

École polytechnique de Louvain

Adding New Features to an Account Management Web Application

Author: **Nicolas SAUVENEE**

Supervisors: **Etienne RIVIÈRE, Damien FRANÇOIS**

Readers: **Thomas KEUTGEN, Benoît DUHOUX, Ramin SADRE**

Academic year 2019–2020

Master [60] in Computer Science

Abstract

The *Consortium des Équipements de Calcul Intensif* (CÉCI) is a consortium that organizes the sharing of high-performance computing facilities and resources among five French-speaking universities in Belgium. To manage the access rights of researchers in the consortium to clusters of CÉCI's partner universities, a web application called Relog was created ten years ago. However, the evolution of web technologies and the needs of the universities have changed. The objective of this thesis is to improve and add new features to this application in order to keep it up-to-date and in accordance with the new needs of CÉCI. This work covered three contributions to the application. The first was to allow the application to handle new types of projects. The second was the development of a tool to generate reports based on demographic information extracted from the application database. The last was an application security analysis covering common types of security threats.

Acknowledgements

We want to thank Damien Francois for his advice, his patience and the time he has given us to complete this thesis.

We would like to thank all CISM staff for the permissions and certificates needed to access the application, and more specifically Patrick Vranckx, who prepared the virtual machine for us.

We would also like to thank Etienne Rivière for his time and help during the writing of this thesis and throughout the academic year.

Contents

1	Introduction	4
1.1	Description of the General Context: HPC	4
1.2	Presentation of the Users: CISM and CÉCI	5
1.3	Presentation of the Application: Relog	6
1.4	Structure of the Thesis	6
2	Background	8
2.1	Context	8
2.2	Relog as a Web-Based Solution	9
2.3	Objectives and Challenges	9
2.3.1	Adding New Project Types	10
2.3.2	Implementation of a Tool to Extract Demographic Information	10
2.3.3	Security Audit	11
2.4	Technical Architecture Overview	11
2.5	Principal Tools Description	13
2.5.1	SQLAlchemy	14
2.5.2	Bottle	14
2.5.3	KnockoutJS	15
2.6	Summary	16
3	Contribution 1: Adding New Project Types	17
3.1	Context	17
3.2	Problem	18
3.3	Implementation	19
3.3.1	Initial Implementation	19
3.3.2	Back-End Implementation	20
3.3.3	Front-End Implementation	21
3.3.4	Use Case Scenario	22
3.4	Software Testing	24
3.4.1	Testing Approach	24
3.4.2	Unit Testing	24

3.4.3	Integration Testing	27
3.5	Summary	27
4	Contribution 2: Implementation of a Tool to Extract Demographic Information	28
4.1	Context	28
4.2	Design Choice	29
4.2.1	Technology Stack	30
4.2.2	Software Architecture	30
4.3	Use Case Scenario	33
4.4	Configuration	35
4.4.1	Parameterization Need	35
4.4.2	Configuration Format	36
4.4.3	Configuration Options	37
4.5	Summary	38
5	Contribution 3: Security Audit	39
5.1	SQL Injection	40
5.1.1	SQL Injection in SQLAlchemy	40
5.1.2	Security Vulnerabilities of SQLAlchemy	40
5.1.3	SQL Penetration Testing	43
5.2	Cross-Site Request Forgery	44
5.2.1	CSRF Vulnerability in Relog	44
5.2.2	Mitigation of CSRF Risks	45
5.3	Cross-Site Scripting	45
5.3.1	Non-Persistent XSS	46
5.3.2	Persistent XSS	46
5.4	Vulnerability Scanning	47
5.4.1	Scan Results	47
5.5	Summary	49
6	Conclusion	50
A	Contributions	51
A.1	Default Username Size	51
A.2	Message in Web UI Misleading	52
A.3	Create a New Email Template	52
A.4	SSH2 Key Format Management	53
A.5	Summary	55

B Documentation: Adding new project types	56
B.1 User Documentation	56
B.2 Developer Documentation	63
B.3 Administrator Documentation	73
B.4 Integration Testing	76
C Documentation: Extract Demographic Information	81
C.1 Documentation	81
D Security Audit	91
D.1 Sqlmap	91
D.1.1 Example of HTTP Request Tested	91
D.1.2 Example of Request Generated by Sqlmap	91
D.1.3 Obtained Results	91
D.2 ZAP Scanning Report	93
List of Figures	103
List of Tables	104
Bibliography	106

Chapter 1

Introduction

This thesis was prepared in collaboration with the Calcul Intensif et Stockage de Masse (CISM [1]). For this work, Antoine Denauw and myself worked together for some parts of the thesis during the first semester of the academic year. The sections that have been done jointly are the introduction and the background. Furthermore, we also collaborated on the first contribution described in Chapter 3 and in which the front-end implementation has been done by Antoine Denauw.

In this introduction, we will start by describing the general context of this thesis and the notion of high-performance computing. Then, we will present the CISM, the CÉCI, and the Relog application we have been working on. Finally, we will explain the structure and the different chapters of this thesis.

1.1 Description of the General Context: HPC

The ability to collect and analyze more and more data is becoming a major challenge in all scientific disciplines. To enable scientists and engineers to make new discoveries, the need for high computing power and capacity is becoming increasingly important. In this context, the use of High-Performance Computing facilities (HPC [2]) is required to meet tomorrow's challenges.

High-performance computing, also known as supercomputing, is the practice of aggregating computing power by using a supercomputer to analyze data and perform calculations to solve big problems. Such a computer provides much higher performance than a simple desktop workstation. One of the best-known types of supercomputer is the HPC cluster, which can be defined as a set of computers connected to each other and configured to appear as a single machine [3].

Nowadays, most universities are equipped with their own clusters. For example, the supercomputing center of UCLouvain hosts several computing clusters in a data center on campus, including the DC III that we visited, and that is illustrated in Figure 1.1. The computing clusters installed at the UCLouvain are managed by the "Calcul Intensif et Stockage de Masse" (CISM), and the latter is a member of a larger organization called CÉCI [4], which organizes the sharing of all HPC computing resources among francophone universities across the country. During this thesis, we worked on the application used to manage access to all clusters belonging to CÉCI member universities.



Figure 1.1: Photos taken during the visit of Data Center III at the UCLouvain [5].

1.2 Presentation of the Users: CISM and CÉCI

Before presenting in more detail the application we had to improve, it is important to understand the actors we worked for and their respective roles in the management of HPC computing resources.

CISM [1] stands for "Calcul Intensif et Stockage de Masse" and was created in 2004 by the users of the HPC facilities of different departments of the Université catholique de Louvain. CISM organizes HPC training courses, offers expertise in code porting, operates computing clusters and storage at UCLouvain, etc. In 2018, CISM operates three clusters, and two of them are part of a larger organization called CÉCI.

CÉCI [4] is the 'Consortium des Équipements de Calcul Intensif'; a consortium of high-performance computing centers from the French-speaking universities of

Belgium (UCLouvain, ULB, ULiège, UMons, and UNamur) supported by the F.R.S-FNRS and the Walloon Region.

The objective of CÉCI is to provide each researcher from the consortium’s partner universities access to computer clusters adapted to his or her needs. By organizing the sharing of HPC resources between universities, CÉCI enables a diversity of equipment that a university could not achieve alone. Moreover, to help researchers in the field of supercomputing, CÉCI also aims to offer them with the best training and advice possible.

1.3 Presentation of the Application: Relog

The Relog application has two main roles. The first is to provide access to CÉCI’s clusters to all researchers from Belgium’s French-speaking universities. Through this application, any researcher from these universities can ask for an account via a web form, and after validation of the latter by a CÉCI administrator, this account gives them access via SSH [6] to all CÉCI clusters. The second role of the application is to allow researchers to create projects to access a large supercomputer of Wallonia. Technical details, as well as a more precise description of this supercomputer and the application, are discussed in the next chapter to help the reader understand the features we have implemented.

To get used to this application and how it works, we have reached an agreement with Pr. Rivière and Dr. François to first solve some bugs and fixes useful to our understanding, but also to the application itself. These fixes were made by a merge request to the GitLab [7] of the application, and were then accepted by the developer in charge. We describe them in Appendix A.

