A Demonstration of 4D Digital Avatar Infrastructure for Access of Complete Patient Information

Project acronym: MyHealthAvatar

Deliverable No. 5.2
Model and clinical data repositories interfaces & evaluation report

Grant agreement no: 600929

Dissemination Level		
PU	Public	**X**
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)	

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	MyHealthAvatar
Project Full Name:	A Demonstration of 4D Digital Avatar Infrastructure for Access of Complete Patient Information
Deliverable No.:	D5.2
Document name:	Model and clinical data repositories interfaces & evaluation report
Nature (R, P, D, O)[1]	R
Dissemination Level (PU, PP, RE, CO)[2]	PU
Version:	1
Actual Submission Date:	08/03/2016
Editor: Institution: E-Mail:	Nikolaos Christodoulou ICCS – NTUA nikchris@mail.ntua.gr

ABSTRACT:
This deliverable describes the interfaces (Graphical User Interfaces and Application Programming Interfaces/web services) that have been developed for the Tool/Model and Data repositories, described in D5.1. It also presents the results from the evaluation of these repositories.

KEYWORD LIST:
Interface, Graphical User Interface, Application Programming Interface, Web Service, Tool/Model repository, Data repository, Evaluation

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600929.

[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 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**
0.1	27/02/2016	Draft	Nikolaos Christodoulou
0.2	29/02/2016	Revision	Georgios Stamatakos
1.0	08/03/2016	Final	Nikolaos Christodoulou, Georgios Stamatakos

List of contributors

- Nikolaos Christodoulou, ICCS – NTUA
- Georgios Stamatakos, ICCS – NTUA
- Nikolaos Tousert, ICCS – NTUA
- Eleftherios Ouzounoglou, ICCS - NTUA

Contents

1 Executive Summary

Owing to the highly fragmented health systems in European countries, gaining access to a consistent record of individual citizens that involves cross-border activities is very difficult. MyHealthAvatar is a proof of concept for the digital representation of patient health status. It is designed as a lifetime companion for individual citizens that facilitates the collection of, and access to, long-term health-status information. This is extremely valuable for clinical decisions and offers a promising approach to acquire population data to support clinical research, leading to strengthened multidisciplinary research excellence in supporting innovative medical care.

The VPH initiative has led to the collection and integration of predictive models and heterogeneous data to interpret and predict the progress of diseases and the effectiveness of treatments, which have laid down the foundation for new knowledge discovery. The use of these tools can show the quantified progress of the patient and promote a better modulation of treatment and a faster recovery. These are expected to improve the reliability, repeatability and timeliness of medical decisions and interventions. In addition to data access, MyHealthAvatar is also an interface to access integrative models and analysis tools, utilizing resources already created by the VPH community. Overall, it can contribute to individualized disease prediction and prevention and support healthy lifestyles and independent living.

Personalisation is one of the key features of the MyHealthAvatar representation. These avatars are like personal bags of individual citizens. Given a set of long term and consistent health information, data analysis and simulations supporting clinical decisions can be made based entirely on the individualized information from specific patients. This can allow the healthcare system to be tailored for individual care and personalised prevention.

Work package 5, “Models & Repositories”, focuses on the development of clinically oriented repositories that cover the needs of the MyHealthAvatar project.

At the end of the project, the repositories, described in D5.1, will be included into an application. These applications should be able to communicate with physical users and third party applications. To that end, certain interfaces have been developed:

- A Graphical User Interface (GUI), which will allow users to access certain tables of the repository to enter, alter and delete tools/models and their pertinent data
- An Application programming Interface (API), bases on RESTful web services, that will allow the same functionality, remotely, for applications

2 Introduction

2.1 Purpose of this document

This document presents the interfaces that are developed for the WP5 Tool/Model and Data repositories. As they are divided into a Graphical User Interface for physical users and an Application Programming Interface for other applications, for each group, their technical and design details are presented, along with some functionality examples.

Finally, the results of a technical evaluation regarding the completed Tool/Model and Data repositories application are presented.

2.2 Structure of the Deliverable

Chapter 3, briefly describes the application developed by ICCS which includes all the work done in the context of WP5 and the Nephroblastoma Use Case, as it is described in WP9 deliverables. Chapter 4 describes the implementation details of the repositories' Graphical User Interface. Chapter 5 presents the Application Programming Interface by listing its web services' URL's. Finally, in chapter 6, the details about the evaluation of the repositories are given (questionnaire and presentation of results).

3 The IAPETUS Application

3.1 Introduction

IAPETUS is a prototype web application build by the *In Silico* Oncology and *In Silico* Medicine Group, ICCS-NTUA. It is composed of two major modules. A Tool/Model Repository capable of storing simulation models and pertinent/assisting tools, as well as their individual attributes in separate tables, and a module that allows the user to set up and carry out model executions, as well as handle the outcoming results. It also includes the tool execution engine that was described in D5.1. In Figure 1, its high level architectural description, is presented [1].

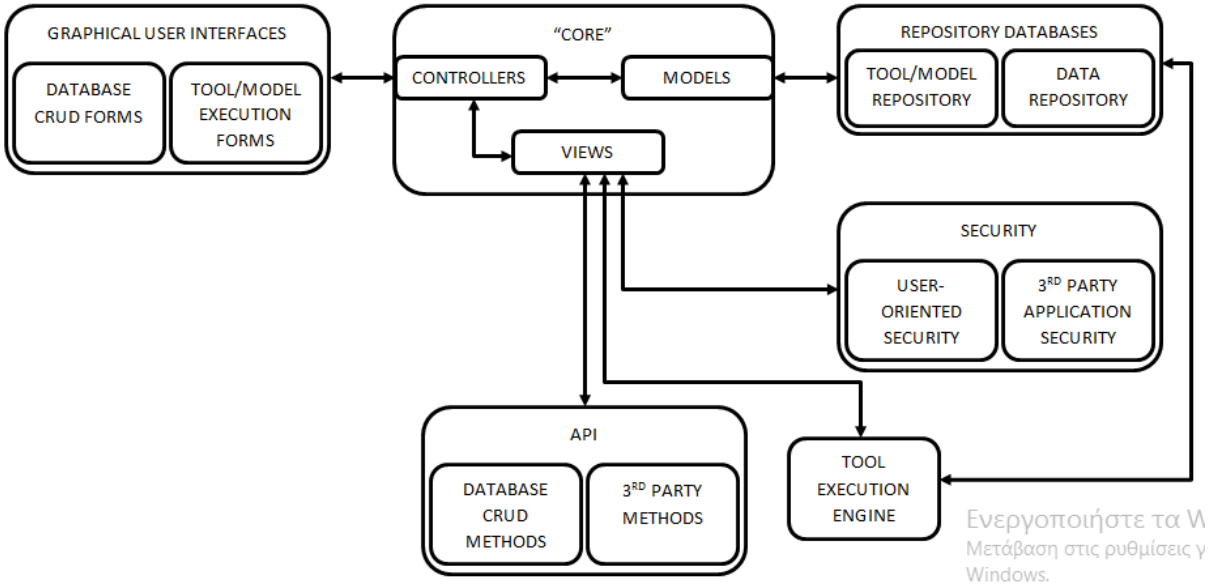


Figure 1: Architectural Diagram of IAPETUS

3.2 General Details

In its current version the entire application, being deployed with all the software tools described in D5.1, is heavily based on python's Django. Therefore, all the major actions in IAPETUS, are managed by views, residing at the application's core. This also includes the handling of the RESTful services of the API, being developed using Tastypie. As for the GUI's, they are based on Django templates, which are the framework's versions of "Controllers" in the Model – View – Controller architectural design.

Finally, in the context of the present deliverable, only the part of IAPETUS that pertains to the Tool/Model and Data repositories will be explained and presented. The Oncosimulator module and its functions is presented in D9.4

4 Graphical User Interfaces (GUI's)

4.1 Introduction

The Tool/Model and Data repositories Graphical User Interfaces correspond to the "Controller" part of the Model-View-Controller design architecture that is followed in the development of the repositories and eventually, IAPETUS. In the Django Framework the controllers are called "templates" and are usually implemented as html pages, which communicate with the views via the http request-response mechanism. As a result, they support basic http functions such as GET and POST. Code-wise, these pages can include everything else that is normally included in a regular html page (html tags, css/javascript scripts, etc). However, in order to tailor and link them to a repository's views, certain statements must be used that belong to the Django Template language. These statements can control and change the content of a template, depending on a set of arguments that is passed to the template by the calling view in the latter's "response" object.

4.2 The Django template language

The Django template language is what defines templates. If a text (simple, html, etc.) or even a python string is marked-up with elements of this language, they can be called templates. This language is based on the following simple elements:

- **Block tag**: A block tag is a symbol within a template that does something. It can output content, serve as a control structure (an "if" statement or "for" loop), grab content from a database or enable access to other template tags. Block tags are surrounded by "{%" and "%}".
  Example template with block tags:
  {% if is_logged_in %}
  Thanks for logging in!
  {% else %}
  Please log in.
  {% endif %}

- **Variable**: A variable is a symbol within a template that outputs a value. Variable tags are surrounded by "{{" and "}}".
  Example template with variables:
  My first name is {{ first_name }}. My last name is {{ last_name }}.

- **Context:** A context is a "variable name" -> "variable value" mapping that is passed to a template. A template renders a context by replacing the variable "holes" with values from the context and executing all block tags. In the case of html pages, the context is placed within the http "response" object

4.3 Tool/Model and data repository templates

A “flowchart” through the various views for the Tool/Model and data repository is given in figure 2. These views have grouped together functions on multiple database tables. The templates take this approach one step further and utilize the same template for multiple views.

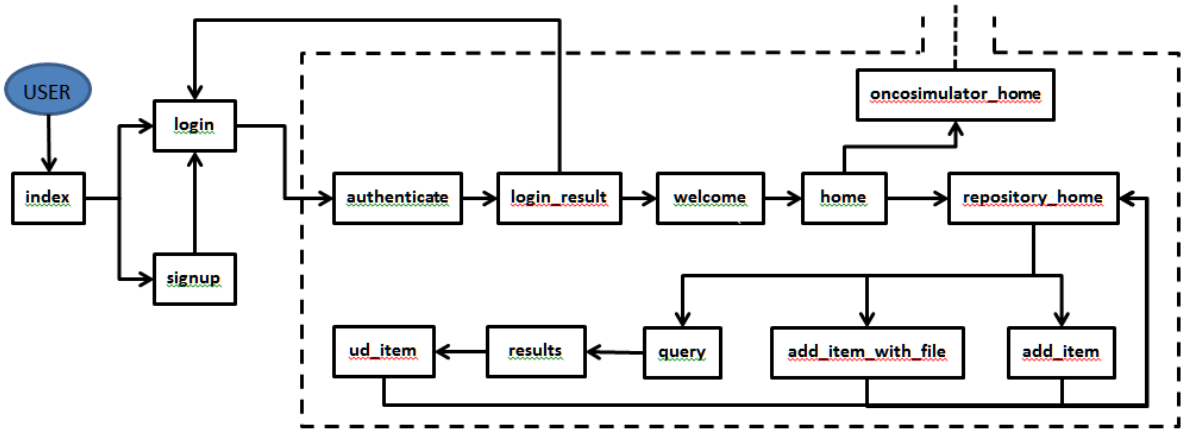


Figure 2: Tool/Model and data repository view “workflow”. The “oncosimulator_home” view is not a part of the repositories module of IAPETUS, however it is added for completion reasons. The dashed line represents the “area” in which only a logged user may access. This includes the Oncosimulator module of IAPETUS.

In the following sub-sections, each template will be described, regarding the views it manages and pertinent screenshots with the end results will be presented

4.3.1 index.html

This template interacts only with the “index” view. It presents the initial screen of IAPETUS where the user is prompted to either sign up for a new account, or login using their own.

Figure 3: index.html (IAPETUS initial screen)

4.3.2 login.html

This template interacts only with the “login” view. It contains a “username” and a “password” field. The user must enter their credentials, in order to enter the repository.

