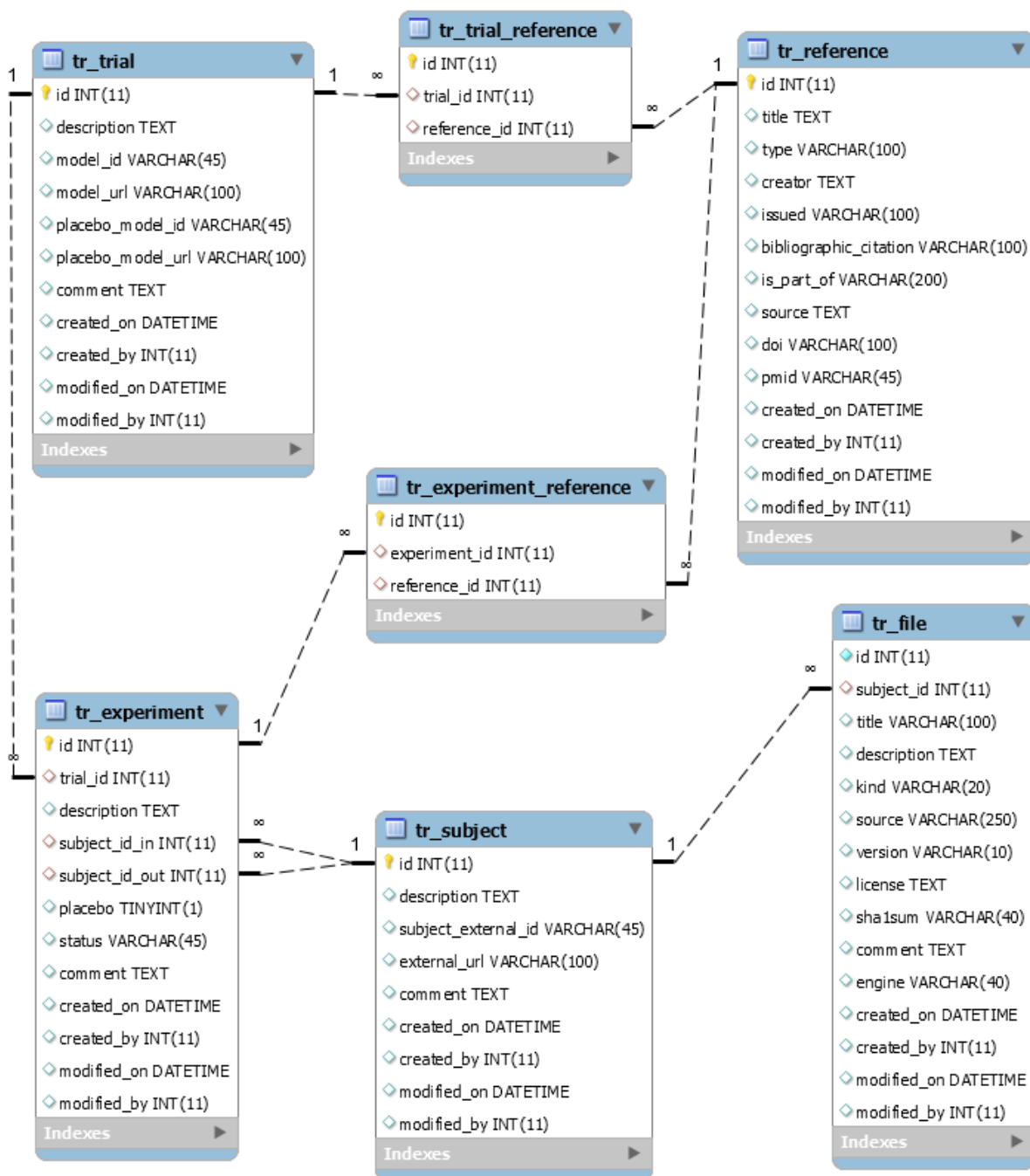


Figure 29: Entity Relationship (ER) diagram of *in silico* trial repository

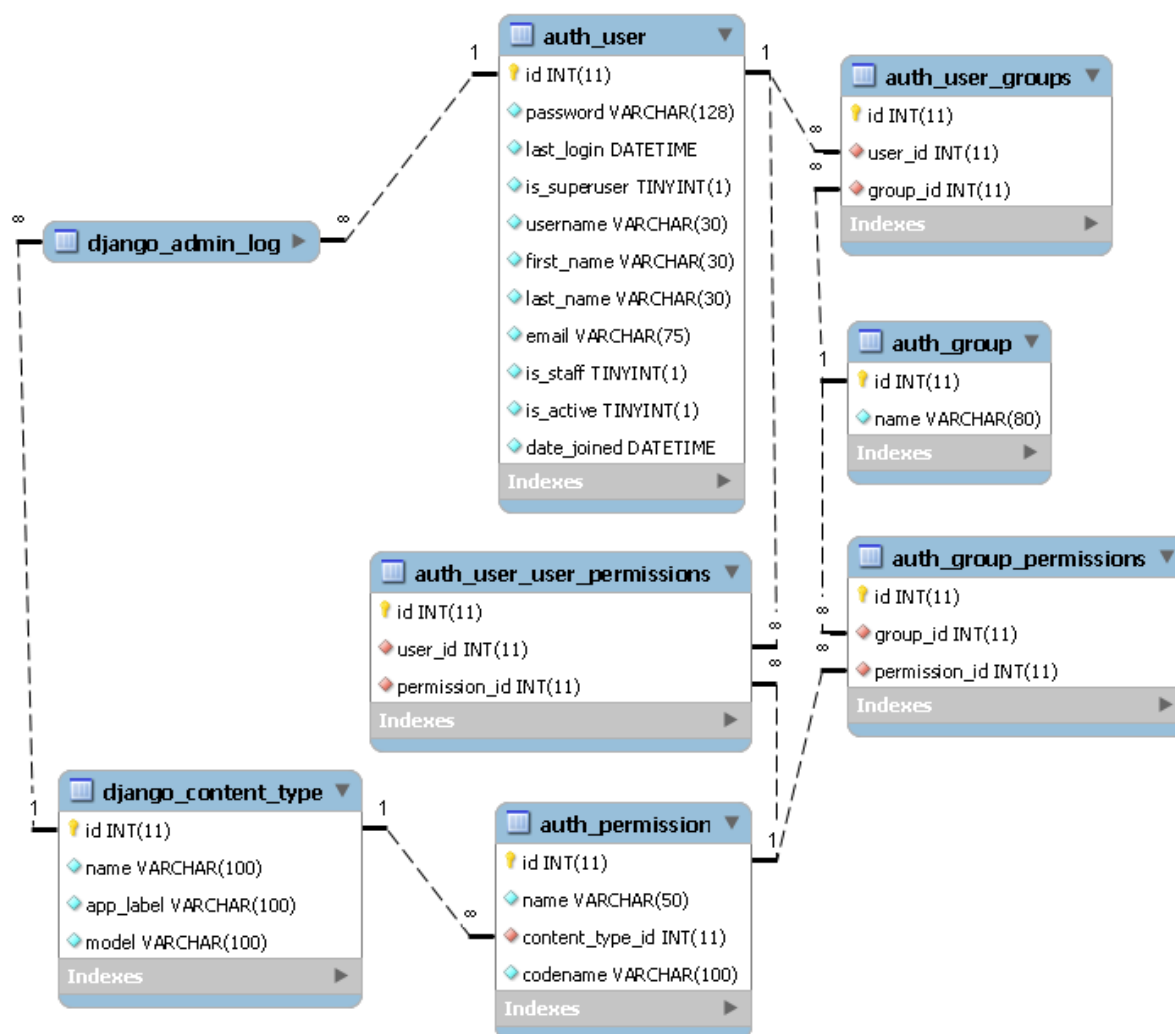


Figure 30: Entity Relationship (ER) diagram of tables associated with basic authentication and authorization

Most of the components depicted in figure 27 (apart from the browser and the databases) have been wrapped up into a whole Django project which is considered to be an installation of Django. This project is consisted of two applications, the model/tool application and the *in silico* trial application. The term application describes a python package that provides some set of features and applications may be reused in various projects. Moreover, applications include some combination of models, views, templates, template tags, static files, urls, middleware etc. and each application has each own models, views and urls.

In the context of CHIC, the model/tool application, which is inside the Django project, provides the web-based interface (GUI and services) in order to allow users to interact with the model/tool repository. The *in silico* trial application provides the web-based interface (GUI and services) in order to allow users to interact with the *in silico* trial repository.

Figure 31 depicts the way model/tool and *in silico* trial repositories exist inside the same Django project. As shown in figure 31, each application uses its own data models in order to create a virtual object database in python, and these data models always need to be synchronized with the corresponding database.