1.4 Structure of the Thesis

In this thesis, we will cover three contributions we have made to the application. Two of these contributions are features we have implemented. The first was to allow the application to handle the creation of new types of projects that could be deployed on CÉCI’s clusters in the future. The second was to develop a tool to generate reports based on demographic information extracted from the application’s database. Finally, the last contribution was to perform a security audit of the application covering common types of security threats.

In addition to the introduction chapter, we decided to structure our work in four parts to present distinctly the different contributions we made during this thesis. Two of these four chapters are directly related to the improvements we have made to the Relog application and for which we can find the complete software documentation in Appendix [B](#) and [C](#).

The second chapter will discuss the background of the thesis necessary to understand our work, we will present the Relog application, the different objectives and challenges that were accomplished, and finally, a technical description of the architecture of the application.

The third chapter will present the main feature we had to implement to allow the Relog application to handle new types of projects. We will start by analyzing the context, then we will present the implementation, and lastly, we will discuss the software testing process.

The fourth chapter will focus on the tool we have created to generate reports using information extracted from the database of the Relog application. We will first detail our design choices as well as the technologies we have used, then we will show an example of a report that this tool can generate and explain the configuration possibilities of the latter.

Finally, we will discuss the results of the security analysis of the Relog application we performed, and we will terminate with a conclusion.

Chapter 2

Background

This chapter will be divided into 5 sections. We will first introduce the context that led to the development of the Relog application. Thereafter, we will introduce the Relog application and the problems it solves. Afterward, we will present the different objectives of this thesis, and finally, we will discuss the technical architecture as well as the main tools that we used.

2.1 Context

To understand how the Relog application works, it is necessary to grasp the context and the reasons that led CÉCI to develop it. As seen previously, access to the various CÉCI clusters is done directly through the SSH protocol. To allow users to connect to the CÉCI clusters, the different universities must be able to store their SSH public keys. For this purpose, the solution that was chosen is to use a directory service, which can be seen as a specialized database, optimized for reading operations, and accessed through the Lightweight Directory Access Protocol (see Section 2.4 for more details regarding the LDAP [8]). Before the creation of the Relog application, each university was responsible for managing its own directory service by, for example, adding new data manually.

When CÉCI was founded, the problem of centralizing, creating, and maintaining data related to its users emerged. Since each researcher who is a member of a CÉCI university must be able to access all the clusters of all CÉCI member universities, it was necessary to find a way to centralize this process and to provide a solution to ensure that the data could be replicated and accessible to all.

2.2 Relog as a Web-Based Solution

In this context, the idea of a web-based solution appeared. The initial concept was that the user creation workflow and maintenance could be done through a web application. The role of the latter would, therefore, consist of updating a single central directory, which would be replicated to the different CÉCI's LDAP servers to make the information available to all partner universities.

The Relog application has concretized this idea of a web application allowing to update a central directory server. The creation of accounts, the updating of the different status of CÉCI users by the administrators will be done from a web interface.

Subsequently, Relog has been extended to handle the creation of projects that bring together several users and give them access to a larger Tier-1 cluster. This type of cluster, which we will describe in more detail in Section 3.1, is managed on a national level and has a higher computing performance than university clusters. To achieve this, users must complete an online form, and once an administrator has validated it, a UNIX group and its corresponding project will be created on the Tier-1 cluster. Each member will automatically be able to access it using its CÉCI SSH key and will be added to the UNIX group. This web form was created as a filter to standardize the management of user rights and the subsequent creation of projects with a few clicks.

Relog has now been in place for more than ten years and is still frequently used. As we can see in Figure 2.1, in 2018, it had more than 2184 registered accounts for all its partner universities, including a majority for the Université catholique de Louvain and Université de Liège. In addition to the users, more than a hundred projects have been registered to date.

2.3 Objectives and Challenges

Now that we have a better overview of the purpose of the Relog application and before discussing its technical aspects, we will present succinctly the different objectives that we will have to accomplish during this thesis as well as the challenges that we will have to face.

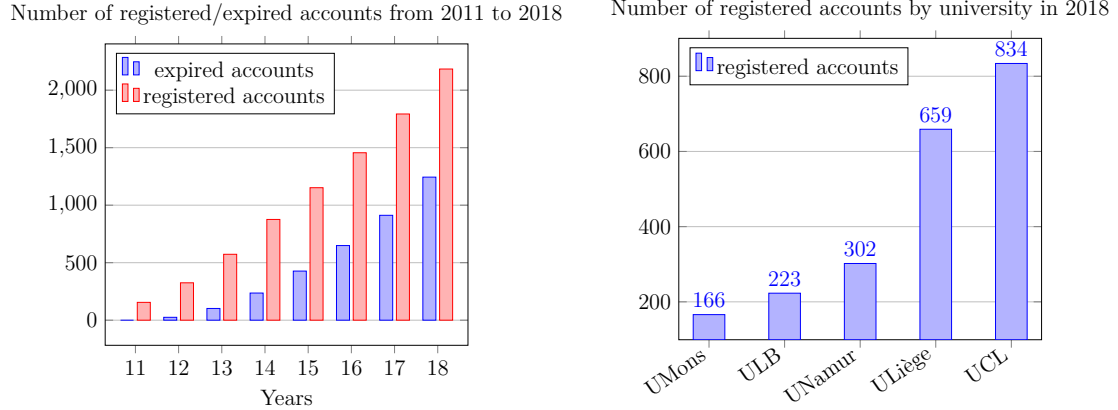


Figure 2.1: Statistics on the number of accounts registered on the Relog application.

2.3.1 Adding New Project Types

The first objective will be to enable the Relog application to manage the creation of projects other than those of the Tier-1 type. It is important to understand that only projects that are intended to be run on the Tier-1 supercomputer can be created directly from the Relog application. The creation of projects and groups for CÉCI clusters is not yet feasible directly from the application. This may have some drawbacks, for example, if a user wishes to share files with only certain users, he will have to contact an administrator, and the latter will have to create a UNIX group manually.

To achieve this task, we will have to face several challenges. The first difficulty will be to guarantee that the different types of projects can be inserted into the central directory server without creating negative interactions. For example, by ensuring that for each project created, the value of its `gidNumber` is assigned within a range specific to its project type. Another important objective will be to allow CÉCI members to create new types of projects while allowing them to define the characteristics that will be required by it (CPU hours, disk space, etc.). Finally, we will have to ensure the quality and the validity of our implementation through software testing.

2.3.2 Implementation of a Tool to Extract Demographic Information

For the second task, we will have to develop a tool to generate reports using information extracted from the database of the Relog application. The reports generated will be primarily related to the CÉCI institutions, and more specifically,

to data on the number of registered and expired users as well as information on projects.

Since the reports could subsequently be integrated directly on the CÉCI website, one of the main objectives will be to make it possible for CÉCI members to configure the reports to be generated.

2.3.3 Security Audit

Finally, we will perform a security analysis of the Relog application in which we will cover several types of threats. We will begin by verifying whether attacks of the SQL injection type can be exploited on the Relog application, then we will analyze the risk of cross-site request forgery and cross-site scripting attacks, and lastly, we will try to detect potential vulnerabilities in the application using scanning tools.

2.4 Technical Architecture Overview

In this section, we will detail the architecture of the Relog application, as well as the general structure that has been put in place by the CÉCI. We will illustrate the different interactions with a concrete example of account creation.

The back-end of the Relog application has been developed in `Python` using the `Bottle` [9] framework that we will describe in the next section. The application's endpoints are defined in the file `relogd.py`, and the client-side interactivity is mainly done by exchanging data in JSON format. To do this, the Relog application delivers static content to end users, and uses the `Knockout.js` [10] library to develop the client-side interactivity by applying the `Model-View-ViewModel` [11] pattern discussed in Subsection 2.5.1. In the example of Figure 2.2, the file `relogd.py` will serve the static file `init.html`, which will allow a user to create an account using a valid email. Once the user has provided an email address and sent it, an AJAX request will be sent to the back-end of the Relog application. The user will then receive an email with a link to a form to fill in to complete the account creation process.

Relog uses `SQLite` [12] as a relational database management system. The tables are defined via the classes in the `model.py` file using `SQLAlchemy` [13], a Python-based object-relational mapper, which we will detail in the next section. In the example 2.2, once the user has completed all the fields and submitted the form, a new row will be inserted into the `user` table with all the information provided.