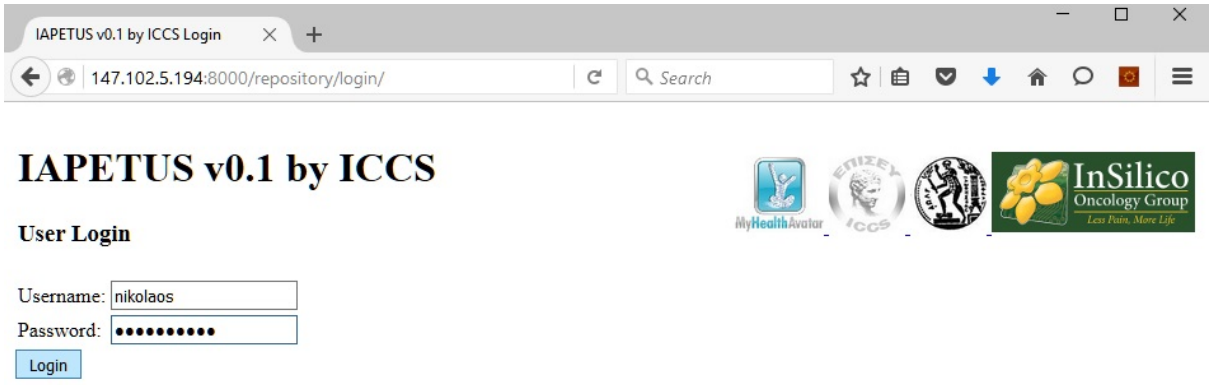


Figure 4: login.html

4.3.3 userAuthFunctions.html

This template interacts with almost all views, but mostly with the “login_result” view. After a user trying to log in, the set of credentials they entered will be checked across the user data holding table of the repository. Depending on the result, a set of arguments is passed to the template. These arguments contain the messages that inform the user about the success or failure of his attempt and also give them links to redirect them either back to the login screen (if they entered false credentials) or to IAPETUS home screen (if they provided correct credentials

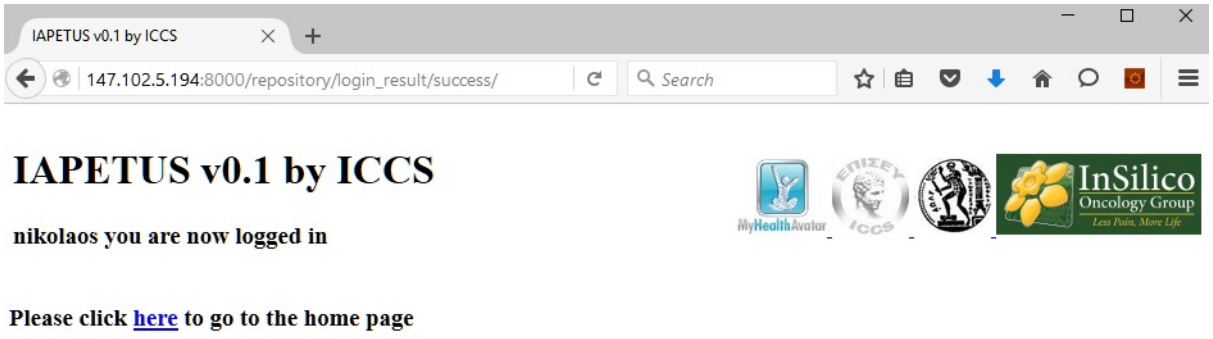


Figure 5: userAuthFunctions.html after successful login

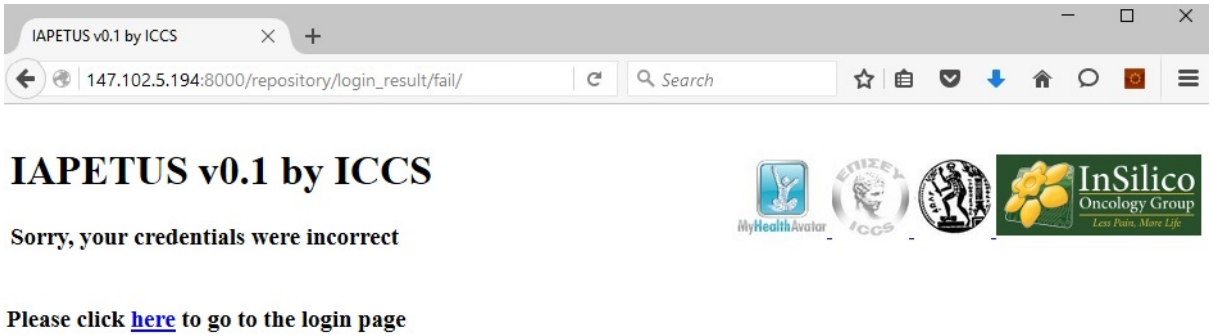


Figure 6: userAuthFunctions.html after unsuccessful login

The same template is used to prevent users from unauthorized access to the other functions of IAPETUS. Should anyone try to enter a template by e.g. entering its URL directly to the browser, userAuthFunctions.html will be returned instead, prompting them to login.

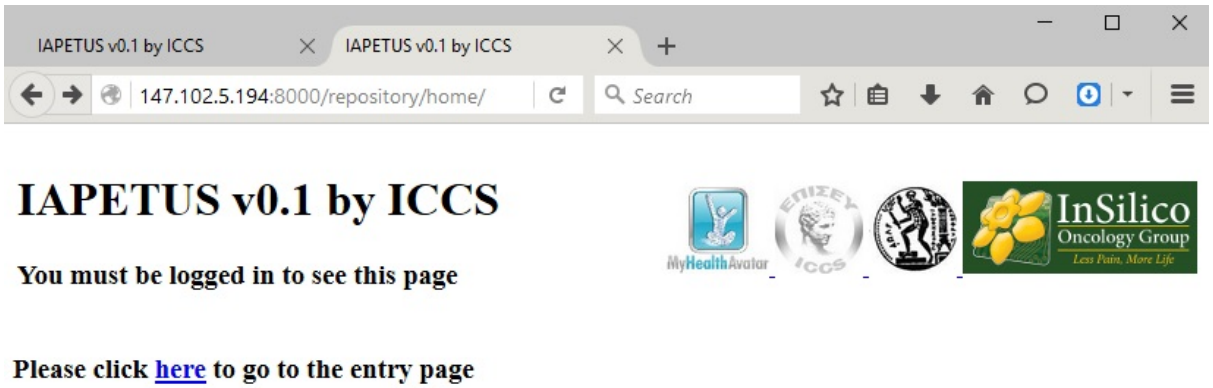


Figure 7: userAuthFunctions.html after preventing unauthorized access

4.3.4 home.html

This template is associated with the “home” view. It contains IAPETUS home screen, giving the user the choice to either enter the Tool/Model Repository or the Oncosimulator module.

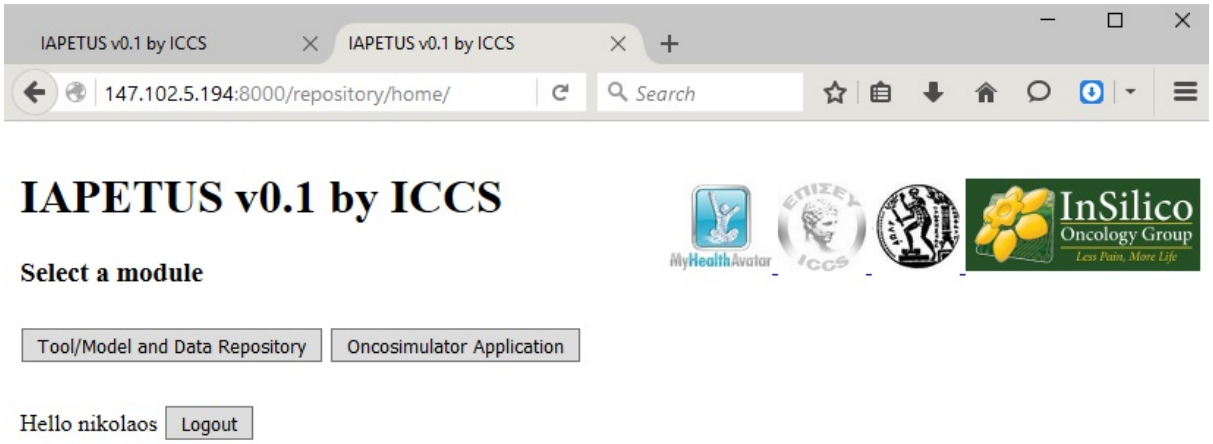


Figure 8: home.html (IAPETUS home page)

4.3.5 repositoryHome.html

This template is associated with the "repository_home" view. It serves as the home page for the repository module of IAPETUS and contains a series of buttons for accessing the templates that allow entering, editing, or deleting data from the repository.

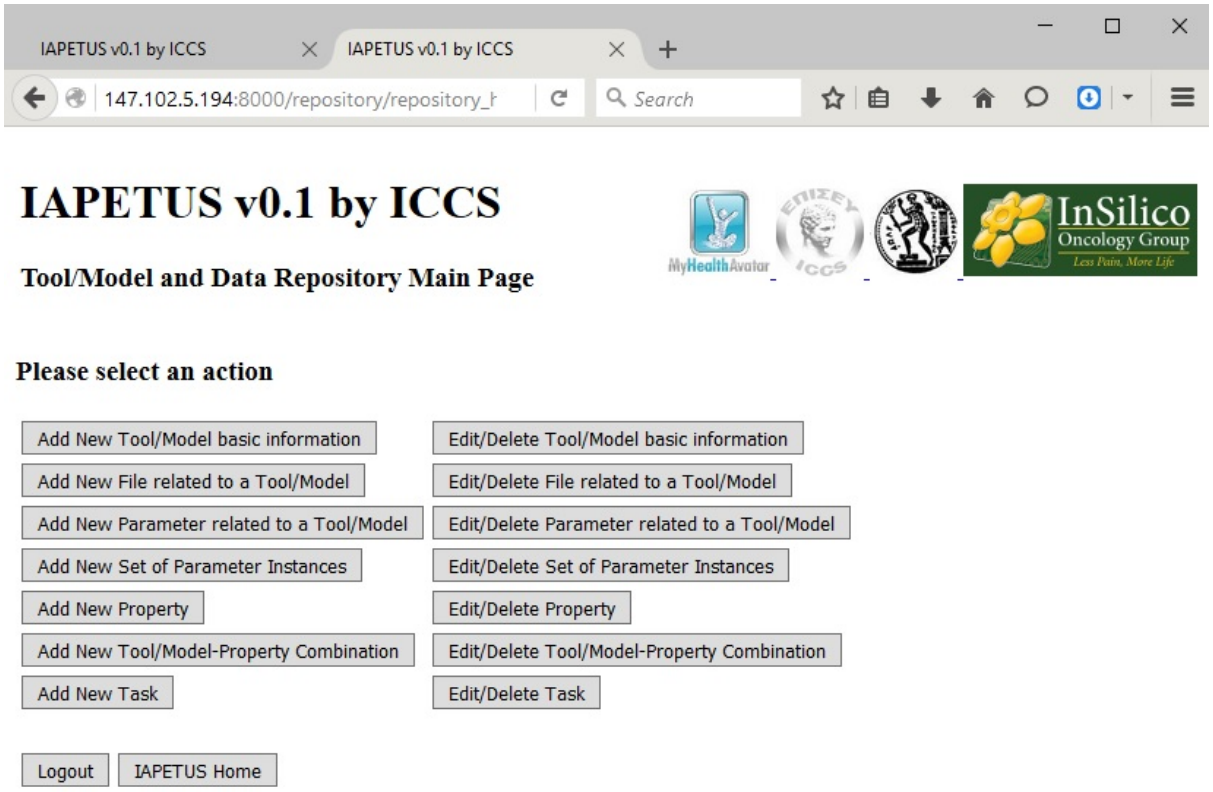


Figure 9: repositoryHome.html (IAPETUS repository module page)

4.3.6 crudFunctions.html

This is the most functional template. It controls the “add_item”, “add_item_with_file”, “ud_item” and “query” views. As an “item” we are referring to any entity that can be stored to the repository by the user (Tool/Model, File, Parameter, InstanceParameterSet, Property, Tool – Property Combination, Task) The first three views group together all insertion actions depending on whether the user is also uploading a file to the repository or not and the editing/deleting of stored data. The last view is responsible for accepting queries pertaining to stored items, based on the latter’s names, and returns them to the user. Therefore, this template is involved with all the database transactions.