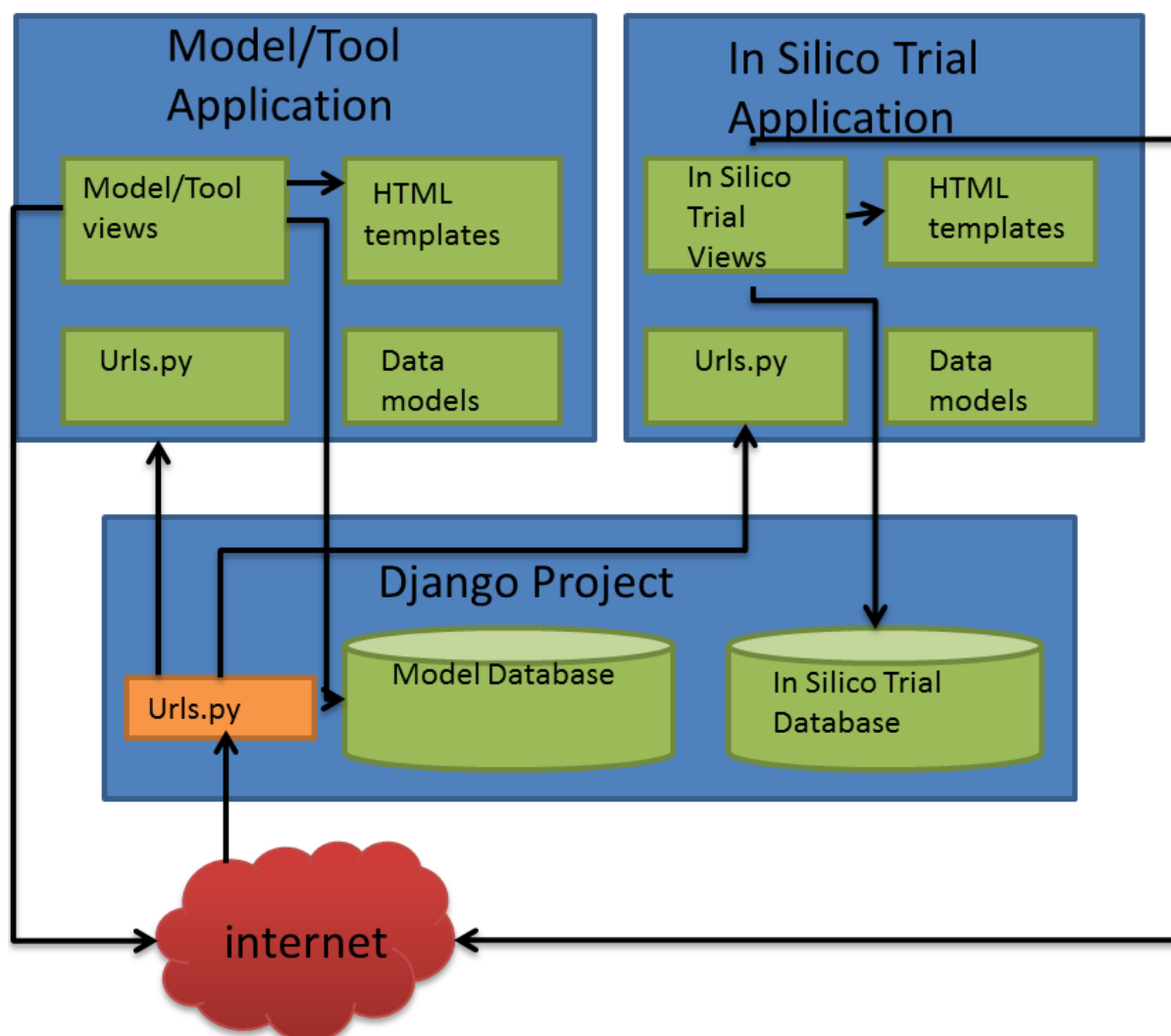
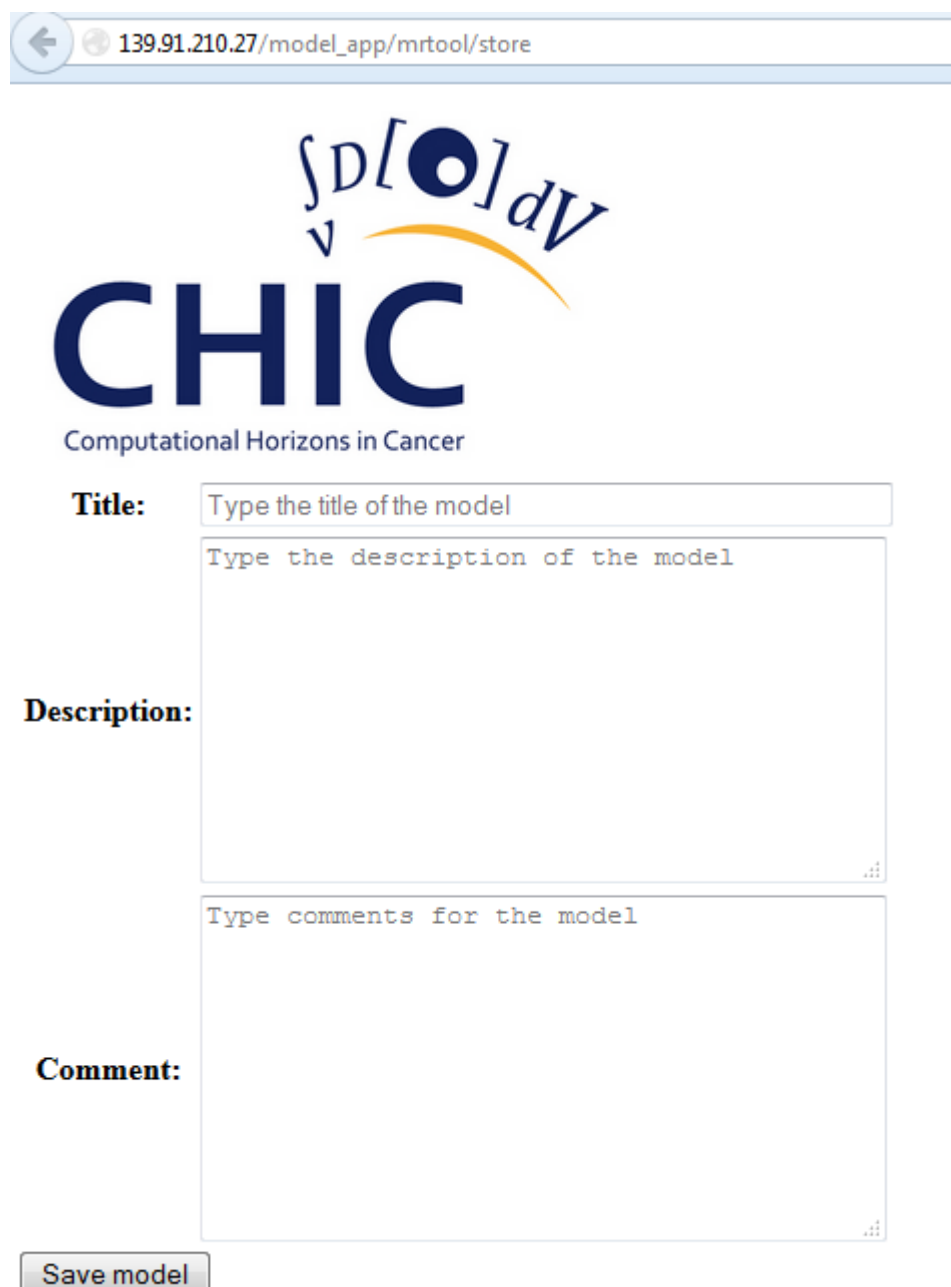


Figure 31: Model/tool and *in silico* trial applications inside the same Django project

According to figure 31, when a request for dynamic page comes in to the Apache server, it transfers control to the Django project. The file `urls.py` of the Django project analyses the url and decides which application should be used for this request. The application then decides which function in file `views.py` is called. The functions in `views.py` file prepare the data to be presented with the html template and return the html to the client. As shown in figure 31, all the code needed for the implementation of the model/tool and *in silico* trial repositories, apart from the mysql code, is organized in four components (the views, the data models, the html templates and the urls dispatchers). In every scenario of interaction between the user and the model/tool or the *in silico* trial repository, the following are going to be used: at least a model (Python class) from the data models component, a view function (Python function) from the views component, an html template (template which contains the static parts of the desired html output as well as some special syntax describing how dynamic content will be inserted) and the url dispatcher from the corresponding application which maps the requested URL with the corresponding view function.

In order to be clarified how the components in figure 31 operate so as to enable the interaction with the repositories, the code of the components that is going to be executed in order to satisfy requests for storing basic information of (hypo)models is presented below. When the user (researcher, clinician, administrator) wants to store in the model/tool repository a model, the first thing he/she has to do is to provide information about the title of the model/tool, the description of the model/tool and the comments (if any) for the model/tool. Any other information concerning the model/tool (properties, parameters, files, references, etc.) is going to be inserted later on in different pages of the web application. The page for inserting the basic information for models/tools is presented in figure 32:



139.91.210.27/model_app/mrtool/store

CHIC
Computational Horizons in Cancer

Title:

Description:

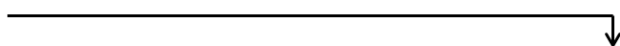
Comment:

Figure 32: Page for storing basic information of a model/tool

When the “Save model” button from figure 32 is pressed, a view function will be called in order to store the basic information of the model to the model/tool database. The basic information of the model is going to be stored in the table “mr_tool” of the model/tool database (Fig. 28). After the storage of the model’s basic information, the view function will transfer the user to another page where he/she could continue storing further information, related to the model (parameters, properties, files, etc.), to the model/tool repository.

The HTML template which is responsible for the presentation logic of the page for inserting basic information about the model is depicted in figure 33:

This is the requested URL when the researcher presses the “submit” button in order to store the basic information of a model



```
<form method="post" id="store" action="/storemodel/">
{% csrf_token %}
<br>
<label>Title:</label>
<input type="text" size="47" name="Title" >
</br>
<br>
<label>Description:</label>
</br>
<textarea rows="10" cols="50" name="Description" form="store"
</textarea>
<br>
<label>Comments:</label>
</br>
<textarea name=Comments rows="10" cols="50" form="store"
</textarea>
<input type="submit" value="Submit">
</form>
```

Figure 33: Django Template language for storing basic information of a model

After pressing the “Save model” button (fig. 32), Django runs through each url pattern in urls.py file (see fig. 34) successively, and stops at the first one that matches the requested url. By pressing the “Save model” button, the user actually makes an HTTP post with the requested url being `http://139.91.210.27/model_app/mrtool/store/`. The full path of the requested URL consists of the IP of the virtual machine provided by FORTH (139.91.210.27), the name of the application inside the Django project (model_app) which is responsible for the interaction with the model/tool repository, the name of the table (mrtool) which is going to accept a new record, and the name of the operation (store) that is going to be applied to the database table.

As Django lets you design urls however you want, with no framework limitations, there has been an effort to design a clean, elegant url scheme for the model/tool and *in silico* trial web application. After the presentation of the page by the html component and the http post request, the next component that is going to be presented is the urls dispatcher.

The urls dispatcher is a python module, informally called (url configuration) which is pure python code and is a simple mapping between url patterns (simple regular expressions) to python functions (views). The python module which is going to map the requested url (for storing the model in the database) to a specific view is presented below:

Django stops because this url pattern matches the requested url (/model_app/mrtool/store)

Now Django imports and calls the given view, which is a simple python function. The view gets passed (among others) an instance of HttpRequest

```
urlpatterns=patterns("",
    url(r'^model_app/mrtool/store/$', views.storemodel, ←
        name='storemodel'),
    url(r'^model_app/mrtool/delete/$', views.deletemodel,
        name='deletemodel'),
    url(r'^model_app/mrtool/allmodels/$', views.allmodels,
        name='allmodels'),
    url(r'^model_app/mrproperty/store/$', views.storeproperty,
        name='storeproperty'),
    .
    .
    .
```

Figure 34: The urls.py file, which belongs to model_app application (the application in charge of model/tool repository) is responsible for mapping the requested url to a specific view

As shown in figure 34, there is a pattern that matches the requested url, and as a result, Django imports and calls the given view (storemodel), which is a simple python function in charge of storing the necessary information to the model/tool database. The storemodel view function should not communicate directly with the model/tool database, because this would violate the MVT pattern shown in figure 83. Instead, storemodel view function makes use of the data models component in order to interact with the model/tool database. As it has been mentioned previously, the data models component provides a database-abstraction API that lets store, delete, update and view information from the corresponding database. The data models component consists of model classes which represent database tables. In the context of model/tool and *in silico* trial repositories, for each database table there is a corresponding python model class which provides the database-abstraction API for this table. The model class that is going to be used in order to satisfy the request for storing basic information of models is named MrTool and represents the database table mr_tool (see fig. 28). Figure 35 demonstrates the similarities between this data model class and the corresponding database table.