The latter will then have to wait until a CÉCI administrator validates the creation of his account to be able to access the different clusters.

As seen previously, Relog accesses and maintains information directories using the Lightweight Directory Access Protocol. The latter is a standards-based protocol that sits on top of TCP/IP for interacting with directory servers and is often used for authentication and storing information about users, groups, and applications [8]. The information that will be contained in the directories will be the data that needs to be replicated and accessed directly by the clusters. To grasp the purpose of the Relog application, it is therefore important to understand the CÉCI's LDAP infrastructure that has been put in place.

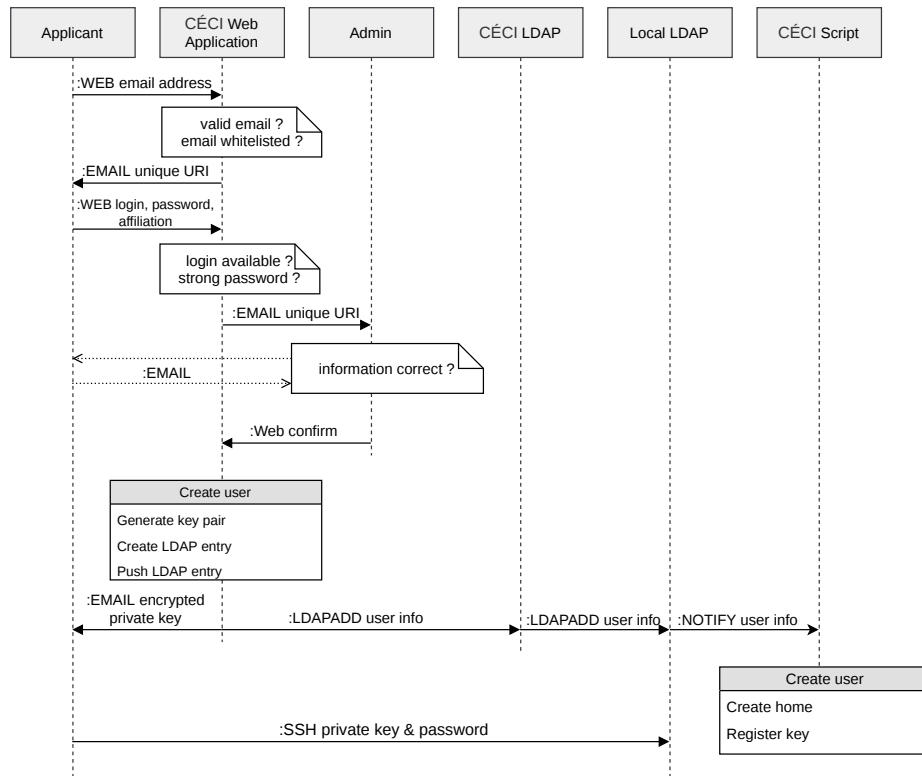


Figure 2.2: CÉCI login management workflow, adopted from the Relog documentation.

CÉCI's LDAP infrastructure is based on master-slave replication (see Figure 2.3), we have a central server hosting the Relog application as well as the master LDAP server with which Relog communicates directly. For example, the Relog application

will add a new entry into the directory-server database when a new project has been created and confirmed by a CÉCI administrator. LDAP directories of the master LDAP server are replicated at regular intervals on the different CÉCI's slave LDAP servers. To return to the previous example, when an administrator validates the creation of the user account, some of his data including his public key will be pushed by the application on the master LDAP server, and thereafter, these modifications will be transferred to all slave servers.

Finally, on the front-end clusters, a daemon (SLUFL [14]) polls the LDAP directory from its LDAP server at regular intervals, which allows, for example, to set up the new users on the clusters when a change in the LDAP directory is detected. Once the latter have been updated, the new user will be able to access them using his private key via SSH.

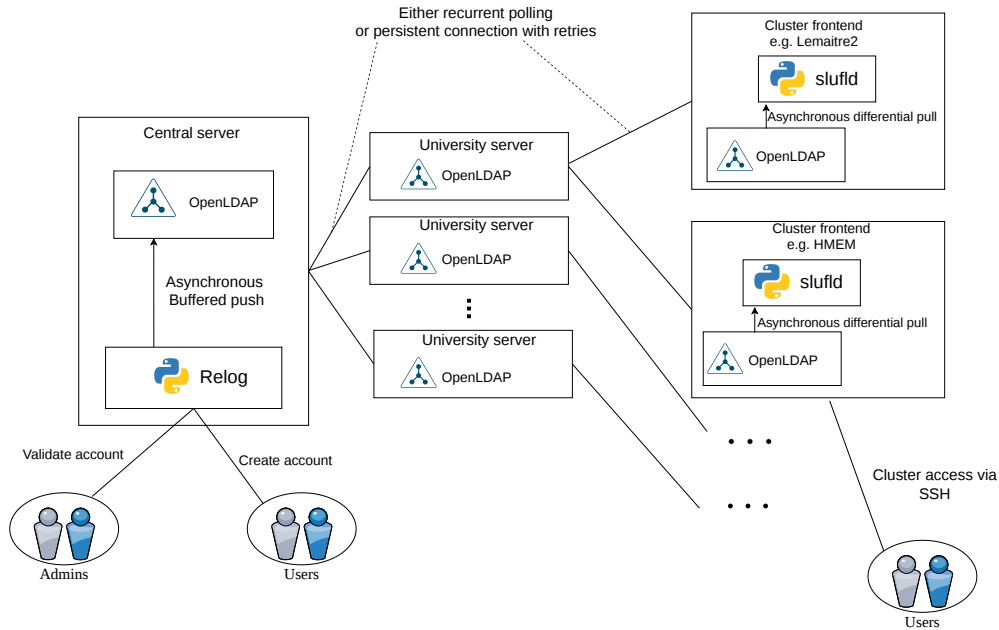


Figure 2.3: CÉCI's LDAP architecture, adopted from the Relog documentation.

2.5 Principal Tools Description

This part of the chapter will consist of a more detailed description of the main tools used by the Relog application.

2.5.1 SQLAlchemy

SQLAlchemy [13] is a Python SQL toolkit and object-relational mapper released under the MIT License. Using this tool allows the Relog application to define the mapping between database schemas and classes. In contrast to other ORM systems, SQLAlchemy allows the application to describe the database tables and defining our own classes, which will be mapped to those tables, all at once. On Listing 2.1, we can see the class `Institution` performing the two tasks discussed above together. In Relog, the classes describing the different tables of our database are defined in the `models.py` file.

```
1 class Institution(Base):
2     __tablename__ = 'institution'
3     __table_args__ = (UniqueConstraint('parent', 'name'),)
4
5     id = Column(Integer, Sequence('institution_id_seq'), primary_key=True)
6     name = Column(String)
7     abbreviation = Column(String)
8     levels = Column(String)
9     admin_email = Column(String)
10    parent = Column(Integer, ForeignKey('institution.id'))
```

Listing 2.1: Class `Institution` including directives to describe the database table `institution` of the Relog application to which it will be mapped.

Another advantage of using SQLAlchemy comes from the fact that the application can create, read, update, and delete database objects as if they were normal Python objects. This made it easier for the Relog application to get an effective access-control list with SQLite.

2.5.2 Bottle

Bottle [9] is a simple and lightweight micro-framework for Python. The web application Relog is created using the Bottle framework on a CentOS server. Bottle uses a powerful routing engine that Relog exploits in order to define its different static and dynamic routes 2.2. The advantage of this framework comes from its performance compared to other Python's web framework [15]. Moreover, it is easy to learn and ideal for small web applications, which makes it the ideal framework for the Relog application.

```

1  # Static content for init, user, project and board
2  @route('/init/')
3  def init():
4      return static_file('init.html', root=STATIC)
5
6  @route('/user/:id')
7  def user(id):
8      return static_file('user.html', root=STATIC)
9
10 @route('/init-project/')
11 def init_project():
12     return static_file('init_project.html', root=STATIC)

```

Listing 2.2: Examples of routes defined in the Relog application using the **Bottle**'s route decorator.