Django allows the developer to “map” the database tables into models and these models into one or many forms depending on what portion of the model is for the user’s eyes [2]. Templates are using views to render forms to their html code [3]. The content is loaded into an html <form> tag and associated with a view. In this case, this template has two <form> tags, one for the CRUD functionalities and one containing the “query” screen. An “if” block determines what to show, based on a flag, passed as argument to the template. If the first choice is made (CRUD), then the user, depending on the button they clicked in the “repositoryHome” screen, which sends the proper argument to the one of the three first CRUD views, will see in this template a set of fields that corresponds to the proper form. This is better explained in the following two figures, which handle the data insertion for the tool/model basic information and files.

Figure 10: crudFunctions.html rendering the form for adding a new Tool/Model

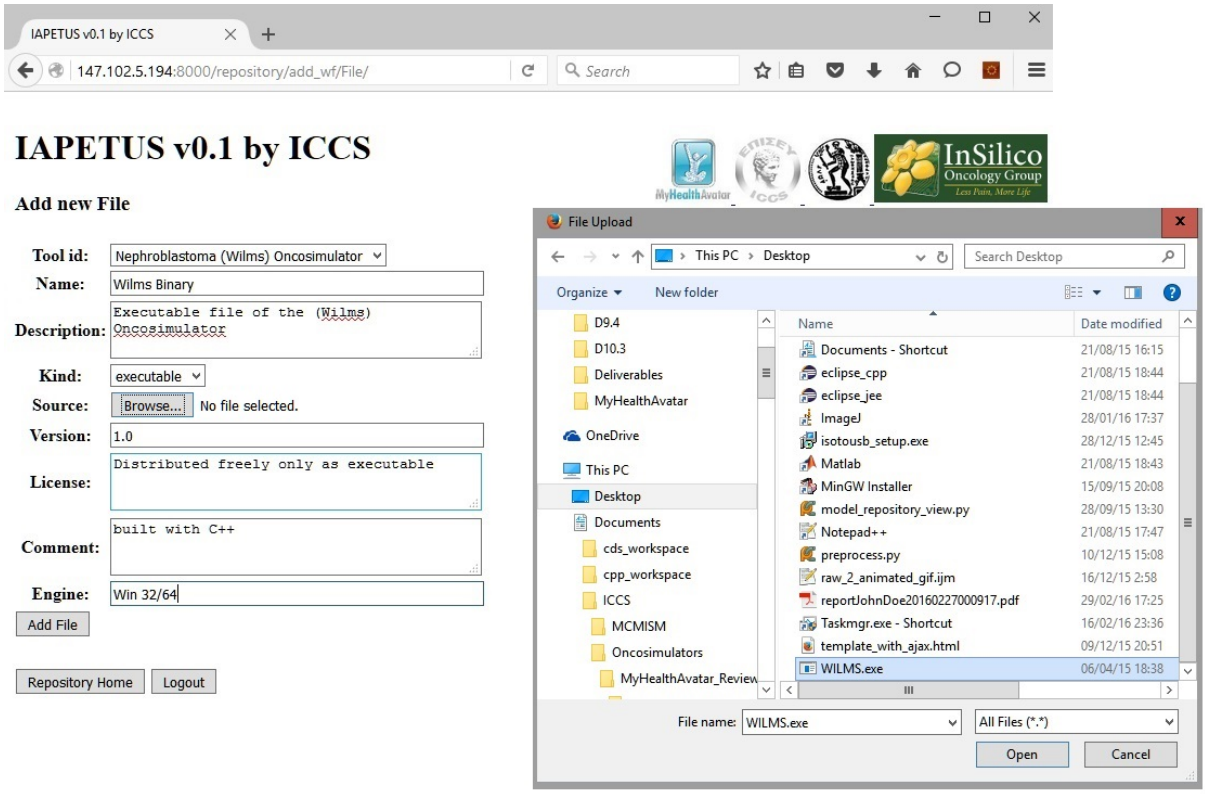


Figure 11: crudFunctions.html rendering the form for adding a new File

If the user has clicked on a “Edit/Delete” button, then the other html <form> tag will be called and the template will load up the fields for querying. The screen is the same for all items that the user is allowed to perform CRUD functions on. The only thing that changes is the name of the item (tool, file, etc.):

Figure 12: crudFunctions.html searching for a File.

After the user submits the query and clicks on a returned result, (as it will be shown in the next template), a new form is rendered by the template that contains the information that the user can see and/or edit. In the same form, a button is also presented, the pressing of which, deletes the entry which is shown:

Figure 13: crudFunctions.html rendering the form for editing or deleting an existing Tool/Model

If the entry contains a file to download, a pertinent button also appears:

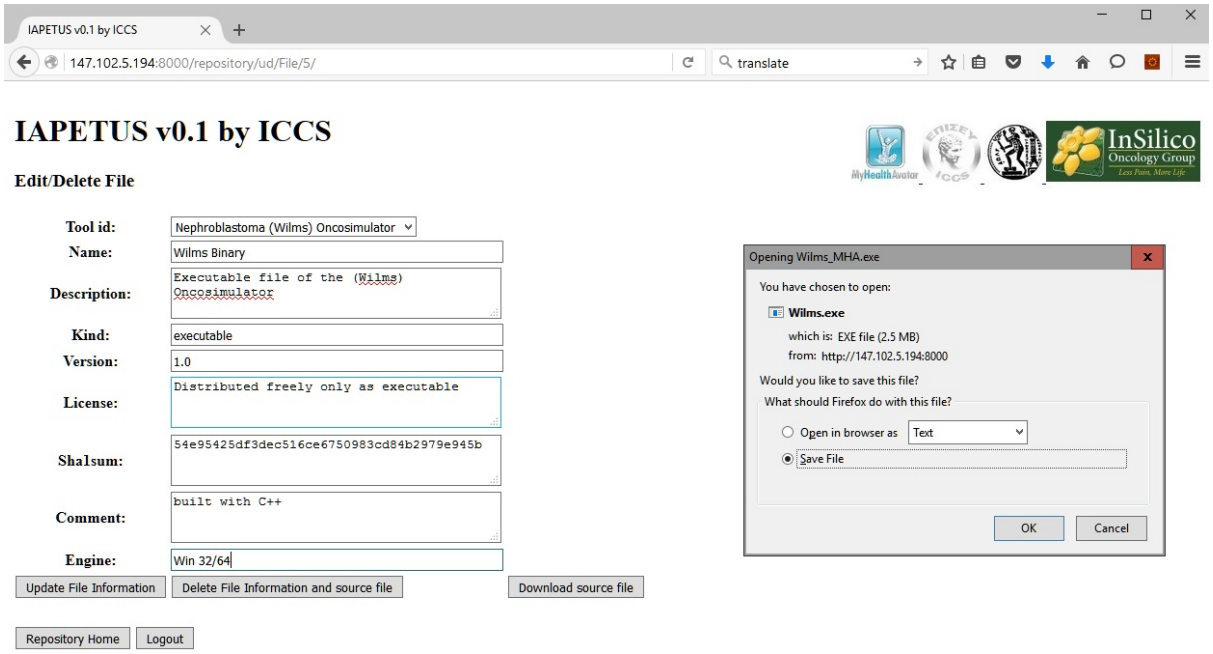


Figure 14: crudFunctions.html rendering the form for editing information, downloading or deleting an existing File

The following Figures will give the template's rendering with respect to the other items CRUD functions:

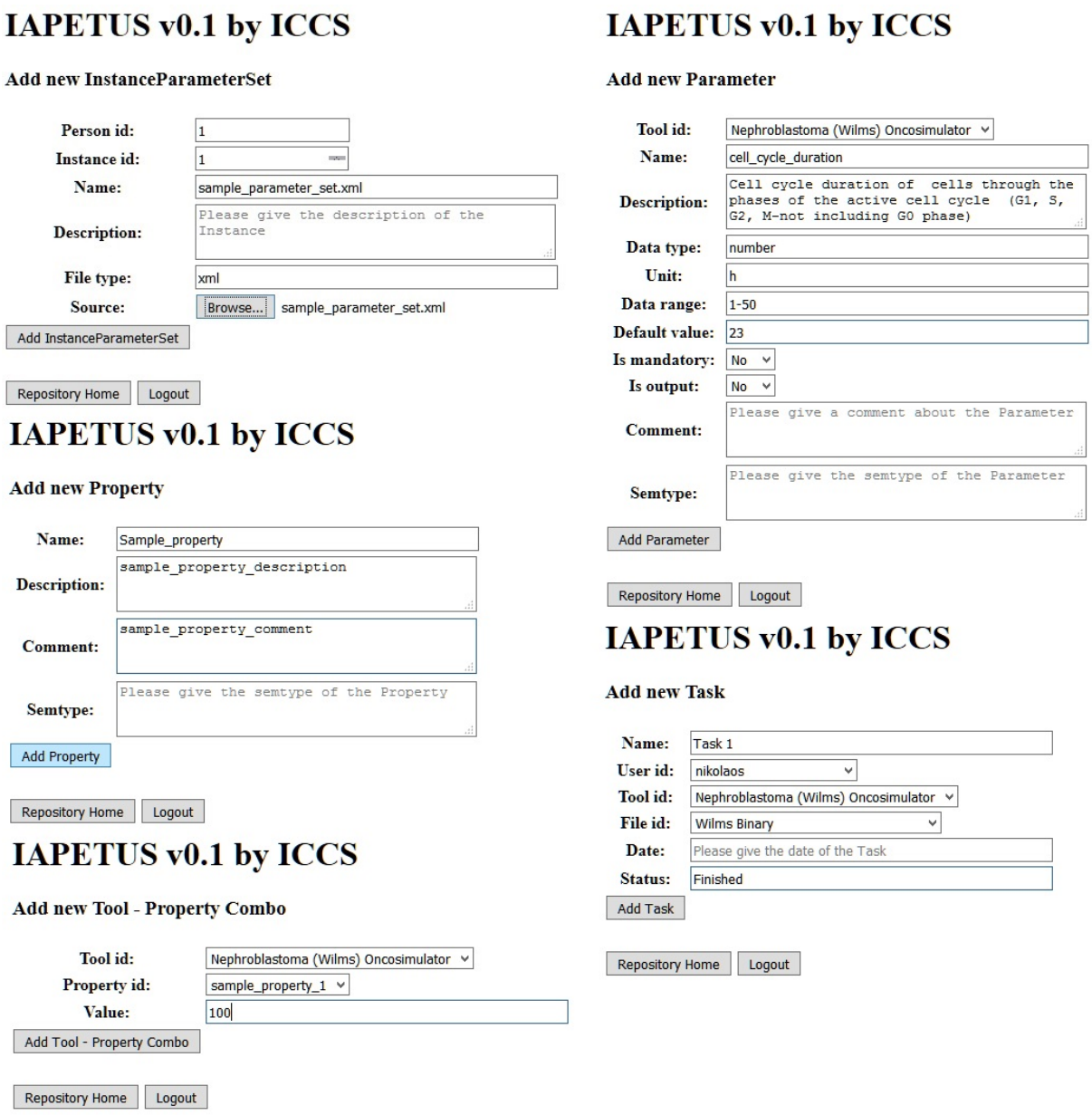


Figure 15: crudFunctions.html rendering forms for adding all remaining Items

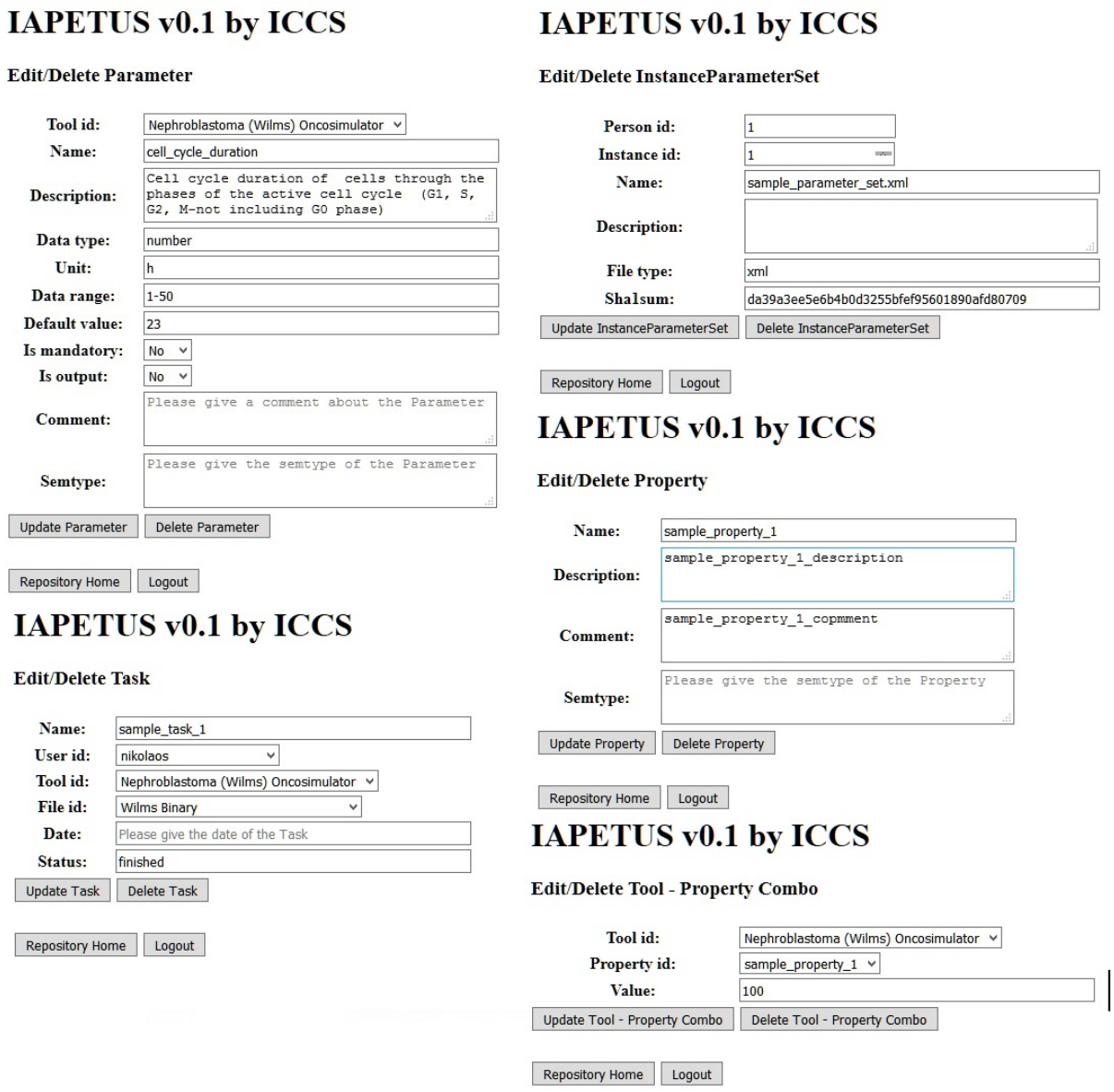


Figure 16: crudFunctions.html rendering forms for editing or deleting all remaining Items.

4.3.7 results.html

This template is linked to the "results" view. When a user gives a query about an item (as explained in the previous template), this is the screen that presents the results. For each retrieved result, a button with the entry's ID is presented. If the user clicks on that, they are redirected to the proper Edit/Delete form, to view, alter, download, or delete the retrieved data. If there is no data, the user is prompted to try again by entering a different name to query

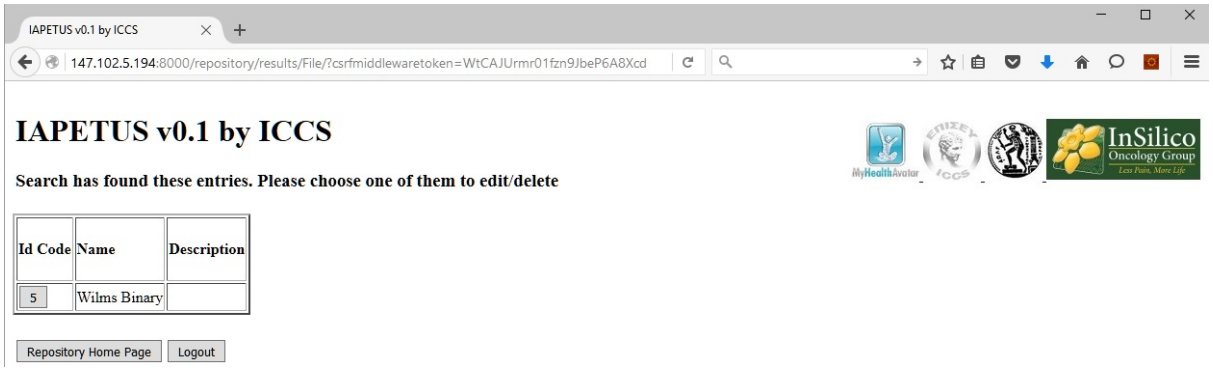


Figure 17: results.html with query results

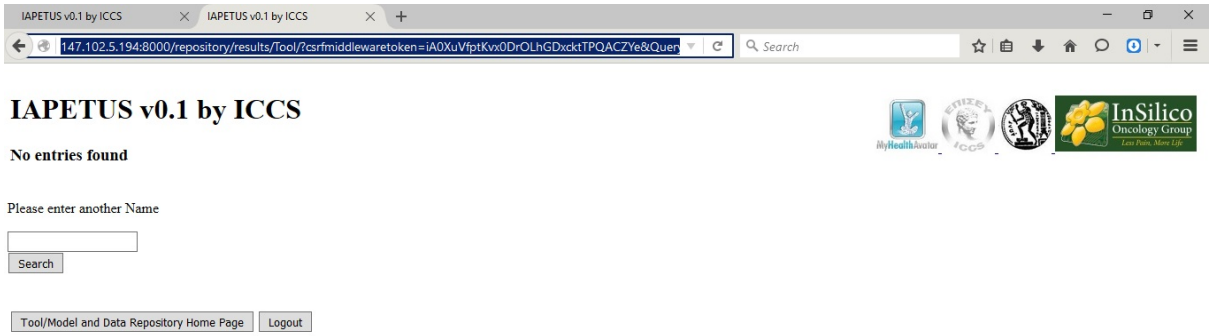


Figure 18: results.html with no results

5 Application Programming Interface (API)

5.1 Introduction

As it was described in D5.1, the framework that was used to develop IAPETUS's API, was Tastypie [4], a python framework that collaborates with Django in order to publish information stored in database tables as JSON via RESTful web services. To that end, the Django models are used to form "resource" objects by utilizing the results of certain queries against the chosen models, which can be further tailored by applying additional filters. These resources are the blueprints for creating the JSON representations. These services also allow the user to add, alter or delete data from the base, or make queries via URLs

In the case of IAPETUS, eight tables were chosen to publish their data via JSON. Seven of them pertain to the "items" described in the section 4.3.6. The eight table is the one that holds the registered users. However, this resources is only limited to showing the first name, last name, last login timestamp and the username of a user. Moreover, this services is limited to administrators only.

5.2 Web services

5.2.1 Basic generic URL form

The generic URL for each web service is currently: http://147.102.5.194:8000/repository/tool_app_api/X/Y/Z/?format=json where:

X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property}. It refers to a specific table, by using the registered name of its corresponding resource. If X is not given, then Y and Z must also not be given, otherwise the request will return a http 404 error (page no found)

Y = {"schema", "set", 1, 2, 3, … n, n∈ ℕ }. This variable refers to the row or rows that are requested. Depending on the value, one of the following things can happen:

- If a number is given, then the JSON data referring to the specific table row are returned. If no entry is found, then an empty string is returned.
- If the word "schema" is given, then the response will contain information about the table's fields and the list of allowed http methods.
- If no argument is given then all the rows will be returned to the user.
- If the word "set" is given, then the user must utilize the Z variable, in order to specify the beginning and the end of the requested set of rows.

Z = {“download”, a;b;c;… : a,b>0 }. This variable is usable only when Y=”set”, or if X is a resource that is linked to a model that serves files and Y is a unique row ID. Otherwise the use of it in any URL will cause a “page not found” error (http 404).

- If the value is “download”, then the file, whose source is located at the path that is contained in the stores table data, is served for download.
- If a set of numbers is given, then for each of the numbers, representing unique row ID’s, the system will either fetch the stored data in JSON format, or will return a JSON entrly "not_found": ["x_1", "x_2", …, "x_n"] where x_1, x_2, …, x_n are numbers that do not correspond to row entries.

5.2.2 List of web services

tool_app_api/		
Description	This method returns all the information about the API’s resources	
URL	http://147.102.5.194:8000/repository/tool_app_api	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
Parameters	?format=json (required)	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no oAuth token inside HTTP header	
	403 http status code if oAuth token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: OAuth <OAuth2.0 access token>
Json Response		
A list of JSON objects, one for each different API resource that has the resource’s name as a key and a nested JSON object as value. The nested JSON object contains key-value pairs for the URL that returns the entire list of data and the resource’s schema		

Table 1: Description of the tool_app_api/ service

tool_app_api/X/	
Variable	X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property}
Description	This method returns the entire list of the contents of the table which corresponds to the resource’s name given as value of X
URL	http://147.102.5.194:8000/repository/tool_app_api/X
Encoding	application/x-www-form-urlencoded

HTTP Method	GET	
Parameters	?format=json (required)	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no oAuth token inside HTTP header	
	403 http status code if oAuth token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: OAuth <OAuth2.0 access token>
Json Response		
A list of JSON objects, one for each table row. Each object contains a set of key-value pairs. Each key is the table column name, and each value is the value stored into the repository. An additional pair is included in that set which pertains to the object's unique URL within the API		

Table 2: Description of the tool_app_api/X/ service

tool_app_api/X/schema/	
Variable	X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property}
Description	This method returns the schema of the table which corresponds to the resource's name given as value of X
URL	http://147.102.5.194:8000/repository/tool_app_api/X/schema
Encoding	application/x-www-form-urlencoded
HTTP Method	GET
Parameters	?format=json (required)
Returns	200 OK & JSON object
	400 http status code if bad request
	401 http status code if no oAuth token inside HTTP header
	403 http status code if oAuth token not verified
	500 http status code if internal server error
HTTP Header	Name: Authorization / Value: OAuth <OAuth2.0 access token>
Json Response	
A list of JSON objects, explaining the resource's allowed http methods for individual rows and the table as a whole, default format, default limit of rows that are shown as results, the corresponding table's fields and the fields that can be used as query arguments. The objetc about the fields, contains a set of nested JSON objcets, that have the table column's name as key and a JSON object with all the column's properties (e.g. nullable, blank field, readonly, etc.) as a value.	

Table 3: Description of the tool_app_api/X/schema/ service

tool_app_api/X/Y/		
Variables	X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property} Y = (1, 2, 3, … n, n∈ ℕ }.	
Description	This method returns the stored data of the row with ID nuber equal to Y, which belongs to the table that corresponds to the resource's name given as value of X	
URL	http://147.102.5.194:8000/repository/tool_app_api/X/Y	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
Parameters	?format=json (required)	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no oAuth token inside HTTP header	
	403 http status code if oAuth token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: OAuth <OAuth2.0 access token>
Json Response		
If the value of the Y variable exists in the table as a row ID, then the responce will contain a set of key-value pairs, that show the column names and the column values for that specific row, along with the row's contained object URL within the API. If the value of Y does not exist in the database table, then the response is an empty string.		

Table 4: Description of the tool_app_api/X/Y/ service

tool_app_api/X/Y/download	
Variables	X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property} Y = {1, 2, 3, … n, n∈ ℕ }.
Description	This method serves for download the file, whos path is contained in the row with ID nuber equal to Y, which belongs to the table that corresponds to the resource's name given as value of X
URL	http://147.102.5.194:8000/repository/tool_app_api/X/Y/download
Encoding	application/x-www-form-urlencoded
HTTP Method	GET
Parameters	?format=json (required)
Returns	200 OK & JSON object
	400 http status code if bad request

	401 http status code if no oAuth token inside HTTP header	
	403 http status code if oAuth token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: OAuth <OAuth2.0 access token>
Response		
A file is served for download to the user's computer		

Table 5: Description of the tool_app_api/X/Y/download service

tool_app_api/X/set/Z		
Variables	X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property} Z = $\{z_1;z_2;...;z_n, z_1,z_2,...,z_n \in \mathbb{N}\}$.	
Description	This method returns a set of the stored data for the rows whose IDs are in the list that is the value of Z. These rows belongs to the table that corresponds to the resource's name given as value of X	
URL	http://147.102.5.194:8000/repository/tool_app_api/X/Y	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
Parameters	?format=json (required)	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no oAuth token inside HTTP header	
	403 http status code if oAuth token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: OAuth <OAuth2.0 access token>
Json Response		
Contains 2 key-value pairs: "not_found" and "objects" For each of the values in the set of numbers given as the value of Z: • If the row with that ID exists in the table, a JSON object that contains a set of key-value pairs, that show the column names and the column values for that specific row, along with the row's contained object URL within the API is added to the object that acts as the "objects" value. • If the row with that ID does not exist, then the number is added to the list that acts as the "not-found" value		

Table 6: Description of the tool_app_api/X/Y/ service

5.2.3 Making queries through foreign key relations

Certain resources, just like the models and, by extension, the database tables they are linked to, have some foreign key relations to other resources. These relations can be expressed in the web service URLs with an extra parameter that has the following format:

?A__B=C (1)

where:

- A is the name of the foreign key field of resource that we are requesting data from
- B is the name of a field in the related table
- C is the queried value of B.