<code>create table 'mr_tool' (</code>	<code>Class MrTool(models.Model):</code>
<code>'id' int(11) NOT NULL, ←</code>	<code>id=models.IntegerField(primary_key=True)</code>
<code>'title' varchar(100) DEFAULT NULL, ←</code>	<code>title=models.CharField(max_length=100,blank=True)</code>
<code>'description' text, ←</code>	<code>description=models.TextField(blank=True)</code>
<code>'comment' text, ←</code>	<code>comment=models.TextField(blank=True)</code>
<code>'created_on' datetime DEFAULT NULL, ←</code>	<code>created_on=models.DateTimeField(blank=True,null=True)</code>
<code>'created_by' int(11) DEFAULT NULL, ←</code>	<code>created_by=models.IntegerField(blank=True,null=True)</code>
<code>'modified_on' datetime DEFAULT NULL, ←</code>	<code>modified_on=models.DateTimeField(blank=True,null=True)</code>
<code>'modified_by' INT(11) DEFAULT NULL, ←</code>	<code>modified_by=models.IntegerField(blank=True,null=True)</code>
<code>PRIMARY KEY('id'),</code>	<code>class Meta:</code>
<code>KEY 'mr_tool_title_idx' ('title')</code>	<code>managed=True</code>
<code>) ENGINE=InnoDB DEFAULT CHARSET=latin1;</code>	<code>db_table='mr_tool'</code>
	<code>app_label='model_app'</code>

Figure 35: Data model class and corresponding database table in model/tool repository

As shown in figure 35:

- Each data model class has to provide model meta options
- If a model exists outside of the standard locations, the model must define which app it is part of. For example `app_label='model_app'` means that this data model belongs to the model/tool application (the application responsible of the model/tool repository).

- Managed=True in model meta options means that Django will manage the database table's lifecycles.
- Db_table='mr_tool' means that the name of the database table to use for the model is mr_tool.

Now that the model class MrTool (which represents the table mr_tool) has been defined, the "storemodel" view function can make use of it in order to store the necessary information to the model/tool repository.

Figure 15 presents the last piece of the puzzle, the view component, which will make use of the data model component in order to store the basic information of a model to the model/tool database. It should be noted that from the HttpRequest, the view function has access to the basic information of the model that the researcher wants to store to the repository (via the page).

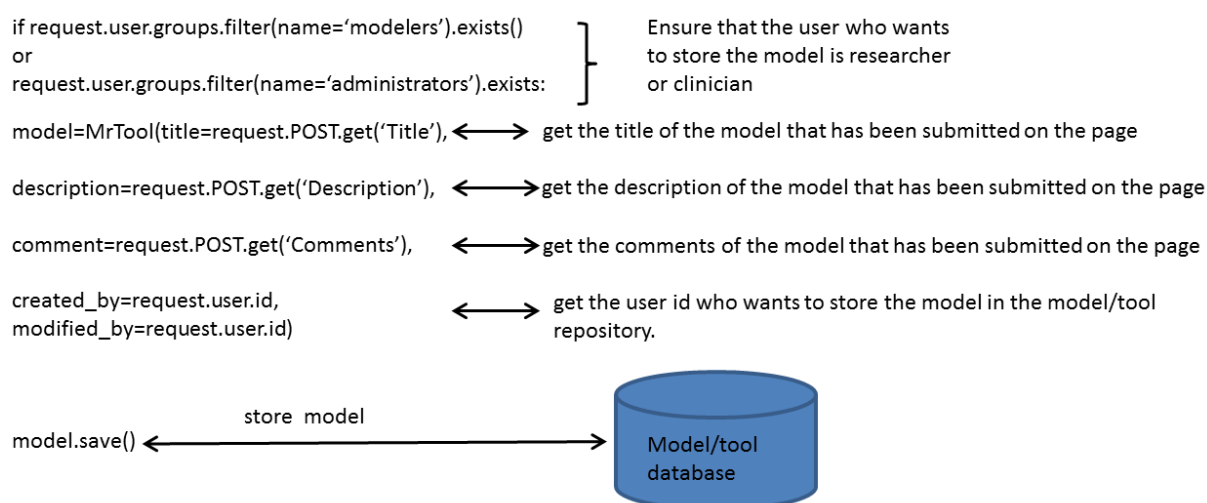


Figure 36: Django view function for storing basic information of a model

Figure 36 demonstrates how the view function "storemodel" (which belongs to the view component) makes use of the MrTool model class (see fig. 35), which belongs to the data models component, in order to store the basic information of the model to the model/tool database. As it is shown in figure 36, the view function ensures that the user who wants to store the model is a researcher or a clinician. Then the view function gets all the necessary information that the user has submitted from the page, and then stores this information in the Repository.

As the MVT pattern is the core software architectural pattern in the model/tool and *in silico* trial repositories, in the following we demonstrate in brief the way this pattern is used in those repositories and, more specifically, how the components of this pattern (data models, views, templates) cooperate in order to enable the interaction with the repositories. The scenario that has been chosen for the demonstration of MVT pattern is the storage of the basic information of a model (title, description, comments) that the user (researcher, clinician, etc.) provides via a page.

6.5 Authentication in model/tool and in silico trial repositories

The model/tool and the *in silico* trial repositories make use of the security framework introduced in deliverable “D5.2 - Security guidelines and initial version of security tools”. Therefore, the users are not directly authenticated by the repositories (service providers) themselves but rather by the CHIC authentication broker (identity provider) to support single sign-on (SSO). This procedure is called brokered authentication. The CHIC security framework further distinguishes between brokered authentication for web services including REST and for websites. The procedure for authenticating the user to the model/tool or the *in silico* trial repository website is the following:

The brokered authentication for the model/tool and *in silico* trial repositories web sites makes use of SAML web browser SSO profile as suggested by the CHIC security framework. The SAML web browser SSO profile is initiated by an end user who visits the protected model/tool and *in silico* trial repositories websites, also called service providers (SP). The SP redirect the user to the assertion provider (also called identity provider (IdP)) passing through an authentication request. The IdP will request the user to authenticate and upon successful authentication the IdP will issue an identity assertion for the user containing all information on the user needed by the SP to authenticate and authorise them. The assertion is then sent back to the SP that will use it to determine whether the user is allowed to access the requested resource. The sequential diagrams which depict the brokered authentication mechanism for the model/tool and *in silico* trial repositories websites are presented in figures 37 and 38 respectively.