2.5.3 KnockoutJS

Knockout.js is a JavaScript library [16] based on the Model-View-ViewModel pattern that facilitates the creation of user interfaces with JavaScript and HTML. The underlying principle of **Knockout.js** is a separation of the visual interface into three parts:

- A model: The application's data, in the case of the Relog application the model part is supplied using AJAX calls to the Relog server.
- A view model: Abstraction of the view exposing data and operations needed by it. With **Knockout.js**, view models are JavaScript objects.
- A view: A visible aspect of the UI i.e. our different HTML documents. The latter represents the state of the view model to which it is linked through the declarative bindings. **Knockout.js** uses observers to make the UI automatically updated when the underlying data model changes.

On Listing 2.3, we can see a simplified example of the use of **Knockout.js** by the Relog application. We can find part of the HTML code of the view defined in the `user.html` file, and, a simplified version of the view model `User` defined in the `relog.js` file inside the script element. In this example, the view is simply the visible UI of the account creation form, and the view model `User` contains its underlying data such as its name. Finally, declarative bindings make it possible to connect the latter. For instance, when a user edits its name, the value of the `first_name` attribute of the `User` view model will be updated with the value of the associated input from.

```

1 <!-- View -->
2 <form>
3   <h2>1. Enter your
4     <span data-bind="visible: is_guest()">
5       guest
6     </span>
7     information
8   </h2>
9   <div class="form-group">
10    <label class="control-label" for="first_name">First Name</label>
11    <input type="text" class="form-control"
12      id="first_name" placeholder="John"
13      data-bind="value: first_name">
14  </div>
15  <button type="submit" class="btn btn-primary"
16    data-bind="enable: enable_save">
17    Save
18  </button>
19 </form>
20
21 <script>
22   // View model
23   var User = function() {
24     this.is_guest = ko.observable(false);
25     this.first_name = ko.observable().trimmed();
26     this.status = ko.observable('draft');
27
28     this.enable_save = ko.computed(function () {
29       if (this.status() !== 'draft'){
30         return false;
31       }
32       return true;
33     }, this);
34   }
35   ko.applyBindings(new User())
36 </script>

```

Listing 2.3: Simplified example of the use of the `Knockout.js` library by the Relog application.

2.6 Summary

In this chapter, we have introduced the background of this thesis to help the reader understand the next chapters, the Relog application, and finally, its general architecture and the structure that CÉCI has developed for it.

Chapter 3

Contribution 1: Adding New Project Types

Now that we have a good understanding of the application, we have decided to prepare the main feature requested by the CISM: *Adding new project types*. The complete software documentation can be found in [Appendix B](#).

3.1 Context

Before going into the details of the implementation of this feature, it would be useful to set out the context and reasons that led CÉCI members to create the issue "Update to handle other types of projects than the Tier-1 projects". To get the background, it is first necessary to understand what the PRACE is and the relationship between the latter and CÉCI.

The PRACE initiative - Partnership for Advanced Computing in Europe - aims to provide access to leading-edge computing and data management resources for large-scale scientific and engineering applications through a pan-European research infrastructure offering HPC services and supercomputers [\[17\]](#). The PRACE infrastructure adopts a tiered approach with 3 distinct layers, the first being the European Centres (Tier-0) with an average number of cores of 100,000 and a computing performance of 2 petaFLOPS, the second the National Centres (Tier-1) with an average number of cores of 10,000 and a computing performance of 200 teraFLOPS and finally the Regional/University Centres (Tier-2) with an average number of cores of 1,000 and a computing performance of 20 teraFLOPS. Through these different layers, the PRACE infrastructure encourages researchers to test and develop their techniques first at the lower levels (Tier-2) and then to apply for computing time on country-level Tier-1 clusters and Europe-level Tier-0

infrastructures [18].

Belgium has been a member of the PRACE initiative since 2012, and in the Wallonia-Brussels Federation, Tier-2 computers are managed by CÉCI. Cenaero, which is an applied research center focused on the development of numerical simulation and optimization in the aerospace sector [19], has been responsible for implementing and operating Tier-1 Zenobe supercomputer in Wallonia. Most of the resources provided by the machine are dedicated to academic research.

The CÉCI allows users to access the Zenobe supercomputer. Access to Zenobe can only be done through projects created by CÉCI users who hold a permanent position within a member university. When creating a project, the user must fill in a certain number of fields such as CPU hours (an estimate of the number of CPU hours the project will require), disk space (the maximum disk space the project will need), and scratch space (the space on the hard disk drive that is dedicated to temporary storage). Then, other CÉCI users can register for this project and will have access to the resources on Zenobe that are allocated to the project.

3.2 Problem

Currently, it is only possible to submit one type of project on the Relog application. The only projects that can be created are those intended to be run on the dedicated CÉCI share on the Zenobe supercomputer. When a project of a type different from Tier-1 has to be created by a user of the Relog application, the project must be created manually by a CÉCI administrator. Our main challenge will, therefore, be to make it possible to create new types of projects other than Tier-1 projects on the Relog application.

An example of a type of project currently used on CÉCI's clusters and which could be created directly on the Relog application are the projects of type "Managing storage" [20]. This new type of project would not require CPU hours and scratch space, but only information on the storage space that the project needs. Another type of project that is already used by the system administrators of the CÉCI universities and which might be created directly on the Relog application is the 'Linux group' type. This type of project is used for a group of users who want to share data or executable files. It is important to mention that the different types of projects have predefined `GIDnumbers` [21] that will have to be taken into account later on when defining a new type of project.

Finally, when a CÉCI member creates a new project, he should be able to directly choose the type of project he wants to submit among the available project types. Thereafter, the form will have to be adapted according to the fields required by the selected project type. As will be discussed below, the administration interface and the server-side will also have to evolve in order to take into account this new change.

3.3 Implementation

In this section, we will detail the approach we used to solve issue 4 "Update to handle other types of projects than the Tier-1 projects". This section will be divided into four subsections. The first will detail what was already in place on the Relog platform. The second will explain some of the changes we had to make at the back-end level of the application to allow several types of projects. The third subsection will briefly detail how we modified the user interface. Finally, in the last subsection, we will illustrate different possible use case scenarios resulting from our new implementation. For a more detailed description of code changes, refer to [Appendix B.2](#).

3.3.1 Initial Implementation

One of the CISM requirements was that we had to propose a solution based on what was previously designed on the application. Indeed, the Relog application was developed to be extended in the future to manage new types of projects. It was, therefore, important for us to take into account the initial implementation.

Initially, the `ProjectType` model had the attributes `name` and `eula`. The latter correspond to the name and the link to the terms and conditions of the project type. When a project was pushed into LDAP, the `gidNumber` attribute, which is used to identify a UNIX group, was generated from the value of the `last_uid` attribute of the unique counter in the `projectid_counter` table.

For the LDAP project entries, the `description` attribute that defines the associated project in the LDAP was generated from the fields `name`, `acronyms`, `create_date`, `confirmed_by`, `project_description`, `investigator_email`, `investigator_login`, `category`, `type`, `cpu_time`, `disk_space`, `scratch_space`, and `expiration_date` of the inserted project. All these fields except `expiration_date` had to be present in the description.

3.3.2 Back-End Implementation

As the different types of projects that may be needed by the Relog application in the future are not yet all clearly defined, the second requirement of CISM was to allow CÉCI's administrators to be able to create new types of projects quickly and easily. To this end, we decided to develop a script that can be executed on the command line interface by an administrator with access rights on the server. When the latter has to create a new project type, he will then be able to run this script and, using the parameters passed to the program, configure the type of project to be created.

When the script will be executed, the `project_type` and `projectid_counter` tables will be updated according to the arguments passed to the program and which can be found in Appendix B.1. Firstly, a new record will be added to the `project_type` table with the values transmitted to the program during its execution for the `name` and `eula` attributes. Secondly, a new record will be inserted in the `projectid_counter` table. This is a new counter specific to the newly created project type. The `last_uid` attribute of the new record will have the value of the integer that was passed to the program. The value of the foreign key `counter` will then be the `id` of the new counter just inserted. Finally, to allow CÉCI's administrators to define the fields that will be required by the new project type, we decided to include the arguments `cpu_time`, `disk_space`, and `scratch_space`. If these values are passed to the script, the attributes corresponding to the indicated arguments will be set to one in the `project_type` table. However, if they are not specified, the default value of zero will be inserted.