With that in mind, the following table of the variables X,A is given below, that effectively demonstrates all foreign key relations. It must be noted that this parameter can be used with the tool_app_api/X service.

Value of X	Values of A
file	tool_id
parameter	tool_id
task	user_id, tool_id, file_id
tool_property	tool_id, property_id

Table 7: Foreign key relations between resources, that are used to formulate statements as in (1)

Example: http://147.102.5.194:8000/repository/tool_app_api/file/?format=json&tool_id__name=Nephroblastoma%20%28Wilms%29%20Oncosimulator has X="file", A = "tool_id", B = "name" and C = "Nephroblastoma (Wilms) Oncosimulator". This means that this URL will return a JSON object containing information about all the files that refer to the tool/model named "Nephroblastoma (Wilms) Oncosimulator"

5.2.4 Inserting, deleting or altering data

In order to communicate with IAPETUS for remotely inserting, altering, or deleting data, URLs of the form http://147.102.5.194:8000/repository/tool_app_api/X or http://147.102.5.194:8000/repository/tool_app_api/X/Y must be used by the third party application.

An http request must be made that will contain in its header, the OAuth access token, the http method that the request wants to be executed (POST, PUT, PATCH, DELETE) and the actual data to be uploaded in JSON format. If a file is to be uploaded, its path should be included in the JSON data, but an additional header element is needed.

Sending data with POST		
Variables	X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property}	
Description	This method sends a JSON object of data and they are inserted the table that corresponds to the resource's name given as value of X	
URL	http://147.102.5.194:8000/repository/tool_app_api/X	
Encoding	application/json	
HTTP Method	POST	
Returns	201 CREATED	
	400 http status code if bad request	
	401 http status code if no oAuth token inside HTTP header	
	403 http status code if oAuth token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: OAuth <OAuth2.0 access token>
Json Data Object		
It must contain a set of key-value pairs. The keys must be the names of the fields of the model that is linked to resource whose name it the value of X. The values must contain the actual data to be places into the repository, to create a new entry		

Table 8: Description of the web service for POST

Updating data with PUT		
Variables	X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property} Y = {1, 2, 3, … n, n∈ ℕ }.	
Description	This method sends a JSON object of data to update the row, the ID of which is equal to the value of Y, that resides within the table that corresponds to the resource's name given as value of X	
URL	http://147.102.5.194:8000/repository/tool_app_api/X/Y	
Encoding	application/json	
HTTP Method	PUT	
Returns	204 NO CONTENT if update is successful	
	400 http status code if bad request	
	401 http status code if no oAuth token inside HTTP header	
	403 http status code if oAuth token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: OAuth <OAuth2.0

		access token>
Json Data Object		
It must contain a set of key-value pairs. The keys must be the names of the fields of the model that is linked to resource whose name it the value of X. The values must contain the actual data to be placed into the repository, to replace the existing entry.		

Table 9: Description of the web service for PUT

Partial entry update with PATCH		
Variables	X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property} Y = {1, 2, 3, … n, n∈ ℕ }.	
Description	This method sends a JSON object of data to update part of a row, the ID of which is equal to the value of Y, that resides within the table that corresponds to the resource's name given as value of X	
URL	http://147.102.5.194:8000/repository/tool_app_api/X/Y	
Encoding	application/json	
HTTP Method	PUT	
Returns	202 ACCEPTED if update is successful	
	400 http status code if bad request	
	401 http status code if no oAuth token inside HTTP header	
	403 http status code if oAuth token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: OAuth <OAuth2.0 access token>
Json Data Object		
It must contain a set of key-value pairs. The keys must be the names of the fields of the model that is linked to resource whose name it the value of X. The values must contain the actual data to be placed into the repository, to replace the existing entry.		

Table 10: Description of the web service for PATCH

Deleting an entry with DELETE	
Variables	X = {user, tool, file, instance_parameter_set, parameter, property, task, tool_property} Y = {1, 2, 3, … n, n∈ ℕ }.
Description	This method deletes a row, the ID of which is equal to the value of Y, that resides within the table that corresponds to the resource's name given as value of X
URL	http://147.102.5.194:8000/repository/tool_app_api/X/Y

Encoding	application/json	
HTTP Method	PUT	
Returns	204 NO CONTENT if update is successful	
	400 http status code if bad request	
	401 http status code if no oAuth token inside HTTP header	
	410 GONE if the deleted entry is re-requested	
	403 http status code if oAuth token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: OAuth <OAuth2.0 access token>
Json Data Object		
It must contain a set of key-value pairs. The keys must be the names of the fields of the model that is linked to resource whose name it the value of X. The values must contain the actual data to be placed into the repository, to replace the existing entry.		

Table 11: Description of the web service for DELETE

5.2.5 OAuth 2.0

The aforementioned web services need an OAuth 2.0 access token to operate. So, before an application calls a part of IAPETUS's API, they must first call the following web service to receive a limited access token, which must include in their request headers.

tool_app_api/oauth2/access_token	
Description	This method provides an access token to the requesting party, which is used
URL	http://147.102.5.194:8000/repository/tool_app_api/oauth2/access_token
Encoding	application/json
HTTP Method	POST
Returns	200 & JSON object
	400 http status code if bad request
	500 http status code if internal server error
POST data	
The sending data muct contain five variables: the client's ID, the client's secred ID, the grant type of the requested data, and the client's username and passowrd	
Json Data Object	
This object contains the access token, its scope (read, write, etc.), the time that the token is valid and a refresh token.	

Table 12: Description of the tool_app_api/oauth2/access_token

6 Evaluation results

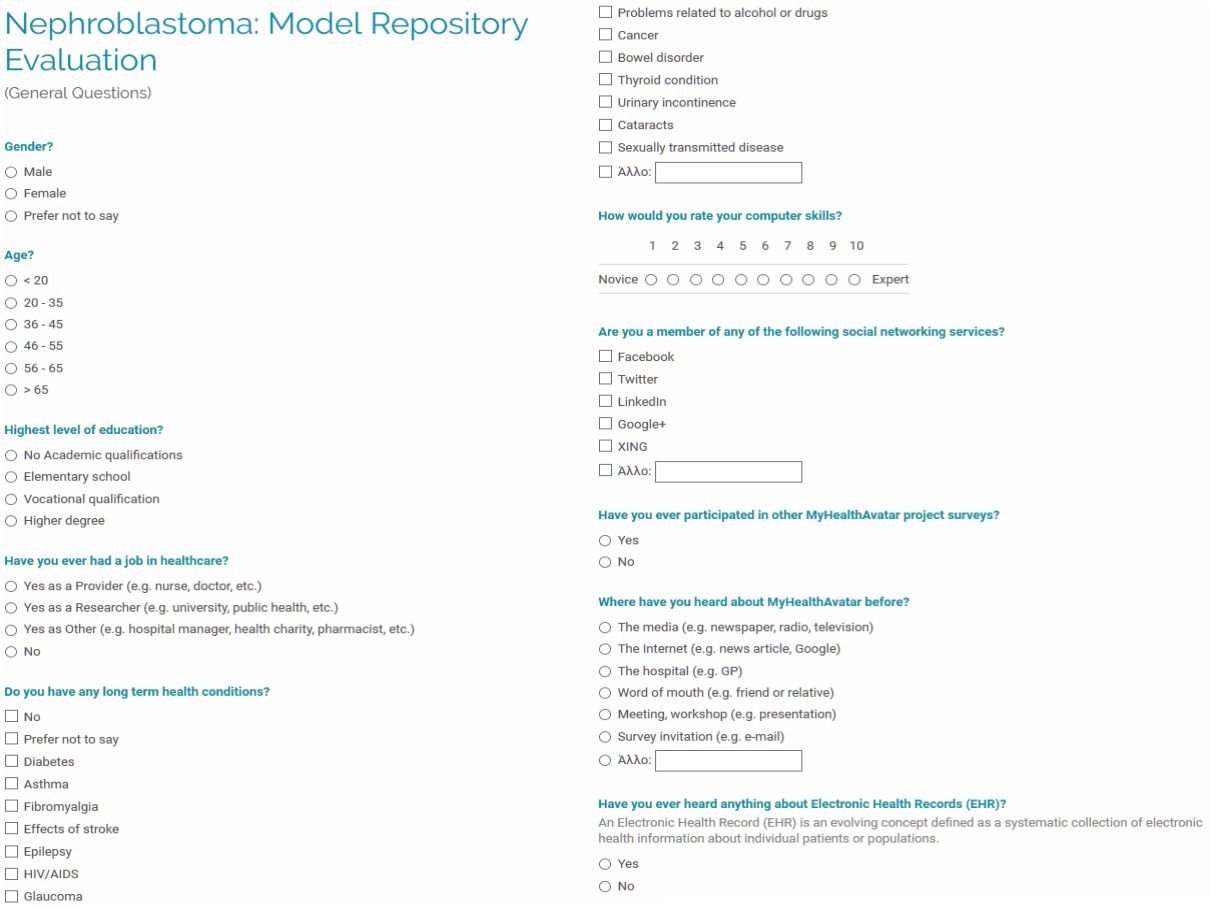
Near the end of the project, a number of workshops were conducted by ICCS (described in deliverables 9.4 and 10.3). In these workshops, the entire work done in the context of the project was presented and IAPETUS was fully demonstrated. Then, the participants were given instructions in the form of a small documentation for IAPETUS and they were informed to visit http://www.ehealthserver.com/mha/ where questionnaires about the repositories and the Nephroblastoma use case were uploaded. In this section, we will only focus on the repositories questionnaire, which was answered by 21 people.

WEB APPLICATION – TOOL/MODEL AND DATA REPOSITORIES		Rating (1 low, 5 high)				
	(software quality characteristics)	1	2	3	4	5
Functionality	Can this web application store models?					
	Can this web application store model attributes? (parameters, etc.)					
	Can this web application store data to use with the models?					
	Can this web application search and present stored data?					
	Can this web application retrieve data in files?					
	Can this web application alter its stored data?					
Efficiency	How quickly does the repository respond to the user requests?					
	Is the application comprehensible?					
	Is support of a technical person needed in order to use this application?					
Compatibility	Do you know other similar application? If yes, is this tool better than the other you know?					
Usability	Can you comprehend the application's functionalities?					
	Can you learn to use the application easily?					
	Can you use the application without much effort?					
	Does the interface look good?					
	Does the interface provide all required information?					
Reliability	Have most of the faults in the software been eliminated over time?					
	Is the software capable of handling errors?					
	Can the services resume working & restore lost data after failure?					
Portability	Can the web application be easily accessed from any pc?					
Security	Are data accessible only to authorized users?					
	Do you think the uploaded data are secure?					
	Does the system prevent unauthorized access?					

Maintainability	Can the software be tested easily?					
Quality in use	How accurate and complete is the software for the intended use?					
	Does the software improve the time or reduce resource for the intended goal?					
	Does the software satisfy the perceived achievement of pragmatic goals					
	Can the software harm people in the intended contexts of use ?					

Table 13: Tool/Model and Data repository evaluation questionnaire.