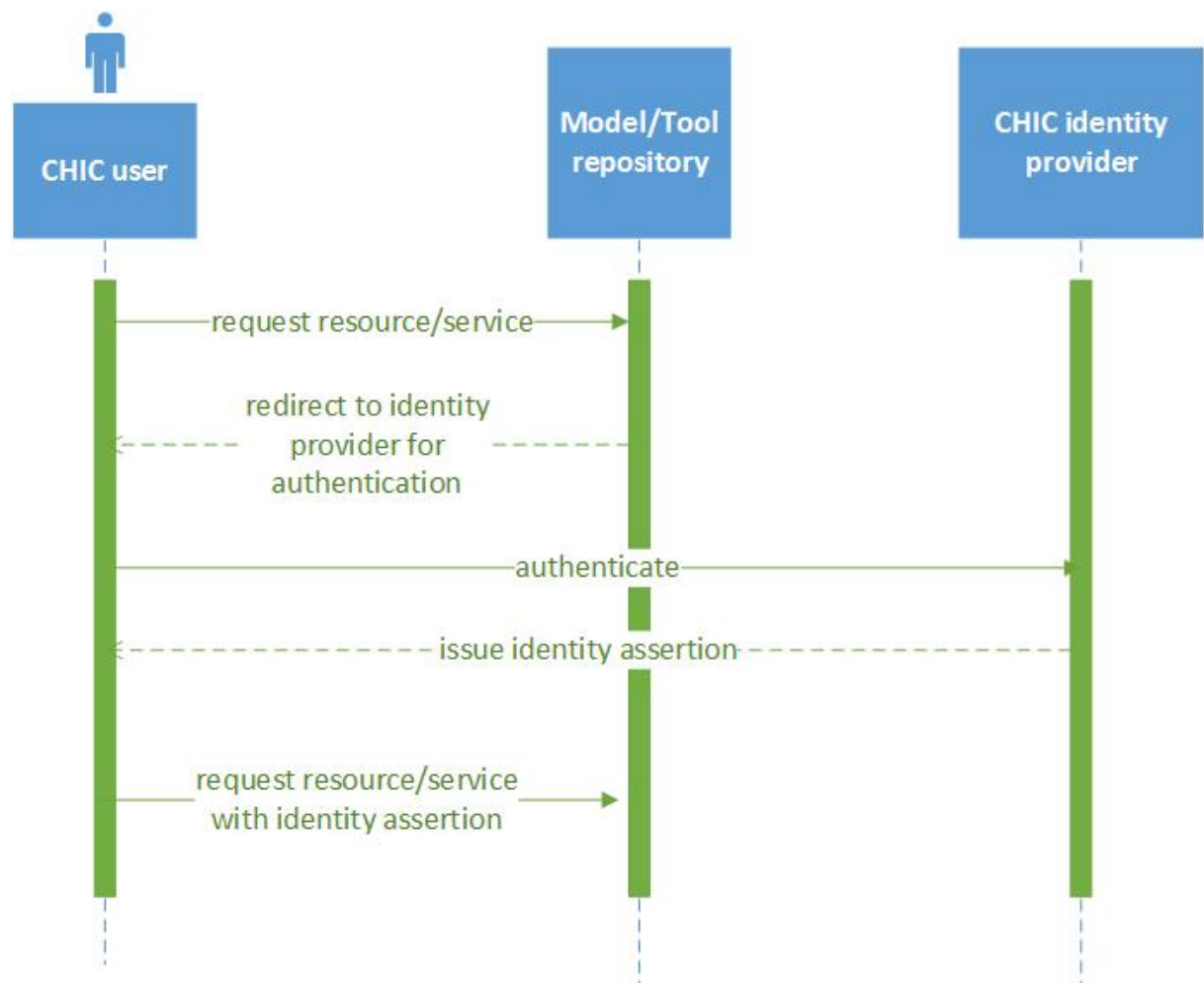


Figure 37: Brokered authentication for the model/tool repository web site

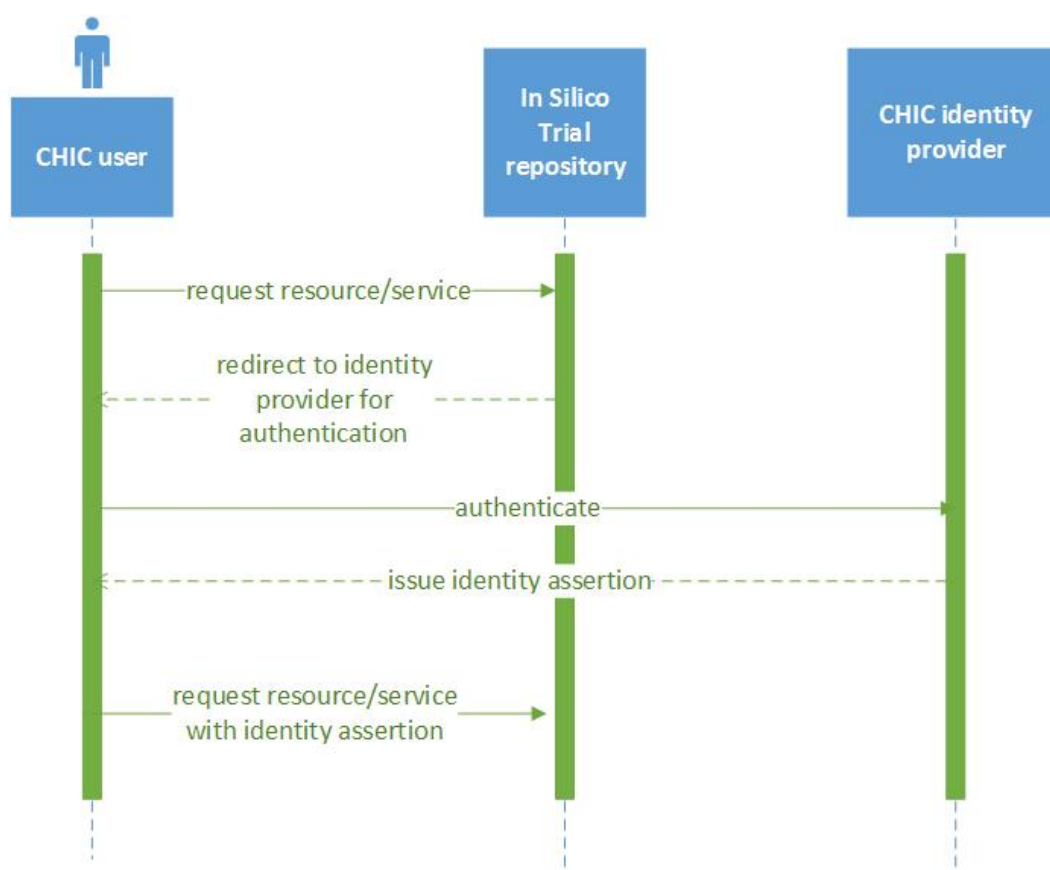


Figure 38: Brokered authentication for the *in silico* trial repository web site

As the model/tool and *in silico* trial repositories provide complete access to the features of the databases with the help of a REST interface, the security token service (STS) provided by CHIC is fully integrated in the authentication process. Before calling the REST service of the model/tool or the *in silico* trial repositories the client needs to send a SOAP (Simple Object Access Protocol) request containing an RST (RequestSecurityToken) to the STS. The STS then returns the identity assertion as a SAML token, embedded in a RSTR (RequestSecurityTokenResponse). The SAML token can then be passed to the REST service through the HTTP authorization header. The sequential diagrams which depict the brokered authentication mechanism for the model/tool and *in silico* trial repositories web services are presented in figures 39 and 40 respectively.

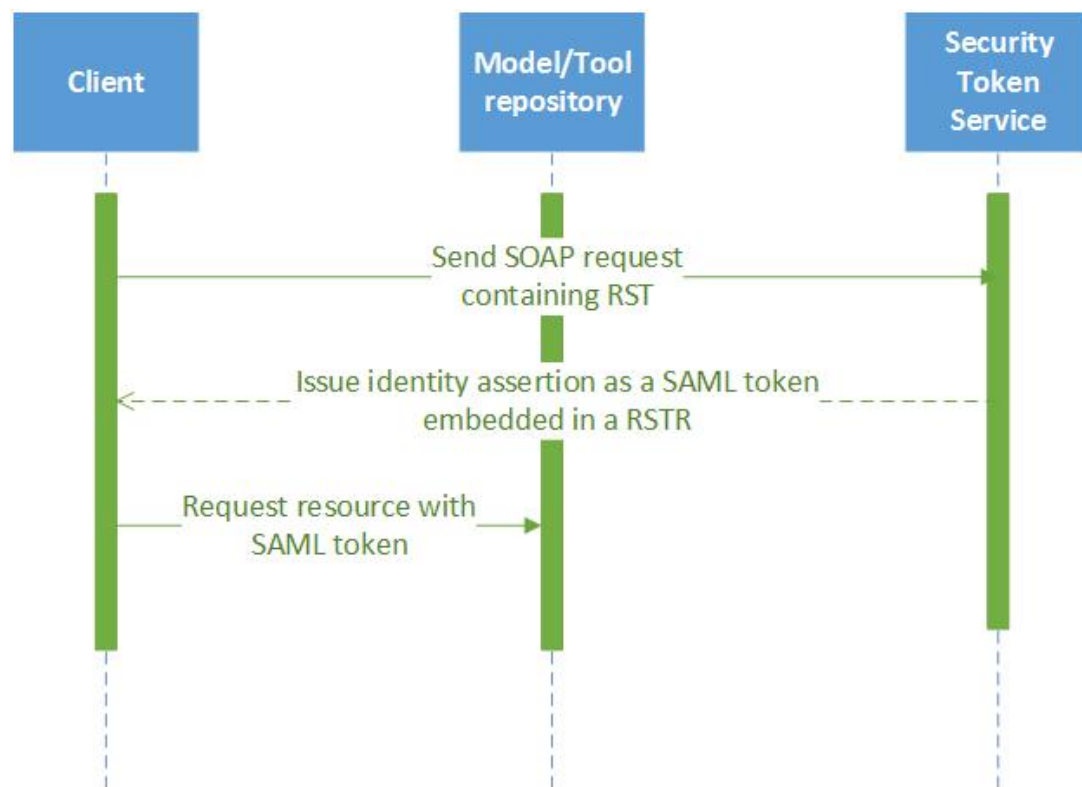


Figure 39: Brokered authentication for the model/tool repository web services

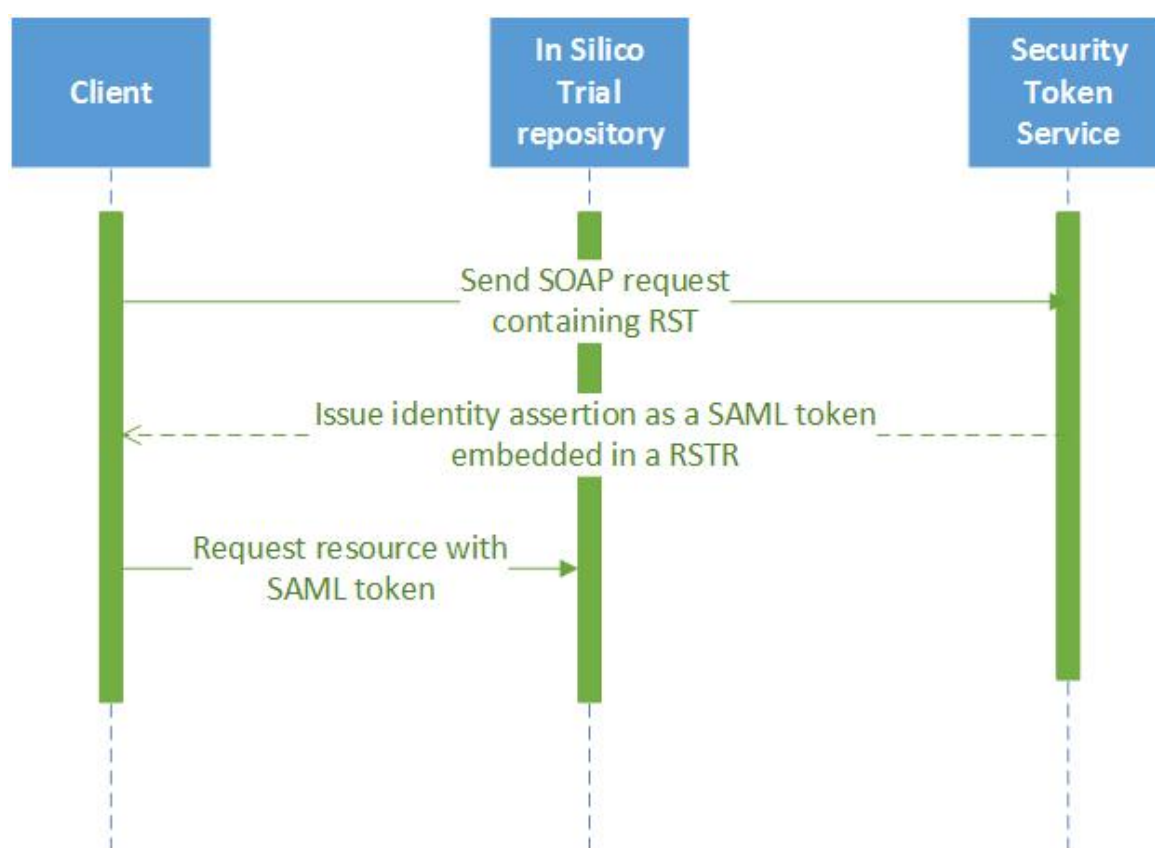


Figure 40: Brokered authentication for the *in silico* trial repository web services

6.6 Model/tool and *in silico* trial repositories administrator site

This chapter is about the administrator site of model/tool and *in silico* trial repositories. The administrator interface is a universal feature needed in any web application that stores and manages data. The administrator site is going to be limited to trusted administrators within the consortium in order to allow them to add users, authenticate users, add groups, add permissions and change the content of model/tool and *in silico* trial repositories. The use case diagram in figure 41 depicts the relationship between the administrator and the different use cases in which they are involved.