Subsequently, we had to adapt the `ProjectType` class to match the new table. We added the `counter`, `disk_space`, `cpu_time`, and `scratch_space` attributes to this class. The `ProjectCounter` class also had to be modified to allow the counter specific to a project type to be incremented. To do this, we added a static method in it. The latter is used to generate the `pid_number` of a project according to its type when a project must be pushed into the LDAP. Thereafter, this value will also be used to define the attribute `gidNumber`. Finally, when we define the description of the group using a dictionary, we get the record from the `project_type` table whose `id` corresponds to the attribute `type` of the current project. We insert the keys `cpu_time`, `disk_space`, and `scratch_space` only if the values of these attributes are set to `true` in the record that was just retrieved.

3.3.3 Front-End Implementation

Now that we have described the different modifications we made to the server-side of the application, we will detail the different changes we have implemented on the interface side of the application.

To adapt the front-end part, we had to modify the `relog.js` file. The code representation of the data and operations on a UI for the project (the view model) is maintained in an object defined by the class `project`. The first step was to make it possible to load the different project types containing the fields `id`, `name`, `eula` as well as the new fields `disk_space`, `cpu_time`, and `scratch_space` indicating for each project type the required fields. To do this, we load all project types using AJAX requests that we send to the server. Each project type is then stored using a dictionary into the observable `available_types`.

The second step was to create `specific_type_fields` observable and the `specific_fields` computed observable. The latter is a function that depends on the observable `available_types` and `type`, and will be automatically updated whenever any of these dependencies changes. The observable `specific_type_fields` will be used to store an object whose keys indicate the type of fields and whose values indicate whether these fields should be displayed in the form. This will be updated each time the `specific_fields` function is called and will take the value of the selected project type. Finally, it is important to understand that when a user changes the type of project he wants to create, the observable `type` will be notified of the changes and will be updated accordingly. As a result, the `specific_fields` function will be called as one of its dependencies has changed, and therefore, the observable `specific_type_fields` will be updated and will contain the fields to be displayed for the selected project type.

Finally, we had to modify the views of the `project.html` and `board.html` files. Concerning the changes we made in the project file, we added a `div` tag for each field in the "set up details" section. We used the visible binding system provided by `knockout.js` to make the associated DOM element for each of these `div` tags hidden or visible according to the value passed to the binding. Therefore, the DOM element for each of these `div` tags becomes visible (invisible) if the value of the key corresponding to the field in the observable `specific_type_fields` is `true` (`false`). For the modifications of the `board.html` file, we proceeded in the same way. However, we made sure that the field containing the project type could not be modified by the administrators.

```

1 <div data-bind="visible: specific_type_fields().disk_space">
2   <div class="form-group" data-bind="css: 'has-' + errors().disk_space">
3     <label class="control-label" for="disk_space">
4       Requested permanent disk space
5     </label>
6     <input type="number" min="0" class="form-control focus_popover"
7           id="disk_space" placeholder="100"
8           data-bind="value: disk_space, valueUpdate:'keyup'"/>
9     <p class="help-block">
10       Maximum shared disk space in GB the project will need
11     </p>
12   </div>
13 </div>

```

Listing 3.1: Example of modification made to the `project.html` file.

3.3.4 Use Case Scenario

Now that we have described the changes we have made on the server-side and the user interface, we will illustrate with different scenarios, the different views we can get for the new forms. For a more precise description of the various changes concerning the administration part, see the user documentation in Appendix B.1.

To start with, we will create two new project types using the command line interface and the `create_project_type` script we implemented. The first type of project that we will create could correspond to the "Storage Management" project type. For this type of project, we want it to require no CPU hours and scratch space, but only information on the storage space that the project needs. The second project that we will create could be applied for "Managing a Linux group" project type. This type of project will be basic and won't require CPU hours, scratch space, and storage space.

To do this, we will execute the commands: `"./create_project_type example2 https://www.example2.html 5000000 --disk_space"` and `"./create_project_type example3 https://www.example3.html 4000000"`.

project_type

id	timestamp	name	eula	disk_space	cpu_time	scratch_space	counter
1	2014-12-..	TIER1	https://..	1	1	1	1
2	2019-12-..	example2	https://..	1	0	0	2
3	2019-12-..	example3	https://..	0	0	0	3

projectid_counter

id	last_uid
1	6000000
2	5000000
3	4000000

Once these commands are executed, we can see in the `project_type` and `project_tid_counter` tables of our database that two new records have been added to each table. The values of the attributes `cpu_time`, `scratch_space`, and `disk_space` have been set to zero except for the attribute `disk_space` of the project type `example2`. This is due to the fact that we passed the `disk_space` attribute as an argument when executing the `create_project_type` script. The two new counters were automatically added, and their `last_uid` attribute values are 4,000,000 and 5,000,000, which are the values we passed to the script. Finally, we can notice that the link between the new project types created and their counter is done through the `counter` attribute of the `project_type` table.

With regard to the form to be filled in when creating a project, we can choose the type of project we want to create in the "identify the project" section, and the fields in the "provide a description" remain the same regardless of the type of project. Only the fields of the "setup details" change.

1. Identify the project

Project Type:

Title:

Acronym:

The acronym must be between 4 and 8 characters long and can only contain lowercase letters

Project Category:

3. Setup the details

Requested permanent disk space:

Maximum shared disk space in GB the project will need

Expiration date:

Example: 2018-12-31

4. Submit!

Once the form is completely filled-in, click the "Save" button. A system administrator will review your request.

If you want to save the current data and submit it later, click the "Save" button.

1. Identify the project

Project Type:

Title:

Acronym:

The acronym must be between 4 and 8 characters long and can only contain lowercase letters

Project Category:

2. Provide a description

Short description of the project:

Please provide as much information as you can such as the objective of the research, the funding source, partners, and a description of the resources you will need. (max 10000 chars)

3. Setup the details

Expiration date:

Example: 2018-12-31

4. Submit!

Once the form is completely filled-in, click the "Save" button. A system administrator will review your request.

If you want to save the current data and submit it later, click the "Save" button.

(a) *Share storage* type

(b) *Managing linux group* type

Figure 3.1: Example of a possible layout for the project creation form.

In Figure 3.1a, we can see the layout of the form for one of the new project types we have created, and which can be applied to the "Share storage" type. To do this, we simply had to select the type `example2`, and the form adapts to match the selected type. As a result, the fields *requested CPU hours* and *requested temporary disk space* are invisible, and their values will not be transmitted to the server when the form is sent or saved.

In Figure 3.1b, we can see the layout of the form corresponding to the second project type created, this visual could be applied to the project type "managing Linux group" in which only the *expiration date* and *users* fields of the "setup the details" section would be visible and could be completed by a user.

3.4 Software Testing

Software testing, as we know, is a very important point, not only to ensure the quality of the software but also to prevent potential security and software failures. This is why, in an application that stores personal data of its users such as Relog, testing must require particular attention. This section will be dedicated to the software testing we have put in place to ensure the validity of our implementation.

3.4.1 Testing Approach

First of all, we had to choose the software testing methodology we were going to use to test the different functionalities we had added to the Relog application. To begin with, we decided to focus on a white-box testing approach at the unit level to make sure that the individual sections of our code worked properly. Once we ensured that the individual parts we implemented produced the expected results, we focused on integration testing. To do this, we used our virtual machine development environment to test different software interactions of the Relog application when the latter was integrated with some of its components.

3.4.2 Unit Testing

Unit testing refers to tests that verify specific sections of individual units or components of the software. The advantage of this approach is that it allows us to focus on small sections of the code, and therefore, to concentrate mainly on the modules and functions that have been modified during the development of our feature.

Before implementing our different unit tests, it was important for us to respect the different quality criteria of the unit tests [22]. A large part of the implementation choices was influenced by these criteria below, as we will see in the next subsections.

- Isolated: One of the most important principles that must be respected by the unit tests, the latter stipulates that unit tests must be independent of each other. If one unit test should fail, this should not have any impact on the other tests. If this principle were to be broken, one of the main advantages of

unit tests, which is that they clearly indicate the source of the error, would be lost.