In the beginning of the questionnaire, there was also a form with general questions:



Nephroblastoma: Model Repository Evaluation

(General Questions)

Gender?
- ○ Male
- ○ Female
- ○ Prefer not to say

Age?
- ○ < 20
- ○ 20 - 35
- ○ 36 - 45
- ○ 46 - 55
- ○ 56 - 65
- ○ > 65

Highest level of education?
- ○ No Academic qualifications
- ○ Elementary school
- ○ Vocational qualification
- ○ Higher degree

Have you ever had a job in healthcare?
- ○ Yes as a Provider (e.g. nurse, doctor, etc.)
- ○ Yes as a Researcher (e.g. university, public health, etc.)
- ○ Yes as Other (e.g. hospital manager, health charity, pharmacist, etc.)
- ○ No

Do you have any long term health conditions?
- ☐ No
- ☐ Prefer not to say
- ☐ Diabetes
- ☐ Asthma
- ☐ Fibromyalgia
- ☐ Effects of stroke
- ☐ Epilepsy
- ☐ HIV/AIDS
- ☐ Glaucoma
- ☐ Problems related to alcohol or drugs
- ☐ Cancer
- ☐ Bowel disorder
- ☐ Thyroid condition
- ☐ Urinary incontinence
- ☐ Cataracts
- ☐ Sexually transmitted disease
- ☐ Άλλο:

How would you rate your computer skills?

1 2 3 4 5 6 7 8 9 10

Novice ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ Expert

Are you a member of any of the following social networking services?
- ☐ Facebook
- ☐ Twitter
- ☐ LinkedIn
- ☐ Google+
- ☐ XING
- ☐ Άλλο:

Have you ever participated in other MyHealthAvatar project surveys?
- ○ Yes
- ○ No

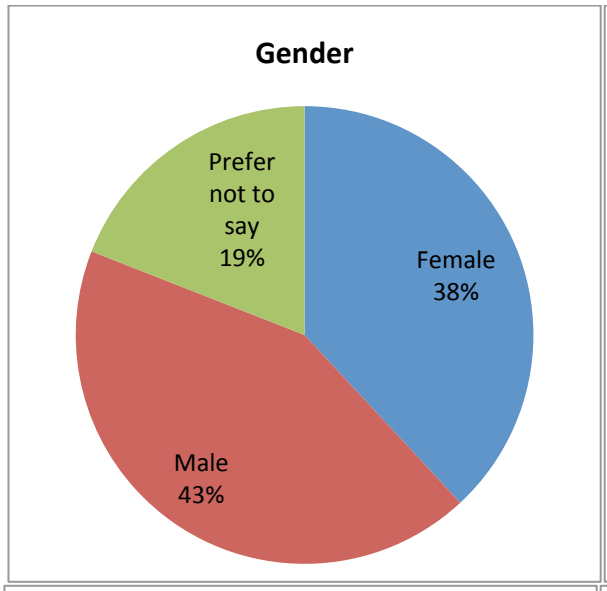
Where have you heard about MyHealthAvatar before?
- ○ The media (e.g. newspaper, radio, television)
- ○ The Internet (e.g. news article, Google)
- ○ The hospital (e.g. GP)
- ○ Word of mouth (e.g. friend or relative)
- ○ Meeting, workshop (e.g. presentation)
- ○ Survey invitation (e.g. e-mail)
- ○ Άλλο:

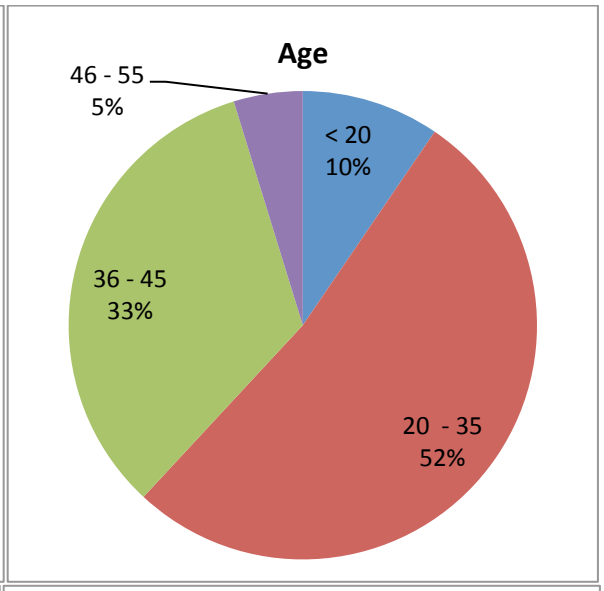
Have you ever heard anything about Electronic Health Records (EHR)?

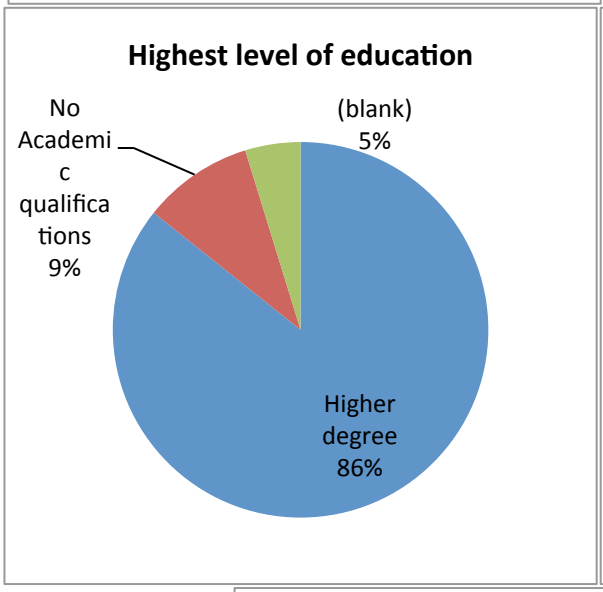
An Electronic Health Record (EHR) is an evolving concept defined as a systematic collection of electronic health information about individual patients or populations.
- ○ Yes
- ○ No

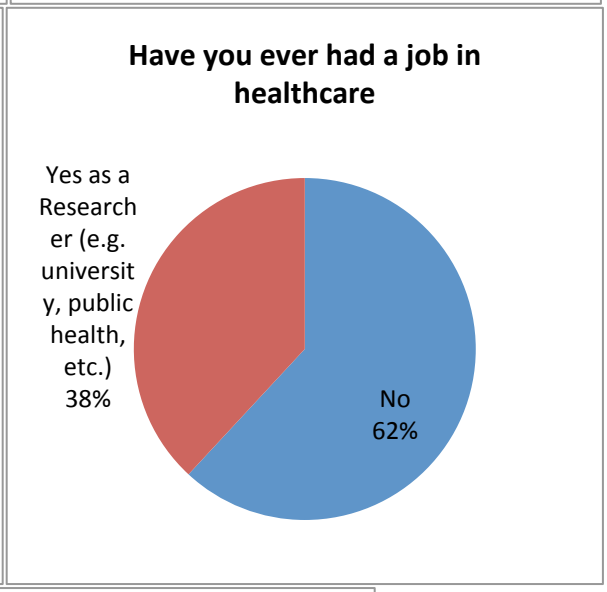
Figure 19: General questionnaire.

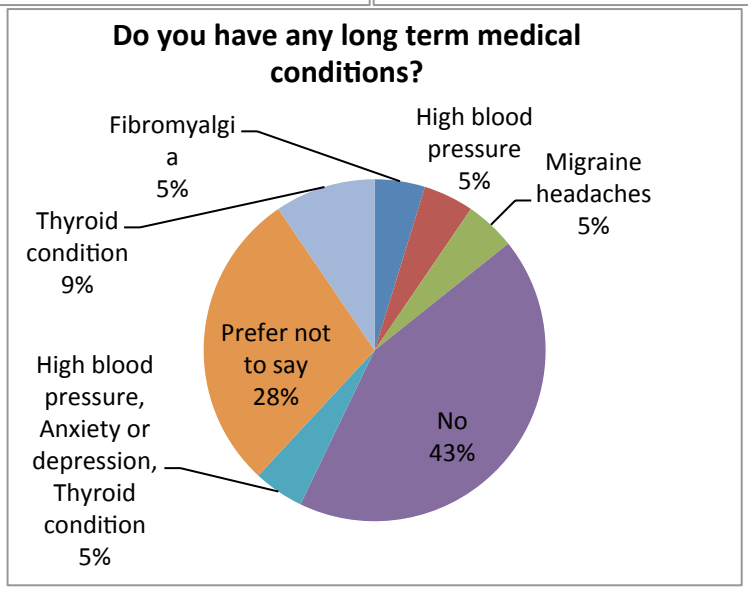
For each of the questions, the results percentages were calculated into pie graphs:


Gender
Prefer not to say 19%
Female 38%
Male 43%

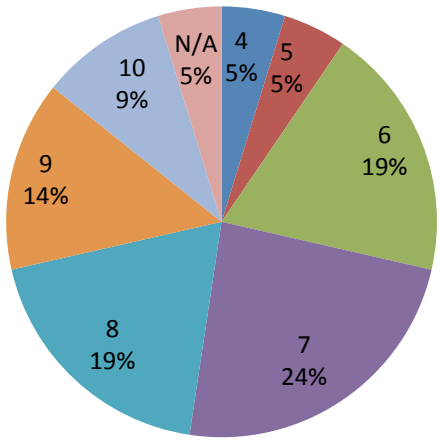

Age
46 - 55 5%
< 20 10%
36 - 45 33%
20 - 35 52%


Highest level of education
No Academic qualifications 9%
(blank) 5%
Higher degree 86%

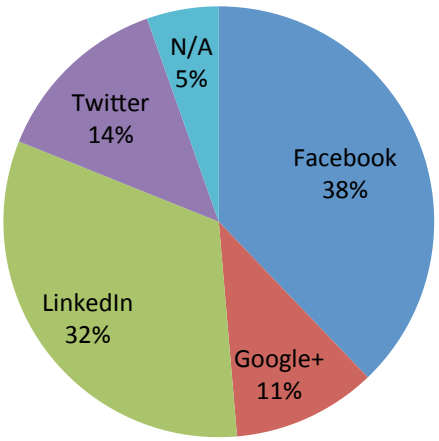

Have you ever had a job in healthcare
Yes as a Researcher (e.g. university, public health, etc.) 38%
No 62%


Do you have any long term medical conditions?
Fibromyalgia 5%
High blood pressure 5%
Migraine headaches 5%
Thyroid condition 9%
Prefer not to say 28%
High blood pressure, Anxiety or depression, Thyroid condition 5%
No 43%

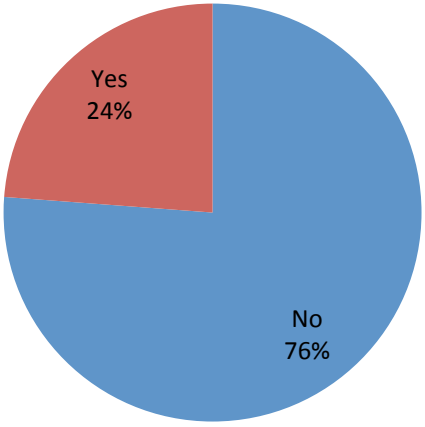
How would you rate your computer skills (1-10)?



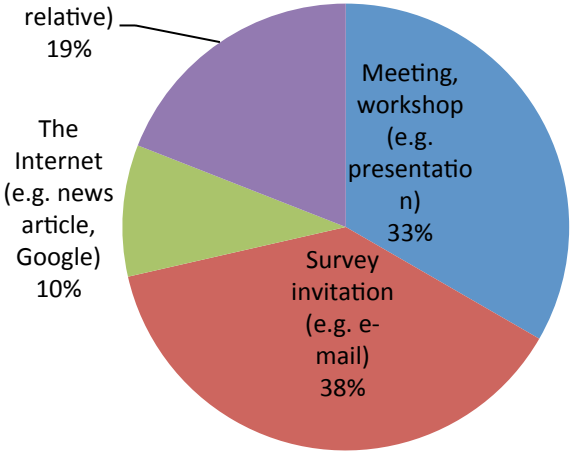
Are you a member of the following social networking services?



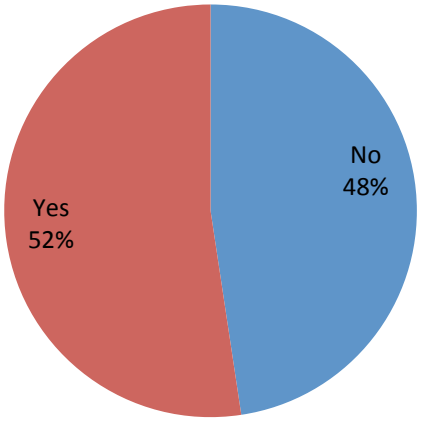
Have you ever participated in other MyHealthAvatar Surveys?