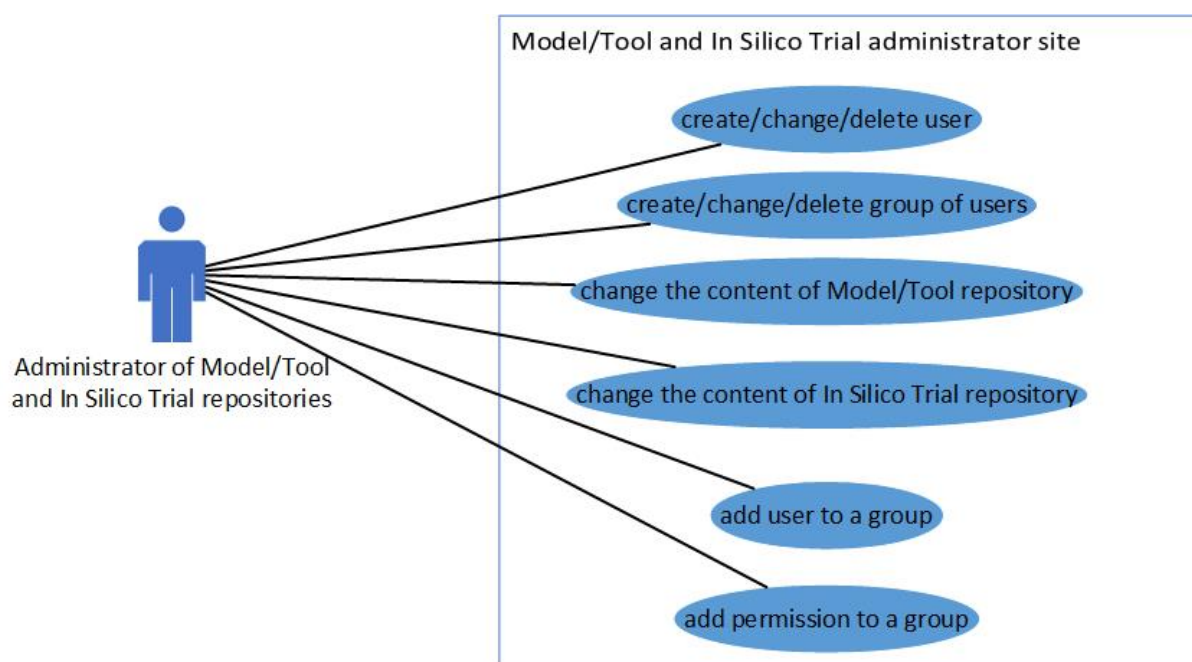


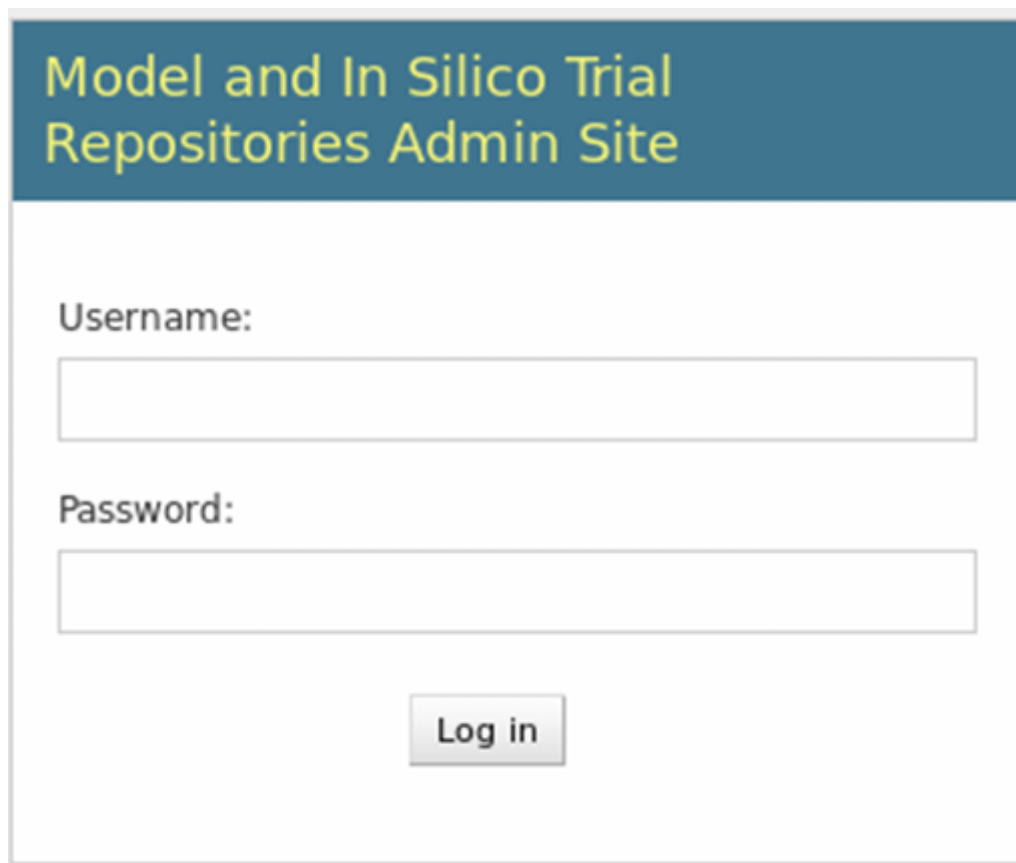
Figure 41: Use case diagram which overviews the usage requirements for model/tool and *in silico* trial repositories administrator site

As shown in figure 41, an administrator can perform the following tasks:

- The administrator can create, change (the information), or delete a user. This user could be a clinician, a researcher, etc. More precisely he/she can authorize a user and save user's information (username, password, first name, last name, email address, active or not) to the system. According to deliverable "D2.2 Scenario based user needs and requirements" there are two types of users, the consortium users and the external users. Both types of users could register in the CHIC platform using a public CHIC registration link. Since there is no trust between an unknown user and the CHIC framework, publicly registered users have limited access. Only a CHIC user administrator can grant these users more access, given that the administrator trusts the user's identity.
- The administrator can create a group of users. This group could be the clinicians group, the researchers/modellers group, the administrators group, etc.
- The administrator can add permissions to a group, such as add, change and delete permissions to specific content of the repositories. For example an administrator could grant the add, change and delete permissions to the researchers group for the model/tool repository and grant only the view permission to the clinicians group for the model/tool repository.
- The administrator can add users to a group. The administrator by adding a user to a group, he/she grants him/her the proper permissions as far as the model/tool and *in silico* trial repositories are concerned.
- The administrator can change the content of the model/tool repository. An administrator has full access to the model/tool database and he/she has the permission to change the content of the whole database.
- The administrator can change the content of the *in silico* trial repository. An administrator has full access to the *in silico* trial database and he/she has the permission to change the content of the whole database.

In the remaining of this section some screenshots of the model/tool and *in silico* trial repository administrator site are going to be presented.

The login page for the model/tool and *in silico* trial repository is presented in figure 42:



Model and In Silico Trial Repositories Admin Site

Username:

Password:

Log in

Figure 42: Login page of the administrator site for model/tool and *in silico* trial repositories

The home page of the administrator site which allows the administrators to add/change group, add/change user, and to change the content of the model/tool and *in silico* trial repositories, is presented in figure 43:

139.91.210.27/admin/

Model and In Silico Trial Repositories Admin Site

Site administration

Authentication and Authorization	
Groups	+ Add Change
Users	+ Add Change

Model_App	
Mr files	+ Add Change
Mr parameters	+ Add Change
Mr propertyts	+ Add Change
Mr references	+ Add Change
Mr tool propertyts	+ Add Change
Mr tools	+ Add Change

Trial_App	
Tr experiment references	+ Add Change
Tr experiments	+ Add Change
Tr files	+ Add Change
Tr references	+ Add Change
Tr subjects	+ Add Change
Tr trial references	+ Add Change
Tr trials	+ Add Change

Recent Actions	
My Actions	None available

Figure 43: Home page of the administrator site for model/tool and *in silico* trial repositories

As shown in figure 43, the administrator is allowed, through the model/tool and *in silico* trial administrator site, to change the contents of the tables which are defined in the entity relationship diagram of model/tool repository (fig. 28), change the contents of the tables which are defined in the entity relationship diagram of *in silico* trial repository (fig. 29) and change the contents of the tables associated with basic authentication and authorization (fig 30).

The page of the administrator site which allows the administrators to select a group in order to change it or delete it is presented in figure 44. As shown in figure 44, the administrator has already created 3 groups, the administrators, the researchers and the clinicians.

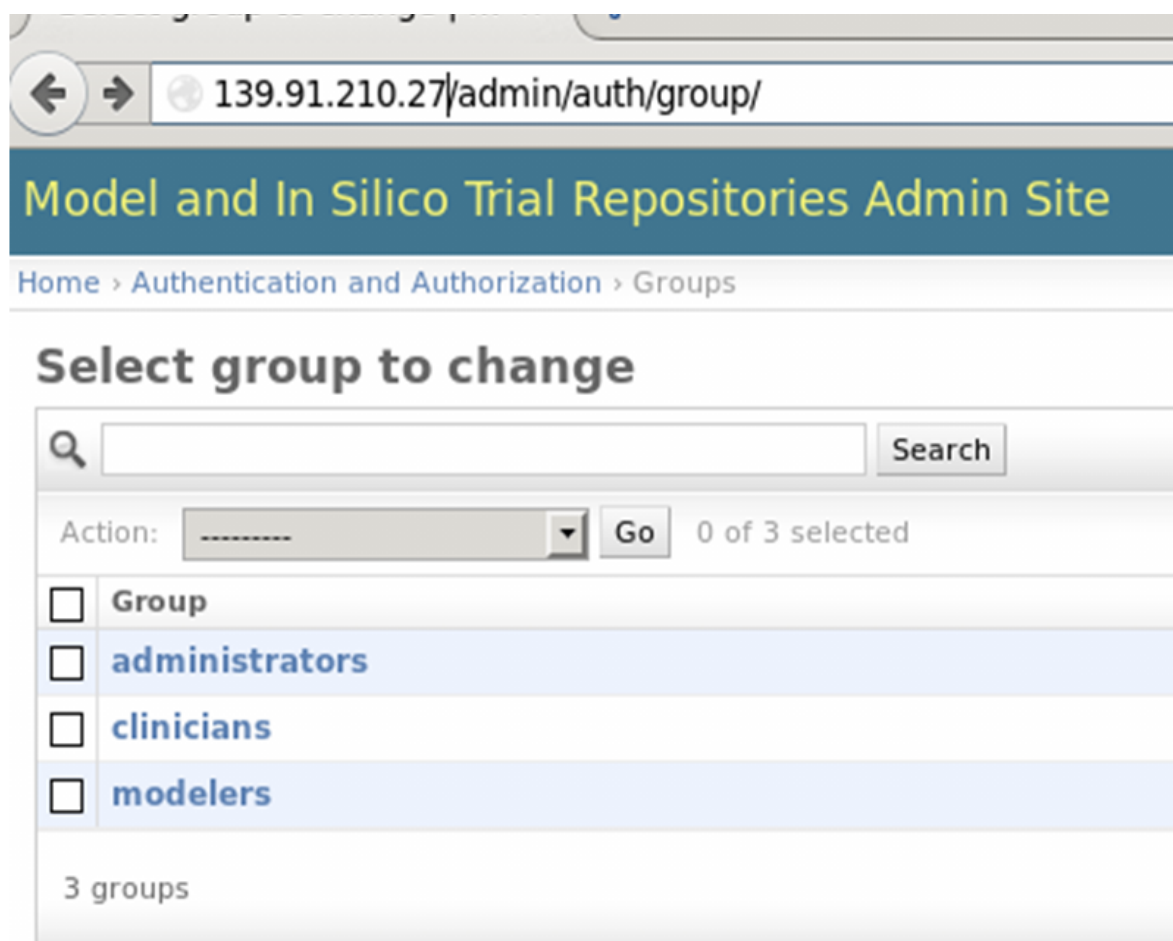


Figure 44: Groups of users' page of the administrator site for the model/tool and *in silico* trial repositories

It should be noted that clinicians have read permission to the model/tool repository, read and create permissions to the *in silico* trial repository and they can change or delete from the *in silico* trial repository only the content which has been created by them. The modellers (researchers) have read and create permissions to the model/tool repository and they can delete (or change) from the model/tool repository only the content which has been created by them. Moreover, modellers have read and create permissions to the *in silico* trial repository and they can delete (or change) from the *in silico* Trial repository only the content which has been created by them.

Figure 45 depicts how the administrator site of the model/tool and *in silico* trial repositories provides the functionality for the administrator to add a user to a group in order for the user to get the right permissions for the repositories.

Change user

Username:
Required: 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:
Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Personal info

First name:
Last name:
Email address:

Permissions

☒ **Active**
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☐ **Staff status**
Designates whether the user can log into this admin site.

☐ **Superuser status**
Designates that this user has all permissions without explicitly assigning them.

Groups:

The groups this user belongs to. A user will get all permissions granted to each of his/her group. Hold down "Control", or "Command" on a Mac, to select more than one.

Available groups

administrators
clinicians

Chosen groups

modelers

User permissions:

Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.