- Self-validating: Unit tests alone are sufficient to tell whether or not a test has passed. They should not require human interaction for this.
- Fast: Unit tests must be able to be executed quickly and not take more than a few seconds to run.
- Test a single unit of work: in order to avoid having to adapt a lot of unit tests when changes are made to the application, each unit test has to test small pieces of code.
- Well-factored: Like all codes, unit tests can be modified or adapted by other developers, so well-written and maintainable tests are essential.

Database Integration

In order to implement our different unit tests, we had to choose how we would integrate them with `Python` and `SQLAlchemy`. One of the sensitive points was the integration of the database and the way we were going to manage it. Indeed, to test the different functions related to the different types of projects, it was imperative to have a pre-filled database. Another important point we had to pay attention to, was to make sure that the different changes in the database states during the execution of a test did not affect the results of the other tests. If this were to happen, the *isolated* principle would be violated.

To solve this problem, the approach we used is composed of the different steps below.

1. Before running all unit tests, a database in memory is first generated. This database will be fed only with the necessary data common to all the tests. For example, we include certain users and administrators as well as different institutions.
2. When we run a test, we need to set up the data required by it, and when the latter has finished running, clean up the data that has been inserted for the next test. A naive approach would be to clean up the database using SQL queries. Although this solution is feasible, this approach is inefficient as it would result in an additional lot of code that would be difficult to maintain and reuse. A second possible approach that we have chosen is to use transactions, a feature available in SQL that can be defined as *"a sequence of operations performed on a database as a single logical unit of work. The*

effects of all the SQL statements in a transaction can be either all committed or all rolled back" [23]. As we can see on Listing 3.2, for each unit test, we will run them in isolated transactions and apply a rollback once they are completed.

```

1 @pytest.yield_fixture(scope='function')
2 def session(ctx):
3     """ Yield a specific session each time a unit test requests it
4     """
5     conn = ctx.engine.connect() # connect to the common database
6     transaction = conn.begin()
7     with ctx.session(acl_key=None, admin_email=None, conn=conn) as session:
8         setup_factories(session)
9         yield session # session that will be used by the test
10    transaction.rollback() # undo all SQL statement effects performed by the test

```

Listing 3.2: Session function of the module `conftest.py`.

3. As seen previously, before executing a unit test, we must generate the data necessary for it. To avoid using a lot of hard-coded data, and consequently breaking the *well-factored* principle by making our code difficult to maintain, we chose to use the `factory_boy` tool [24] to define different factories for our different SQLAlchemy models. On listing 3.3, we can find an example of a factory of our `ProjectType` model that we defined.

```

1 class ProjectTypeFactory(BaseFactory):
2
3     name = factory.Sequence(lambda n: 'type%s' % n)
4     eula = factory.LazyAttribute(lambda obj: 'http://www.cecil-hpc.be/%s/rulescenaero.html' % obj.name)
5     counter = factory.Sequence(lambda n: n * 1000)
6     disk_space = fuzzy.FuzzyInteger(0, 1)
7     cpu_time = fuzzy.FuzzyInteger(0, 1)
8     scratch_space = fuzzy.FuzzyInteger(0, 1)
9
10    class Meta:
11        model = ProjectType
12        force_flush = True
13
14    class Params:
15        tier1 = factory.Trait(
16            disk_space=1,
17            scratch_space=1,
18            cpu_time=1,
19        )

```

Listing 3.3: Factory of the `ProjectType` class.

Test Implementation

To stay in line with the principle of *"test a single unit of work"*, we have divided our unit tests into several modules, each of them being responsible for testing a specific function of our application. Within these modules, we find our different unit tests covering a specific case of the function tested by the module. For example, in the module `test_project_set_status.py`, we find the different tests concerning the `set_status` function of the `Project` class.

```

1 def test_set_status_validate_draft_invalid_acl_key(init_draft_project):
2     project, active_user, session = init_draft_project
3     ctx = models.CTX
4     ctx.set('role', 'public')
5     with pytest.raises(Exception):
6         project.set_status("waiting")
7
8
9 def test_set_status_validate_draft_valid_acl_key(init_draft_project):
10    project, active_user, session = init_draft_project
11    ctx = models.CTX
12    ctx.set('role', 'public')
13    ctx.set('acl_key', project.acl_key)
14    result = project.set_status("waiting")
15    assert result is True
16    assert project.status == 'waiting'
17
18
19 def test_set_status_confirmed_role_key_auth(init_draft_project):
20    project, active_user, session = init_draft_project
21    ctx = models.CTX
22    ctx.set('role', 'public')
23    ctx.set('acl_key', project.acl_key)
24    with pytest.raises(Exception):
25        project.set_status("confirmed")

```

Listing 3.4: Example of tests present in our `test_project_set_status` module.

3.4.3 Integration Testing

During the development of this feature, we had to ensure that the interactions between different components worked properly. We can take the example of the interactions between the Relog application and its LDAP when projects of different types are inserted or with its `lighttpd` web server. To test how the system parts of the Relog application work together, we opted for an integration testing method using a functional type approach. The latter is a type of black-box testing in which the functionality of our application is tested to ensure that the requirements of our software are satisfied.

To do this, we developed 26 manual test cases, each covering a slice of functionality in the system and interactions between components. Integration testing allowed us to fix some bugs that we wouldn't have been able to discover with the unit tests. The complete list of test cases is available in [Appendix B.4](#).

3.5 Summary

In this chapter, we have provided a solution to the "Adding new project types" issue that allows CÉCI administrators to define directly in CLI new project types that must be created on the Relog application. We ensured the quality of our implementation by using unit tests and by making sure that the latter respect the best practices, and finally, we tested the interactions between the different modules used by the application through integration testing.

Chapter 4

Contribution 2: Implementation of a Tool to Extract Demographic Information

In this chapter, we have developed a tool to generate reports using information extracted from the database of the Relog application. The development of this module is a response to issue 48, which was opened on the CÉCI Gitlab, namely, *"Extract demographic information from the database and produce reports"*. The complete software documentation can be found in Appendix [C.1](#).

4.1 Context

Before going into details about the implementation and the architecture choices we made, we will explain the background as well as the underlying problem that led CÉCI members to open this issue.

Currently, the CÉCI provides to its administrators on its web page a set of graphs showing the utilization of its clusters [\[25\]](#). For example, we can find the timeline graphs of the CPU use of CÉCI clusters, filter by clusters or by users, bar charts displaying the number of jobs according to the different categories of projects, etc. Several types of charts are also available, which allows better data visualization according to the information to be processed.

It is within this context that we are going to intervene and be able to contribute. Indeed, the data that are collected mainly concern the use of clusters. The challenge will be to develop a tool to collect other data in order to generate reports that can be integrated on CÉCI's site. The new data in question are those related to the

CÉCI institutions, and more particularly, the data on the number of registered and expired users as well as information on projects.

Initially, to meet this need, the solution that emerged after our meeting with Dr. François was to develop a tool that could generate Portable Document Format (PDF) reports. These reports were intended to provide a clear visualization of the demographic data extracted from the database by displaying various graphs based on this information. Subsequently, these PDF reports were to be made available to CÉCI's administrators by allowing the latter to download them directly from CÉCI's website.

4.2 Design Choice

To propose an optimal solution to the problem, we first had to define the software design, the different technologies to use, as well as the structure to adopt. The idea that we initially opted for was based on the `NumPy` [26] and `Matplotlib` [27] `Python` libraries. Although the latter offers graphic tools for creating static and animated visualizations that are very comprehensive and easy-to-learn, we quickly realized that this approach was not the most suitable given that the customization they offered to generate reports was far too limited.

After a second meeting with Dr. François in which we discussed the problem. We decided to rethink and redefine the solution initially proposed. In order to optimally meet CÉCI's needs, we decided that it was preferable to generate HTML reports directly. Indeed, working with HTML files offers a great possibility of customization, such as the use of `JavaScript` libraries for an optimal and reactive visual rendering of the charts, which would not have been possible when working with PDF reports. In addition, HTML reports can be integrated directly into the CÉCI website without the need for administrators to download anything. This approach was the one we put into practice and which we will discuss in the rest of this chapter.

For the second solution, we have chosen to extract the necessary information from the database in JSON format. Then, this data is used to create HTML reports using a `Python` web template engine, and finally, the use of an open-source data visualization `JavaScript` library allows us to process and display the data in an optimal way.