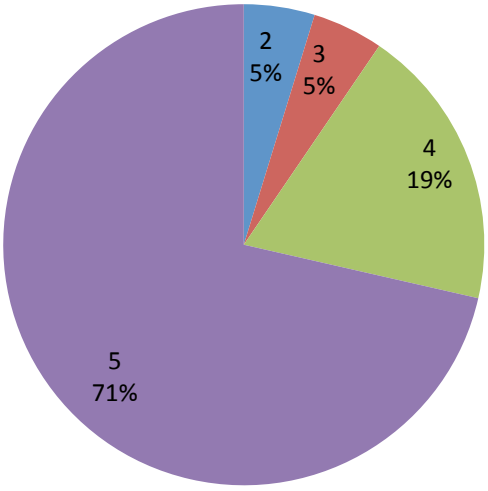
Where have you heard about MyHealthAvatarBefore?

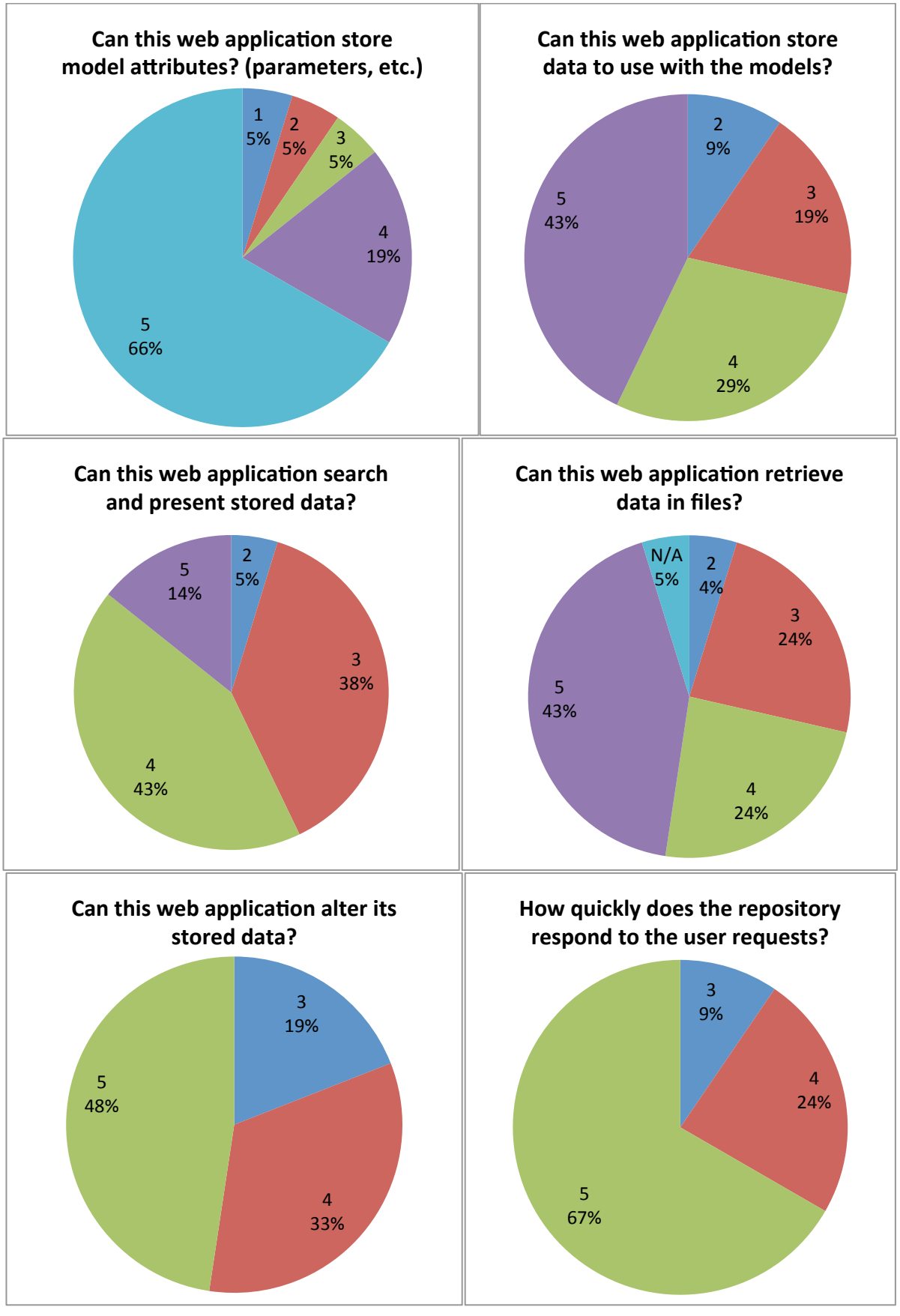


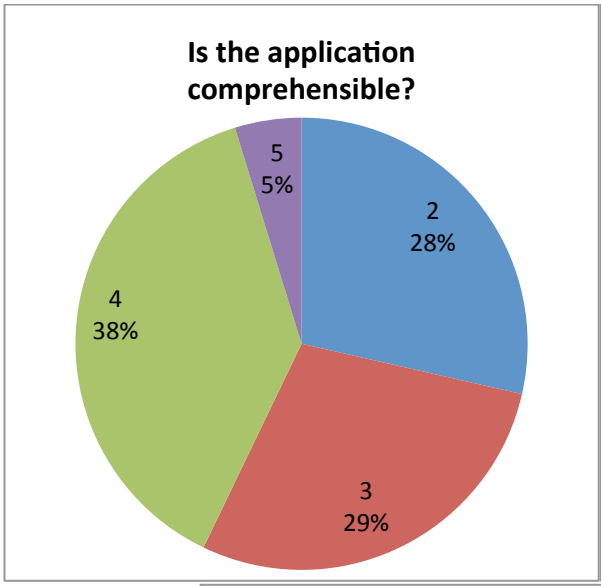
Have you ever heard anything about Electronic Health Records (EHR)?

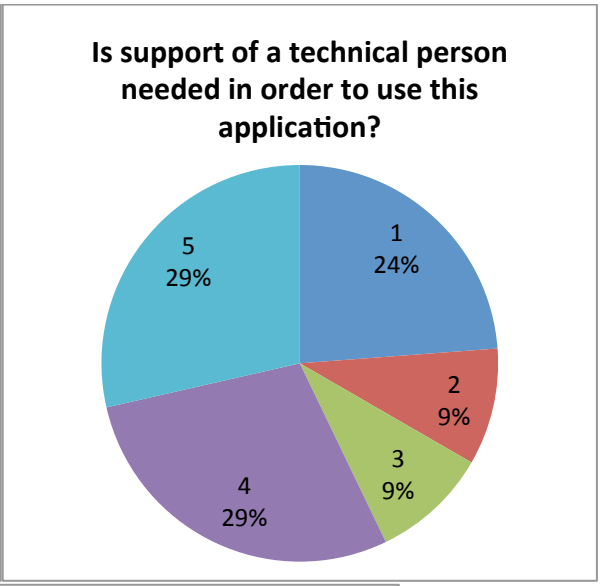


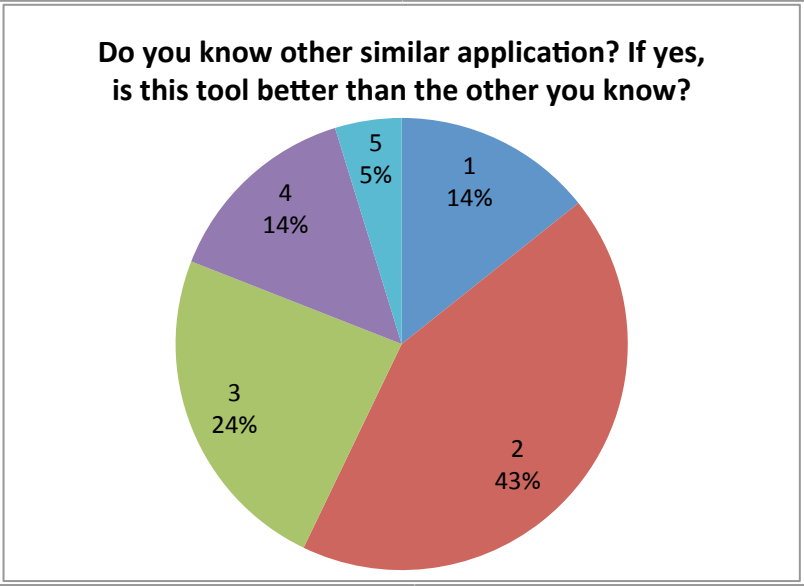
Can this web application store models?

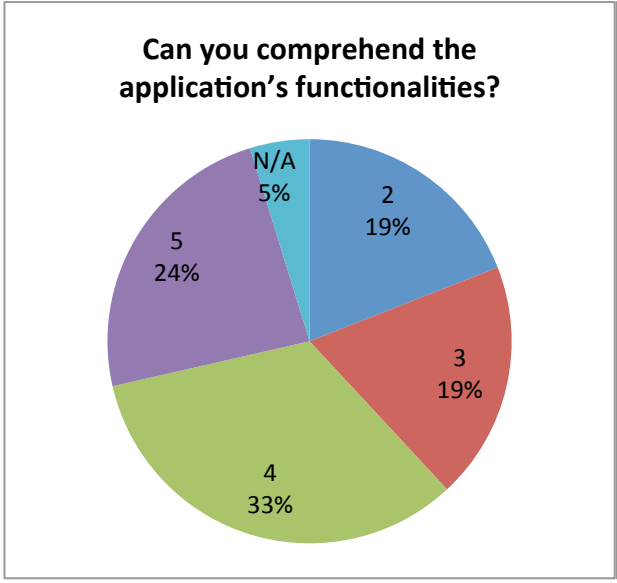


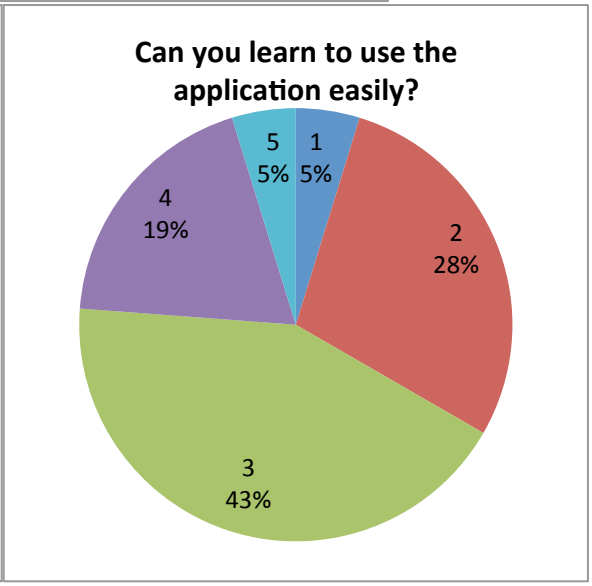

Can this web application store model attributes? (parameters, etc.)
1 5%
2 5%
3 5%
4 19%
5 66%
Can this web application store data to use with the models?
2 9%
3 19%
4 29%
5 43%
Can this web application search and present stored data?
2 5%
3 38%
4 43%
5 14%
Can this web application retrieve data in files?
N/A 5%
2 4%
3 24%
4 24%
5 43%
Can this web application alter its stored data?
3 19%
4 33%
5 48%
How quickly does the repository respond to the user requests?
3 9%
4 24%
5 67%


Is the application comprehensible?
5
5%
2
28%
4
38%
3
29%

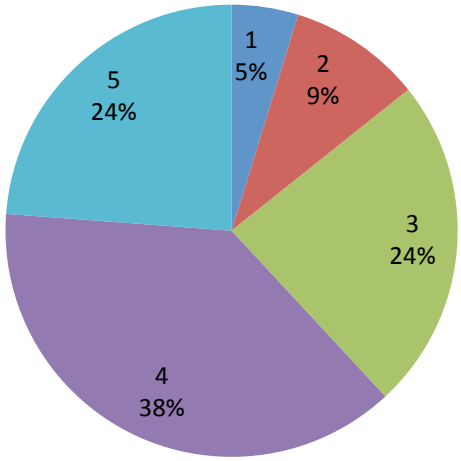

Is support of a technical person needed in order to use this application?
5
29%
1
24%
2
9%
3
9%
4
29%


Do you know other similar application? If yes, is this tool better than the other you know?
5
5%
1
14%
4
14%
3
24%
2
43%

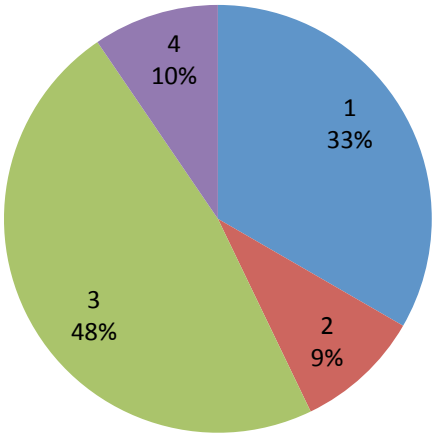

Can you comprehend the application's functionalities?
N/A
5%
2
19%
5
24%
3
19%
4
33%


Can you learn to use the application easily?
5
5%
1
5%
4
19%
2
28%
3
43%

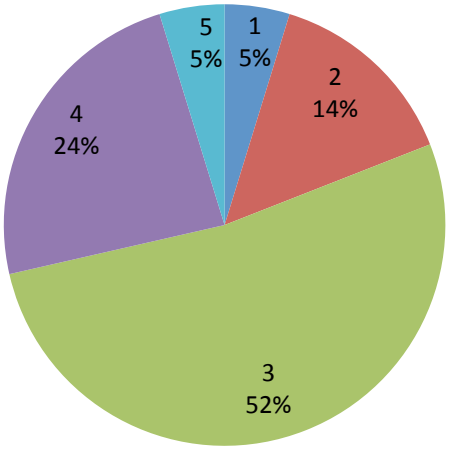
Can you use the application without much effort?