Available user permissions

admin ! loc entru ! Can add loc entru

Chosen user permissions

Figure 45: Administrator adds a user (modeller) to a group (modellers group)

Figure 46 shows the way the administrator can change the information of a model's parameter through the administrator site.

Model and In Silico Trial Repositories Admin Site

Home > Model_App > Mr parameters > MrParameter object

Change mr parameter

Id:

Tool:

Name:

Description:

Data type:

Unit:

Data range:

Default value:

Is mandatory:

Is output:

Comment:

Semtype:

Figure 46: Administrator changes the information of a model's parameter

Figure 47 shows the way the administrator can add permissions to a group (in this case the clinicians group).

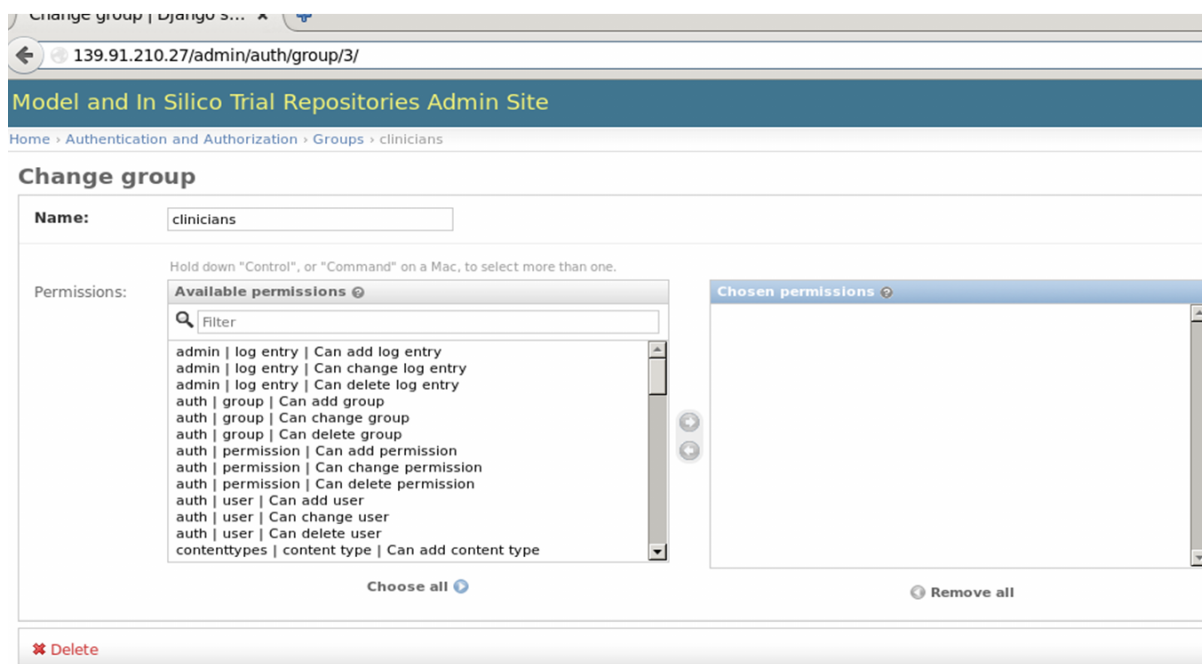


Figure 47: Administrator adds permissions to clinicians group

Figure 48 shows the way the administrator can change the information related to the users (clinicians, researchers, etc.).

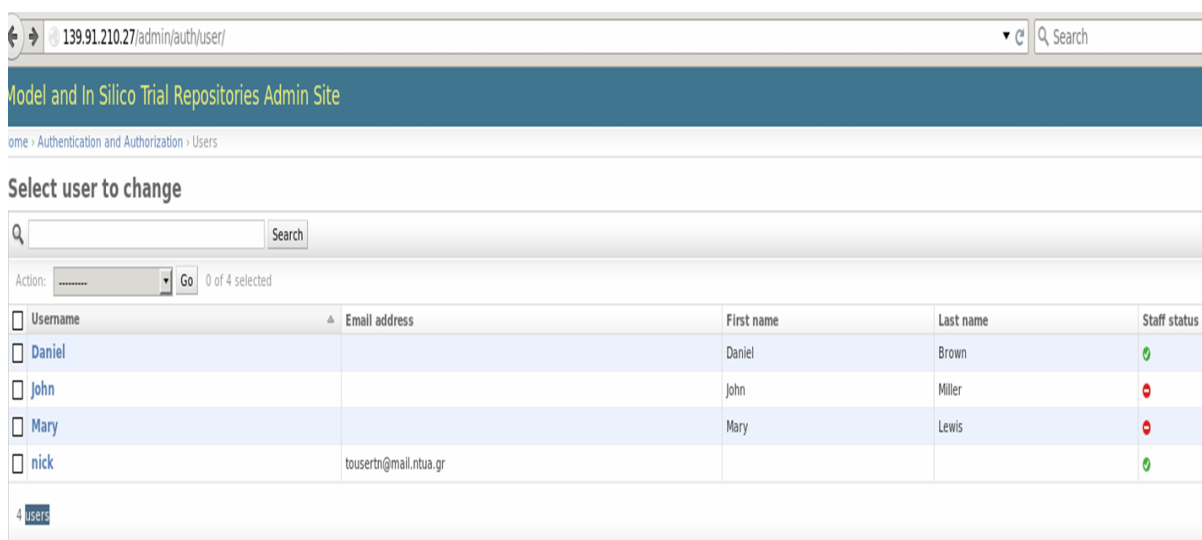


Figure 48: Administrator changes the information related to the users

Figure 28 shows the way the administrator can store an experiment into the *in silico* trial repository through the administrator site.

The screenshot shows a web browser window with the URL `139.91.210.27/admin/trial_app/trexperiment/add/`. The page title is "Model and In Silico Trial Repositories Admin Site". The breadcrumb trail is "Home > Trial_App > Tr experiments > Add tr experiment". The main heading is "Add tr experiment". The form contains the following fields:

- Id:** A text input field.
- Trial:** A dropdown menu with a plus icon.
- Description:** A large text area.
- Subject id in:** A dropdown menu with a plus icon.
- Subject id out:** A dropdown menu with a plus icon.
- Placebo:** A text input field.
- Status:** A text input field.
- Comment:** A large text area.
- Created on:** A section with "Date:" and "Time:" inputs, and buttons for "Today" and "Now". A note below says "Note: You are 2 hours behind server time."
- Created by:** A text input field.
- Modified on:** A section with "Date:" and "Time:" inputs, and buttons for "Today" and "Now". A note below says "Note: You are 2 hours behind server time."

Figure 49: Administrator stores an experiment through the administrator site

6.7 Graphical user interface of model/tool and in silico trial repositories

In this chapter, some screenshots of the web-based user interface and most of its functionalities, concerning the model/tool and *in silico* trial repositories will be presented. The user interface for the model/tool and *in silico* trial repositories focuses on anticipating what users (researchers, clinicians, etc.) might need to do and ensures that the interface has elements that are easy to access, understand, and use, in order to facilitate those actions. Bringing together concepts from interaction design, visual design and information architecture, the graphical user interface for the model/tool and *in silico* trial repositories is based on the following generally accepted principles:

- **The interface should be simple.** The best interfaces are almost invisible to the user. They avoid unnecessary elements and are clear in the language they use on labels and in messaging.
- **The user interface should make use of common user interface elements.** By using common elements in the graphical user interface, users feel more comfortable and are able to get things done more quickly. It is also important to create patterns in language, layout and design throughout the site to help facilitate efficiency. Once a user learns how to do something, they should be able to transfer that skill to other parts of the site.
- **The page should be structured based on importance.** Careful placement of items can help draw attention to the most important pieces of information and can aid scanning and readability.
- **Colour and texture should be used strategically.** There is the potentiality to direct attention toward or redirect attention away from items using colour, light, contrast and texture to your advantage.

- **The system should communicate what's happening.** The users should always be informed of location, actions, changes in state, or errors. The use of various user interface elements to communicate status and, if necessary, next steps can reduce frustration for the user.
- **Defaults are of great importance.** Defaults could reduce the burden on the user. This becomes particularly important when it comes to form design where there is the opportunity to have some fields pre-chosen or filled out.

6.7.1 Graphical user interface of model/tool repository

The graphical user interface for the model/tool repository is based on the use cases presented in figures 50 and 51. The use case diagram in figure 50 depicts the relationship between the researcher/modeller and the different use cases in which they are involved, concerning the model/tool repository. The use case diagram in figure 51 depicts the relationship between the clinician and the different use cases in which they are involved, concerning the model/tool repository.

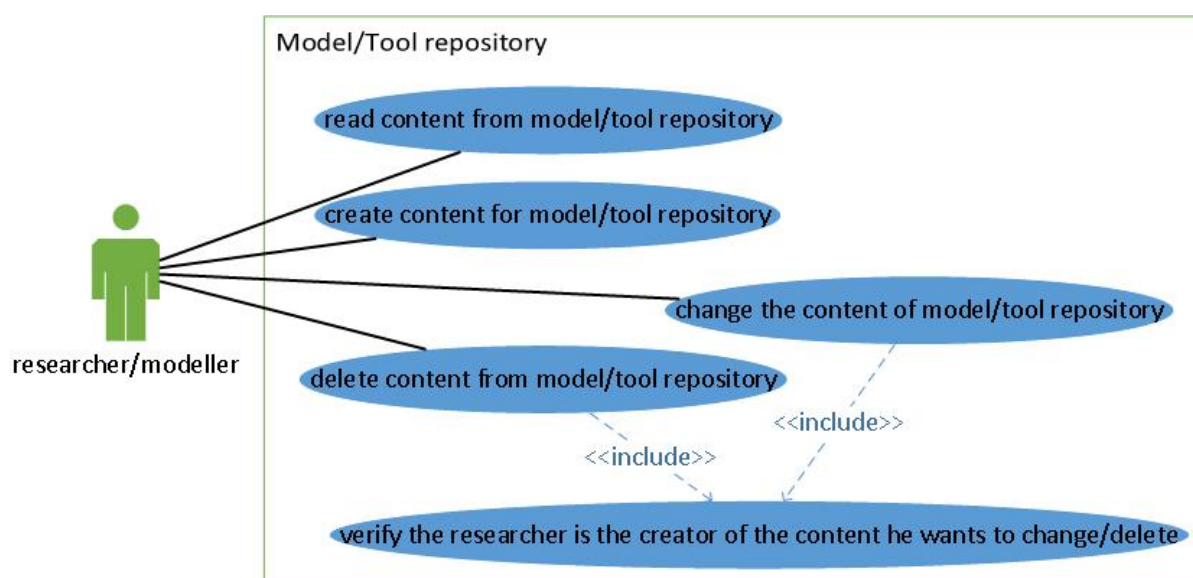


Figure 50: The researcher/modeller has read, create, change and delete permissions to the model/tool repository

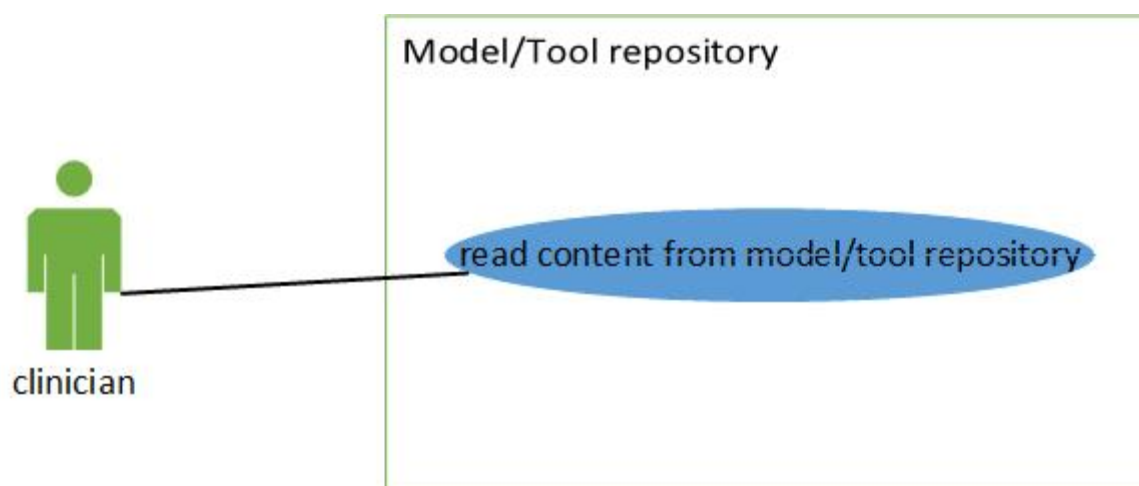




Figure 51: The clinician has read permission to the model/tool repository

Some screenshots of the user interface for the model/tool repository, based on the use case diagrams (fig. 50, fig. 51) will be presented in the remaining chapter.