4.2.1 Technology Stack

The choice of different technologies to solve a particular problem is crucial. In this subsection, we will detail the technologies we have used and the reasons that led us to choose them.

For the template engine, we have opted for `Jinja2` [28], which is a modern and full-featured template engine for `Python`. Although it was not the only option available, the main advantages of the latter are as follows:

- **Security:** One of the points we found important is that it has automatic HTML escaping tools, allowing us to be sure to prevent any possible cross-site scripting attacks.
- **Template inheritance:** To avoid code duplication, one of `Jinja`'s features is the template inheritance, which allows building basic templates that child templates can override. We have exploited this property a lot, as we will see later.
- **Fast and compatibility:** Finally, the last reason is that this template engine is very fast and compatible with a wide range of `Python` versions.

Concerning the chart `JavaScript` library, we decided to use the `Chart.js` [29] library for several reasons:

- **Lightweight and fast:** The library is very light, avoiding a too important loading time of our different pages.
- **Full responsive:** The library contains six different types of charts, and each chart is fully responsive, which allows a beautiful visual rendering.

4.2.2 Software Architecture

In this subsection, we will explain the software architecture that has been decided as well as the project structure.

Component Interaction

As we can see in the sequence diagram below 4.1, the application is divided into several modules, each of them handling a specific task. First, we decided to isolate and separate the modules for extracting demographic information from the database, the main module, and the templates. A shell script is responsible for

calling our different modules in a well-defined order. It is the latter, which serves as the entry point to the application and which is called by a CÉCI administrator. This shell script will first execute the commands to extract the demographic information from the database, once the data is extracted, a command will execute the main module that will generate our different pages of the report. For each of them, the latter will load the associated template and render it by injecting the data retrieved in the JSON files created previously. It should be noted that the template is rendered by taking into account a configuration file (see Section 4.4 for more details). Finally, the last command executed by the shell script allows creating an archive with the generated HTML files and the assets folder.

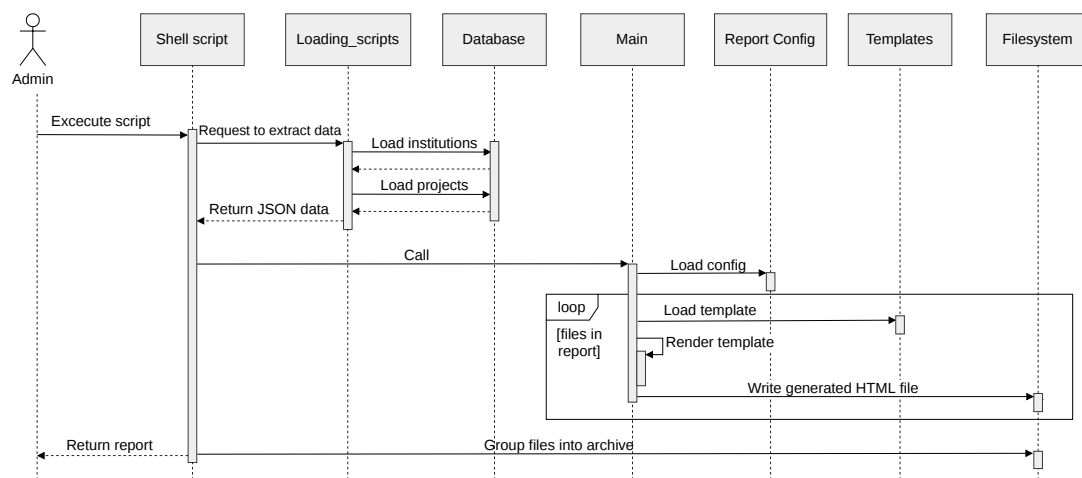


Figure 4.1: Sequence diagram of our different module interactions.

Project Structure

We will now focus on the software structure we have chosen and that can be found on Listing 4.2.

To begin with, the extraction of information related to the CÉCI institutions and the CÉCI projects is done via the scripts `load_institutions.py` and `load_project.py` of the folder `loading_scripts`. These will first retrieve from our `extract_demo_config.json` file the necessary information such as the start and end date of the period that has been set and that the report should cover. Subsequently, these scripts will generate from the database located in the `var` directory, the JSON files following a well-defined schema, and as seen previously, these are the ones that will be used by our `main.py` module to generate the report.

One of the reasons we opted for such a choice of architecture by separating and isolating the loading modules from the main module was to adhere to the single responsibility principle. Indeed, by making sure that the communication is done through JSON files respecting a very precise schema, the module responsible for rendering the templates and generating the report is isolated from the rest of the application. Only the loading files communicate with the **Relog** database, and if the latter should evolve, only the loading modules will have to be adapted.

Concerning our different templates used by the `main.py` module, these are located in the `templates` directory. These are defined using template inheritance with a skeleton that can be found in the `base_template.html`. The latter defines the HTML skeleton that will be shared by all files in the report. Concerning the `institutions` sub-folder, this one contains the `universities_template.html` and `low_level_institutions_template.html` templates, which will be used to generate the files of the report related to the institutions. The file `institutions_template.html` serves as a template parent for the latter. Regarding the `projects` folder, it contains the file `projects_template.html`, which is the template that will be used to create the `projects.html` file.

It would be interesting to illustrate the advantage of the hierarchical structure that has been chosen for the different templates using an example. We can see on Listing 4.1 part of the code used in the file `base_template.html`. The latter allows us to define a section and a footer common to all our files. On the other hand, our different child templates will be able to override the `content` block. As a result, we can organize the different layouts in several layers, which allows better maintainability and reusability.

```

1 <section class="module-small">
2   <div class="container">
3     <div class="row">
4       <div class="col-sm-12 text-center">
5         <h1 class="mh-line-size-3 font-alt m-b-20">{{ config["title"] }}</h1>
6         <h5 class="mh-line-size-4 font-alt">{{ config["secondary_title"] }}</h5>
7       </div>
8     </div>
9   </div>
10 </section>
11
12 {% block content %}
13 {% endblock %}
14
15 <footer class="module bg-light">
16   <div class="container">
17     <div class="row">
18       <div class="col-sm-12">
19         <p class="copyright text-center m-b-0">
20           &copy; 2020 &#x00C9; All Rights Reserved.
21         </p>
22       </div>
23     </div>
24   </div>
25 </footer>

```

Listing 4.1: Code part present in our `base_template.html` file.

Finally, when we developed the application, it was important that the report it generates could be parameterized. For this reason, we have included the configuration file `extract_demo_config.json`.

```

.
├── assets
│   ├── bootstrap
│   ├── css
│   ├── fonts
│   └── js
├── extract_demo_config.json
├── extract_demographic_info.sh
├── loading_scripts
│   ├── load_institutions.py
│   └── load_projects.py
├── main.py
├── templates
│   ├── base_template.html
│   ├── institutions
│   │   ├── institutions_template.html
│   │   ├── low_level_institutions_template.html
│   │   └── universities_template.html
│   ├── macros.html
│   ├── projects
│   └── projects_template.html
├── var
└── db.sqlite

```

Listing 4.2: Directory tree structure of our project.

4.3 Use Case Scenario

Now that we have a better idea of the structure of the application and how it works. We will show an example of a report that can be generated.

The first thing to do is to set the different options of the report using the configuration file. In the example below, we will consider a basic configuration with a single theme for all the charts. After that, we can run our shell script `extract_demographic_info.sh` directly from the command line. Once the report has been generated, the `report_demo_info.tar.gz` archive will be created. In this one, we will be able to find all the files of the report (a full description of the files contained in it can be found in the documentation provided in Appendix C.1).

In Figure 4.2, we can find the page of our report relating to CÉCI's partner universities. We can see in the latter, various information regarding the number of users of each university's clusters. In the introductory section, we have included two charts, one showing the number of active users from each university, and the other showing the total number of registered and expired users from all universities.