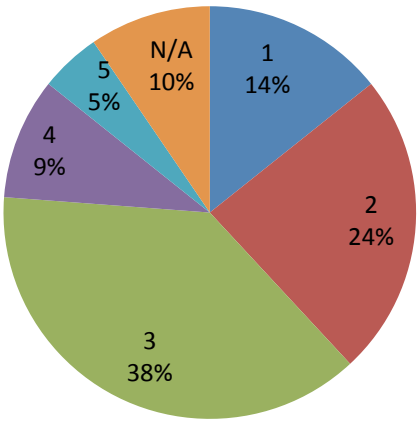
Does the interface look good?



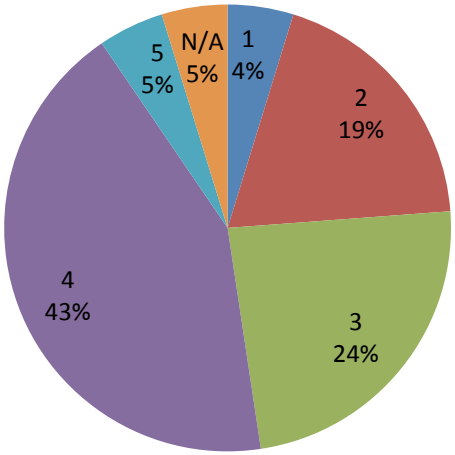
Does the interface provide all required information?



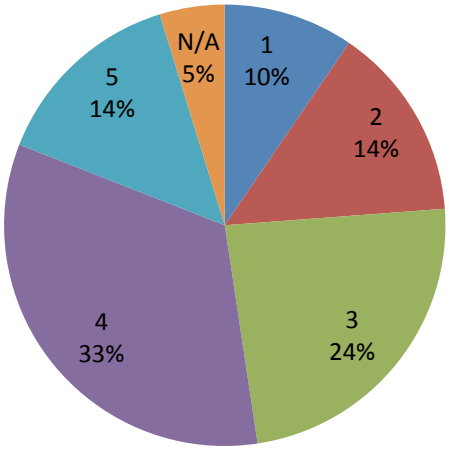
Have most of the faults in the software been eliminated over time?



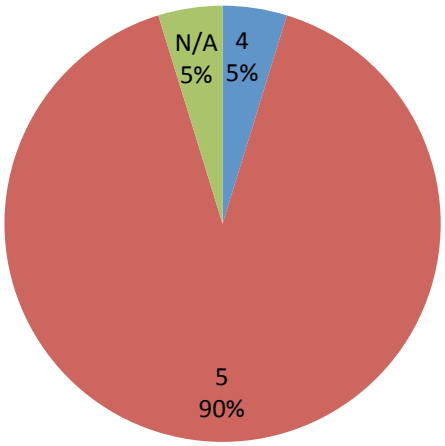
Is the software capable of handling errors?



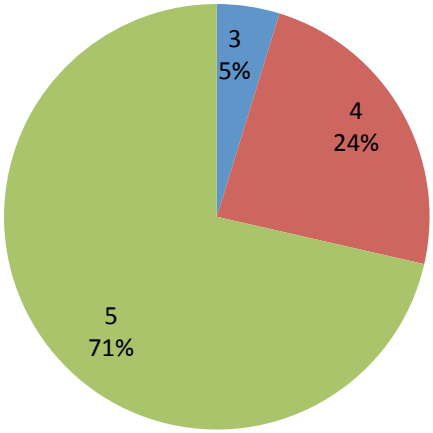
Can the services resume working & restore lost data after failure?



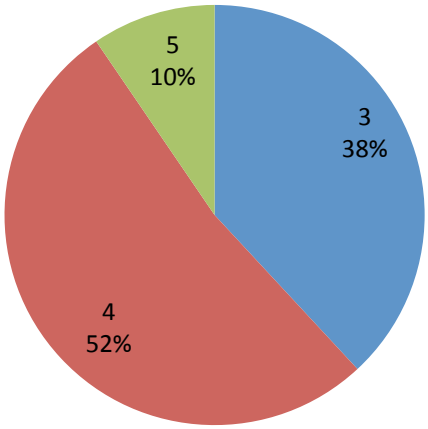
Can the web application be easily accessed from any pc?



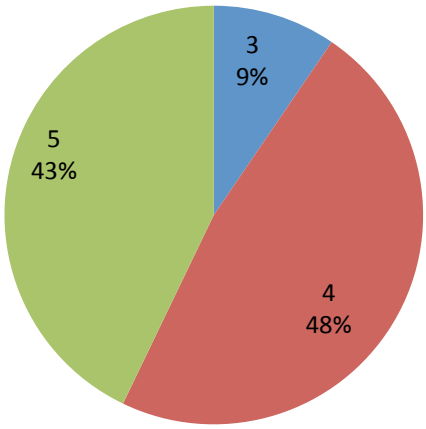
Are data accessible only to authorized users?



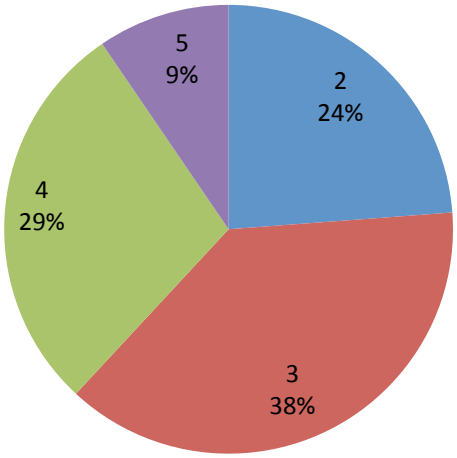
Do you think the uploaded data are secure?



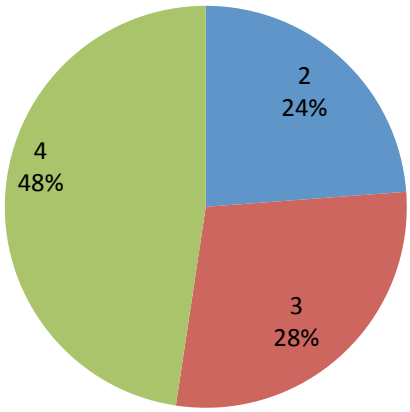
Does the system prevent unauthorized access?

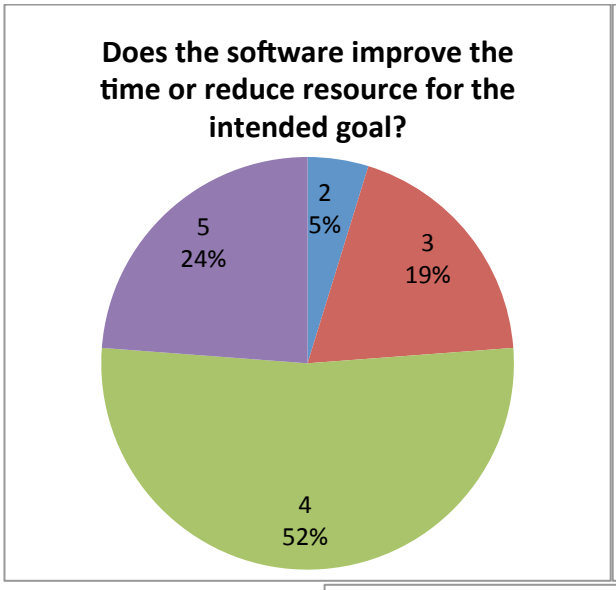


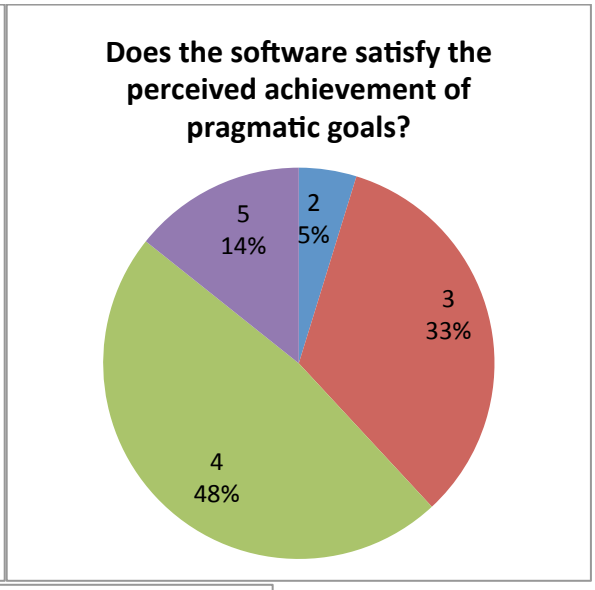
Can the software be tested easily?

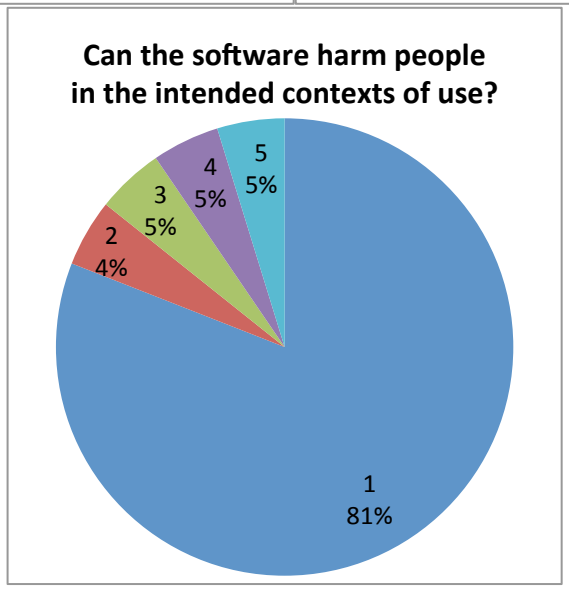


How accurate and complete is the software for the intended use?









7 References

[1] N.Christodoulou, N. Tousert, F. Misichroni, E. Geordgiadi, G. Stamatakos: A Modular Repository-based Infrastructure for the storage and execution support of multiscale models in the context of In-Silico Oncology and In-Silico Medicine (To be submitted in Cancer Informatics)

[2] https://docs.djangoproject.com/en/1.7/topics/forms/

[3] https://docs.djangoproject.com/en/1.7/topics/templates/#templates

[4] https://django-tastypie.readthedocs.org/en/latest/

Appendix 1 – Abbreviations and acronyms

API	Application Programming Interface
CRUD	Create, Read, Update and Delete
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
REST	REpresentational State Transfer
VPH	Virtual Physiological Human