Figure 52 presents the page where the user (researcher/modeller) stores the basic information of their model. As shown in figure 52, in order for the researcher/modeller to store a new model, they have to store their name, a brief description of the model and any necessary comments for the model. The information the user stores through the page presented in figure 52, is related to the `mr_tool` table of the entity-relationship diagram of the model/tool repository (fig. 28).


139.91.210.27/model_app/mrtool/store



CHIC
Computational Horizons in Cancer

[Home](#) / [Log out](#)

Title:

Description:

Comment:

Figure 52: Page for storing the basic information of a new model

Figure 53 presents the page where the user (researcher/modeller) stores a file associated with a specific model/tool. As shown in figure 53, the user has to select the model (from those models that are already stored in the model/tool repository), with which the file is associated, then they have to enter some fields related to the metadata of the file (title, description, version, license, etc.) and before the “Submit button” is pressed the user has to browse the file they want to store. The information the user stores through the page presented in figure 53, is related to the `mr_file` table of the entity-relationship diagram of the model/tool repository (fig. 28).

139.91.210.27/model_app/mrfile/store

[Home](#) / [Log out](#)

Model/tool associated with this file: Molecular Dynamics of clinical mutations in oncogenic receptors ▼

Title:

Description:

Type textual description of what this file represents.

Type of file: document ▼

Version:

License:

Comment:

Engine:

File:
 No file selected.

Figure 53: Page for storing a file associated with a specific model or tool

Figure 54 presents the page where the user (researcher/modeller) stores a parameter associated with a specific model. As shown in figure 54, the user has to select the model (from those models that are already stored in the model/tool repository) with which the parameter is associated, then they have to enter some fields related to the metadata of the parameter (name, description, data type of the parameter, the units in which the parameter is represented, the range of the parameter values, etc.) and then they have to press the “Save parameter” button. The information the user stores through the page presented in figure 54 is related to the `mr_parameter` table of the entity-relationship diagram of model/tool repository (fig. 28).

139.91.210.27/model_app/mrparameter/store

Computational Horizons in Cancer
[Home / Log out](#)

Model/tool associated with this parameter:
Molecular Dynamics of clinical mutations in oncogenic receptors ▼

Name:

Description:

Data type:
number ▼

Unit:

Data range:

Default value:

Is mandatory:
No ▼

Is output:
No ▼

Comment:

Semantic information url:

Figure 54: Page for storing information about a parameter associated with a specific model

As each tool may be associated with a set of references which provide direct or indirect links to additional material, figure 55 presents the page where the user (researcher/modeller) stores a reference associated with a specific model/tool. As shown in figure 55, the user has to select the model (from those models that are already stored in the model/tool repository) with which the reference is associated, then they have to enter some fields related to the metadata of the reference (title, description, type of the resource, creator of the resource, etc.) and then they have to press the “Save reference” button. The information the user stores through the page presented in figure 55, is related to mr_reference table of the entity-relationship diagram of model/tool repository (fig. 28).


139.91.210.27/model_app/mreference/store



[Home](#) / [Log out](#)

Model/tool associated with this reference: Autodock

Title:

Type:

Creator:

Issued:

Bibliographic citation:

Is part of:

Source:

Doi:

Pmid:

Figure 55: Page for storing information about a reference (resource) associated with a specific model/tool

Based on the description of the deliverable “D8.1 Design of the CHIC repositories”, each model tool can have one or more properties that further describes or/and classifies it. In CHIC project there are a set of predefined properties that reflect the 13 model perspectives defined in deliverable “D6.1 Cancer hypomodelling and hypermodelling strategies and initial component models”. The descriptive information of properties (name, description, comment, url representing semantic information about the property) is stored in the entity “mr_property” (see fig. 28). Figure 56 presents the page where the user (researcher/modeller) stores the descriptive information of a property.


139.91.210.27/model_app/mrproperty/store



Computational Horizons in Cancer

[Home](#) / [Log out](#)

Name:

Description:

Type textual description of what this property represents.

Comment:

Type comment for this property.

Semantic information url:

Figure 56: Page for storing a property and its description

Figure 36 presents the page where the user (researcher/modeller) may view, change or delete the basic information (name, description, comment) of the already stored models. Every time the user selects (by choosing its title) an already installed model from the dropdown list (at the top of the page), all the other fields of the page get updated in accordance to this selection. In this way the user may view the basic information of any model already installed in the model/tool repository. Moreover, in the same page, they may update or delete the basic information of the model they selected from the dropdown list by pressing the buttons “Change basic information of the model” and “Delete model” respectively. The user may not insert values in the fields “Created on” and “Modified on” on this page, as this information is inserted automatically by the server.

139.91.210.27/model_app/mrtool/view_update_delete

[Home](#) / [Log out](#)

Title:
Molecular Dynamics of clinical mutations in oncogenic receptors ▼

Description:

All-atom molecular dynamics simulations are performed and analyzed for 30 mutations found in neuroblastoma patients to determine potential of kinase activation.

40ns of MD trajectory is simulated and analyzed for each mutant to determine the oncogenic relevance.

Comment:

Created on: 2014-10-23 13:37:22

Modified on: 2014-11-22 13:20:10

Change basic information of model
Delete model

Figure 57: Page for viewing, changing and deleting the basic information of a selected model

Figure 58 presents the page where the user (researcher/modeller) may view, change or delete the information (name, description, data type, unit, data range, etc.) of the already stored parameters. Every time the user selects (by choosing its name) an already installed parameter from the dropdown list (at the top of the page), all the other fields of the page get updated in accordance to this selection. In this way the user may view the information of any parameter already installed in the model/tool repository. Moreover, in the same page, they may update or delete the information of

the parameter they selected from the dropdown list by pressing the buttons “Change the information of the selected parameter” and “Delete selected parameter”, respectively. The user may not insert values in the fields “Created on” and “Modified on” on this page, as this information is inserted automatically by the server.

139.91.210.27/model_app/mrparameter/view_update_delete

[Home](#) / [Log out](#)

Name:
RDBQT

Description:

(PDB) The Protein Data Bank (pdb) file format is a textual file format describing the three dimensional structures of molecules held in the Protein Data Bank. Describes the three dimensional structures of molecules.

Data type:
file

Unit:
.pdb

Data range:
Type the range of the parameter values seperated by -

Default value:
Type the default value of the parameter

Is mandatory:
No

Is output:
No

Comment:

Type any additional comment concerning this parameter.

Semantic information url:
Type the url which represents semantic information

Created on:
2013-11-20 10:20:21

Modified on:
2013-12-23 13:23:15

Change the information of the selected parameter
Delete selected parameter

Figure 58: Page for viewing, changing and deleting the information of a selected parameter

Figure 59 presents the page where the user (researcher/modeller) may view, change or delete the information (description, comments, url for representing semantic information) of the already stored properties. Every time the user selects (by choosing its name) an already installed property from the

dropdown list (at the top of the page), all the other fields of the page get updated in accordance to this selection. In this way the user may view the information of any property already installed in the model/tool repository. Moreover, in the same page, they may update or delete the information of the property they selected from the dropdown list by pressing the buttons “Change the information of the selected property” and “Delete selected property”, respectively.


139.91.210.27/model_app/mrproperty/view_update_delete



Computational Horizons in Cancer

[Home](#) / [Log out](#)

Name:

TUMOUR-AFFECTED NORMAL TISSUE MODELLING

Description:

Categories: atomic, molecular, cellular, tissue, organ, body system, organism, population, ... combinations

Comment:

Type comment for this property.

Semantic information url:

Type the url which represents semantic information about this property

Change the information of the selected property

Delete selected property

Figure 59: Page for viewing, changing and deleting the information of a selected property

6.7.2 Graphical user interface of the *in silico* trial repository

The graphical user interface for the *in silico* trial repository is based on the use case presented in figure 60. The use case diagram in figure 60 depicts the relationship between the researcher, the modeller, the clinician and the different use cases in which they are involved, concerning the *in silico* trial repository.

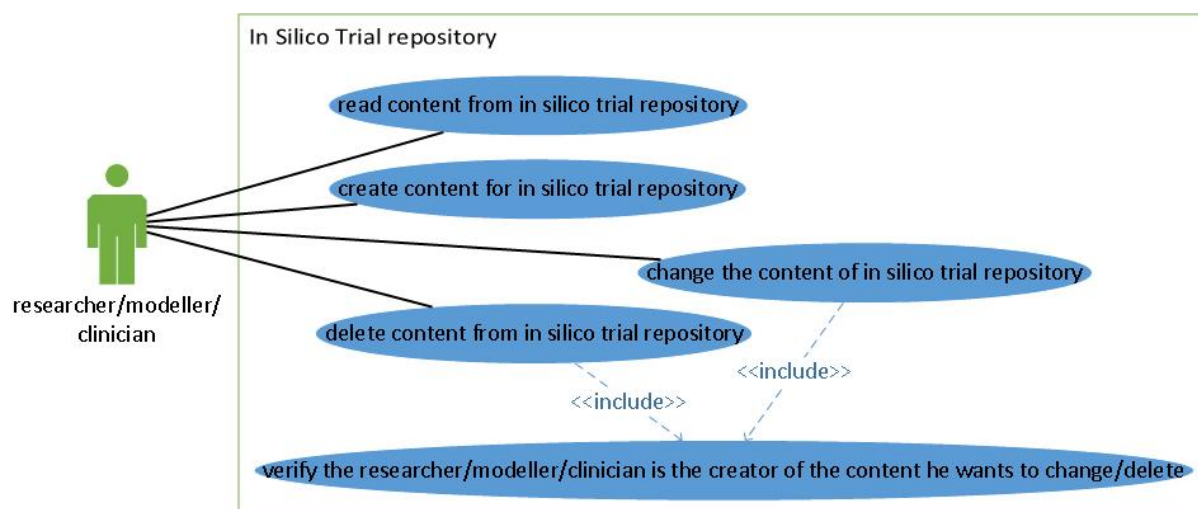



Figure 60: The researcher the modeller and the clinician have read, create, change and delete permissions on *in silico* trial repository

Some screenshots of the user interface for the *in silico* trial repository based on the use case diagram (fig. 60) will be presented in the remaining chapter.

Figure 61 presents the page where the user (researcher, modeller or clinician) stores the information of a trial. As described in deliverable “D8.1 Design of the CHIC repositories”, the *in silico* experiments are organized in *in silico* trials. All *in silico* experiments that are part of the same *in silico* trial use the same *in silico* (hyper)model. Moreover, the (hyper)model that is being used in *in silico* trials is defined in the *in silico* trial entity. As shown in figure 61, in order for the user to store a new trial, they have to store a textual description of the trial, the id of the *in silico* model that is used in the trial, the url where the *in silico* model is located, the *in silico* model that is used as a placebo, the url where the placebo *in silico* model is located, and any additional comment about the trial. The information the user stores through the page presented in figure 61 is related to the “tr_trial” table of the entity-relationship diagram of the *in silico* trial repository (fig. 29).

139.91.210.27/trial_app/trtrial/store



[Home](#) / [Log out](#)

Description:

Type the description of the trial.

Id of the model:

Type the id of the in silico model that is used in the trial.

Url of the model:

Type the url where the in silico model is located.

Model used as a placebo:

Type the in silico model that is used as a placebo.

Url of the placebo model:

Type the url where the placebo in silico model is located.

Comment:

Type any additional comment for the trial.

Save trial

Figure 61: Page for storing information about a trial

Figure 62 presents the page where the user (researcher, modeller or clinician) stores the information of an experiment. As described in deliverable “D8.1 Design of the CHIC repositories”, the *in silico* experiment entity consists of triples of “initial state of the subject” – “*in silico* (hyper)model” – “final state of the subject”. The *in silico* (hyper)model that is used in an *in silico* experiment is not stated in the experiment entity, but in the *in silico* trial entity in which the experiment belongs. As shown in figure 62, in order for the user to store a new experiment, they have to select the (hyper)model used in the experiment, and then they have to enter in the corresponding fields a textual description of the experiment, the id of the subject used as an input, the id of the subject that is produced, the comments related to the experiment, etc. The information the user stores through the page

presented in figure 62 is related to the “tr_experiment” table of the entity-relationship diagram of the *in silico* trial repository (fig. 29).



The screenshot shows a web browser window with the address bar displaying "139.91.210.27/trial_app/trexperiment/store". The page features the CHIC logo and navigation links for "Home" and "Log out". The main form is titled "Description:" and includes several input fields and a dropdown menu. The fields are labeled as follows:


- Description:** A large text area with the placeholder text "Type the description of the in silico experiment."
- Id of subject used as an input:** A text input field with the placeholder text "Type the id of subject that is used as an input to the in silico experiment."
- Id of subject that is produced:** A text input field with the placeholder text "Type the id of subject that is produced after the execution of the in silico experiment."
- Placebo model must be used:** A dropdown menu with "Yes" selected.
- Status of the experiment:** A dropdown menu with "FINISHED SUCCESSFULLY" selected.
- Comment:** A text input field with the placeholder text "Type any additional comments."
- Select the (hyper)model used in the experiment:** A dropdown menu with a dashed line as the selected option.

A "Save experiment" button is located at the bottom left of the form.

Figure 62: Page for storing a new experiment

Figure 63 presents the page where the user (researcher, modeller or clinician) stores the information of a subject. As described in deliverable “D8.1 Design of the CHIC repositories”, the subject may be a person, healthy or not, an animal, etc. The subject may be linked to another data repository, such as the CHIC clinical data repository, a clinical trial management system, a hospital record management system, etc. Every instance of a subject can be accompanied by a set of files. The data of the files are stored internally in a file based repository. As shown in figure 63, in order for the user to store a new subject, they have to enter in the corresponding fields a textual description of the state of the subject, the external id of the subject (this field is used only in the case in which this subject’s state is linked to a subject stored in an external repository of the experiment), the url of the external repository, and any additional comments for the subject. The information the user stores through the page presented in figure 63 is related to “tr_subject” table of the entity-relationship diagram of the *in silico* trial repository (fig. 29).

139.91.210.27/trial_app/trsubject/store


[Home](#) / [Log out](#)

Type a textual description of the state of the subject.

Description:

Type the external id of the subject (only if subject is linked to a subject stored in an external repository).

External id of subject:

Type the url of the external repository (CHIC clinical data repository, e.t.c.).

Url of external repository:


Type any additional comment.


Comment:

Save subject

Figure 63: Page for storing a new subject

Figure 64 presents the page where the user (researcher/modeller/clinician) stores a file which is associated with a specific subject. As shown in figure 64, the user has to select the subject (from these subjects that are already stored in the *in silico* trial repository) with which the file is associated, then they have to enter some fields related to the metadata of the file (title, description, type of file, version, license, comments, etc.) and before the “Submit form” button is pressed the user has to browse the file he/she wants to store. The information the user stores through the page presented in figure 64, is related to the tr_file table of the entity-relationship diagram of the *in silico* trial repository (fig. 29).


139.91.210.27/trial_app/trfile/store



Computational Horizons in Cancer

[Home](#) / [Log out](#)

Select the subject that this file is associated with :

Title:

Description:

Type of file:

Version:


License:


Comment:

File: No file selected.

Figure 64: Page for storing a new file which is associated with a subject

As each trial may be associated with a set of references which provide direct or indirect links to additional material, figure 65 presents the page where the user (researcher/modeller/clinician) stores a reference associated with a specific trial. As shown in figure 65, the user has to select the trial (from those trials that are already stored in the *in silico* trial repository) with which the reference is associated, then they have to enter some fields related to the metadata of the reference (title, type of the resource, creator of the resource, etc.) and then they have to press the “Save reference” button. The information the user stores through the page presented in figure 65 is related to the *tr_reference* table of the entity-relationship diagram of the *in silico* trial repository (fig. 29).


139.91.210.27/trial_app/trreference/store



CHIC
Computational Horizons in Cancer

[Home / Log out](#)

Trial associated with this reference:

Title:

Type:

Creator:

Date of formal issuance:

Bibliographic citation:

Is part of:

Source of the resource:

Doi:

PubMed identifier:

Figure 65: Page for storing information about a reference (resource) associated with a specific trial

Figure 66 presents the page where the user (researcher/modeller/clinician) may view, change or delete the information (description, external id of subject, url of external repository, comments) of the already stored subjects. Every time the user selects (by choosing it's ascending number) an already installed subject (at the top of the page), all the other fields of the page get updated in accordance to this selection. In this way the user may view the information of any subject already installed in the *in silico* trial repository. Moreover, in the same page, they may update or delete the information of the subject they selected by pressing the buttons "Change subject" and "Delete subject", respectively. The user may not insert values in the fields "Created on" and "Modified on" on this page, as this information is inserted automatically by the server.


139.91.210.27/trial_app/subject/view_update_delete



CHIC
Computational Horizons in Cancer

[Home](#) / [Log out](#)

Ascending number of subject:

Type a textual description of the state of the subject.

Description:

External id of subject:
Type the external id of the subject (only if subject is linked to a subject stored in an external repository).

Url of external repository:
Type the url of the external repository (CHIC clinical data repository, e.t.c.).

Comment:
Type any additional comment.

Created on:



Modified on:


Change subject

Delete subject

Figure 66: Page for viewing, changing and deleting the information of a subject

Figure 67 presents the page where the user (researcher/modeller/clinician) may view, change or delete the information (type, title, creator, source of the resource, etc.) of the already stored references. Every time the user selects (by choosing it's ascending number) an already installed reference (at the top of the page), all the other fields of the page get updated in accordance to this selection. In this way the user may view the information of any reference already installed in the *in silico* trial repository. Moreover, in the same page, they may update or delete the information of the reference they selected by pressing the buttons "Change reference" and "Delete reference", respectively. The user may not insert values in the fields "Created on" and "Modified on" on this page, as this information is inserted automatically by the server.



139.91.210.27/trial_app/trreference/view_update_delete



Computational Horizons in Cancer

[Home / Log out](#)

Ascending number of the reference:

Title:

Type:

Creator:

Date of formal issuance:

Bibliographic citation:

Is part of:

Source of the resource:

Doi:

PubMed identifier:

Created on:

Modified on:

Figure 67: Page for viewing, changing and deleting the information of a reference stored in the *in silico* trial repository

7 References

- [1] S. Wimalaratne, P. Grenon, R. Hoehndorf, G. Gkoutos, B. de Bono, An infrastructure for ontology-based information systems in biomedicine: RICORDO case study. *Bioinformatics*. 2012 Feb 1;28(3):448-50.
- [2] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall, 20. October 2004.
- [3] M. Kistler, S. Bonaretti, M. Pfahrer, R. Niklaus, P. Büchler, The Virtual Skeleton Database: An Open Access Repository for Biomedical Research and Collaboration. *J. Med. Internet Res.* 15:e245, 2013.
- [4] C. Rosse, and J. Mejino. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. *J. Biomed. Inform.* 36:478–500, 2003.
- [5] SPARQL, "Query Language for Resource Description Framework (RDF)", [Online]. Available: <http://www.w3.org/TR/rdf-sparql-query>. [Accessed 15 March 2015].
- [6] OpenStack, "Openstack: the Open Source Cloud Operating System", <http://www.openstack.org/software>
- [7] J. McGaw, "Beginning Django E-Commerce", Apress, USA, 2009
- [8] A. Hourieh, *Learning Website Development with Django*, Packt Publishing, UK, 2008
- [9] M. Alchin, "Pro Django", Apress, 2nd ed, USA, 2013

Appendix 1 – Abbreviations and acronyms

<i>SOA</i>	Service Oriented Architecture
<i>HTML</i>	Hypertext Markup Language
<i>CHIC</i>	Computational Horizons in Cancer
<i>HTTP</i>	Hypertext Transfer Protocol
<i>API</i>	Application Programming Interface
<i>HPC</i>	High Performance Computing
<i>VM</i>	Virtual Machine
<i>CSS</i>	Cascading Style Sheets
<i>URL</i>	Universal Resource Locator
<i>XML</i>	Extensible Markup Language
<i>GUI</i>	Graphical User Interface
<i>SSO</i>	Single Sign On
<i>SAML</i>	Security Assertion Markup Language
<i>SP</i>	Service Provider
<i>IdP</i>	Identity Provider
<i>REST</i>	Representational State Transfer
<i>STS</i>	Security Token Service
<i>SOAP</i>	Simple Object Access Protocol
<i>RST</i>	Request Security Token
<i>RSTR</i>	Request Security Token Response
<i>WSGI</i>	Web Server Gateway Interface