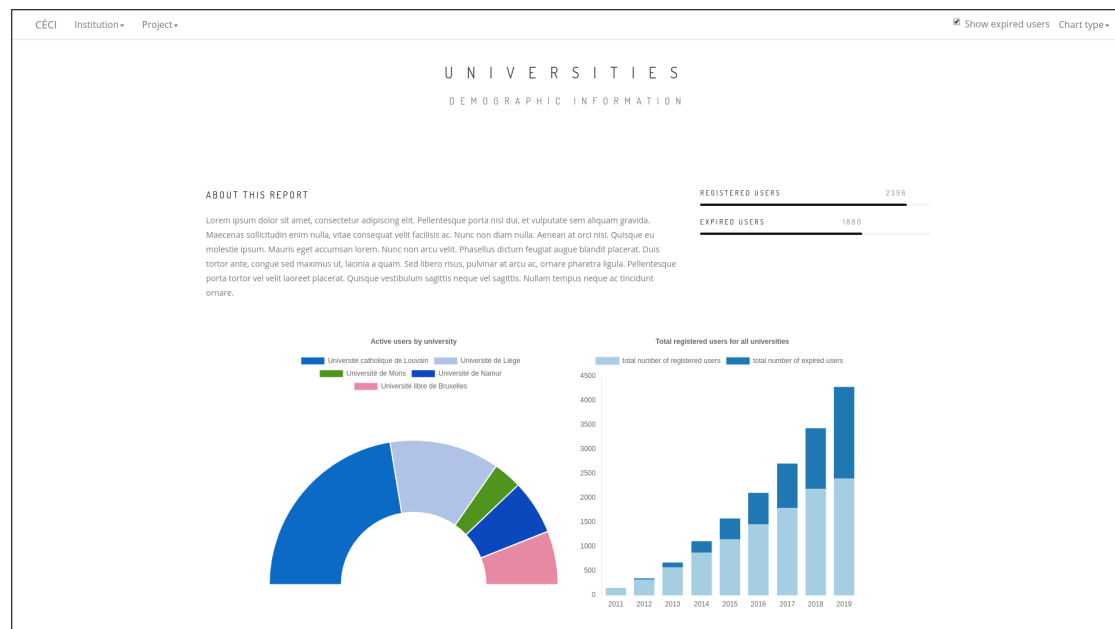


Figure 4.2: Layout of the top section of the page `university.html`.

In the bottom section 4.3, the set of charts for each university is available, for each of them, the user can update the type of charts dynamically using the drop-down menu in the navigation bar.

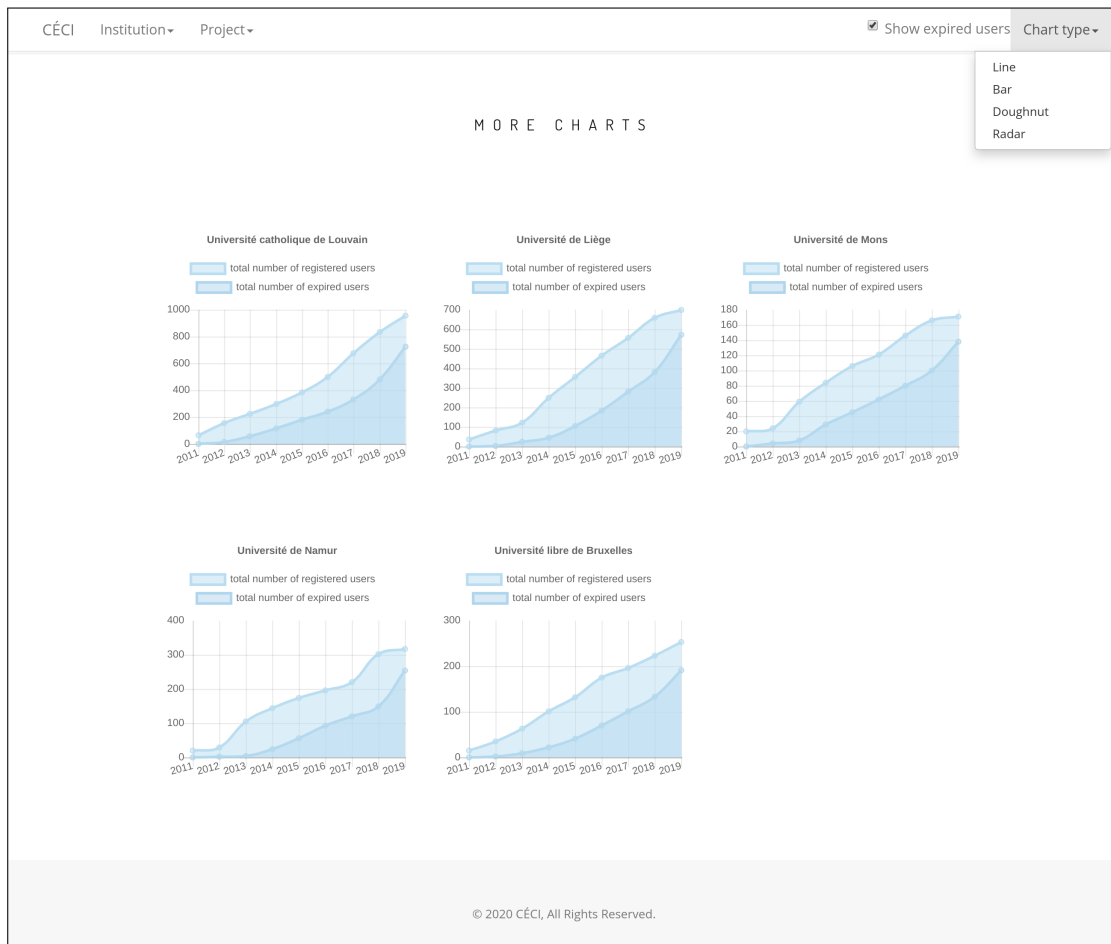


Figure 4.3: Layout of the bottom section of the page `university.html`.

4.4 Configuration

Now that we've shown an example of a report that can be generated, it would be interesting to talk about the configuration file that we decided to integrate into the application.

4.4.1 Parameterization Need

When we were developing this tool, one of the important points for CÉCI was that the report to be generated can be parameterized in a user-friendly way. The main reason is that it would allow harmonizing the report with the different pre-existing charts that we can find on the CÉCI pages. To illustrate this, we can take the case of the different color themes of CÉCI partner universities on the CÉCI

charts, and which must also apply to the charts of our report. Secondly, the use of a parameterization component makes it possible to redefine the parts of the report that are subject to regular changes, such as the different titles of each page, without having to modify the code.

4.4.2 Configuration Format

As we know, there is a multitude of language configurations, all with their advantages and disadvantages. When we implemented the tool for extracting demographic information, we wanted to include a configuration file that would be portable and could be easily reused or adapted to meet CÉCI's future needs. For this reason, we chose to use the JavaScript Object Notation (JSON) data format.

Concerning the structure of our configuration file, the latter contains the objects **university**, **institute**, **pole**, and **project**. Each of them contains different key-values allowing us to parameterize a specific part of the report to be generated. On Listing 4.3, we can see an example of the structure used in our configuration file, and more particularly, the key-values of the **university** object.

```
1 {
2   "json_institutions_name": "data_institutions.json",
3   "json_projects_name": "data_projects.json",
4   "starting_year": 2011,
5   "end_year": 2019,
6   "university": {
7     "document_title": "Universities – Registrations",
8     "title": "Universities",
9     "secondary_title": "Demographic Information",
10    "display_introduction_section": true,
11    "introduction_title": "About This Report",
12    "introduction_text": "Lorem ipsum dolor sit amet, consectetur adipiscing.",
13    "additional_chart_type": {
14      "bar": true,
15      "doughnut": true,
16      "radar": true
17    },
18    "color": {
19      "default": "tableau.Blue20",
20      "ucl": "tableau.Blue20",
21      "ulg": "office.Blue6",
22      "umons": "tableau.OrangeGold20"
23    },
24    "active_users_color": {
25      "ucl": "#997119"
26    }
27  },
28  ...
29 }
```

Listing 4.3: Content example of our configuration file.

4.4.3 Configuration Options

In this subsection, we will present some of the options that can be used to generate the report. The complete list of options is available in Appendix C.1.

- **additional_chart_type:** This option is used to activate the different chart types that the user will be able to choose to visualize the information. When a specific type is activated, the user will be able to dynamically change the type of the charts using a drop-down defined in the navigation bar of the institutions pages. The line chart is the default type, and the latter cannot be deactivated. In Figure 4.4, we can see the types of charts that can be used.

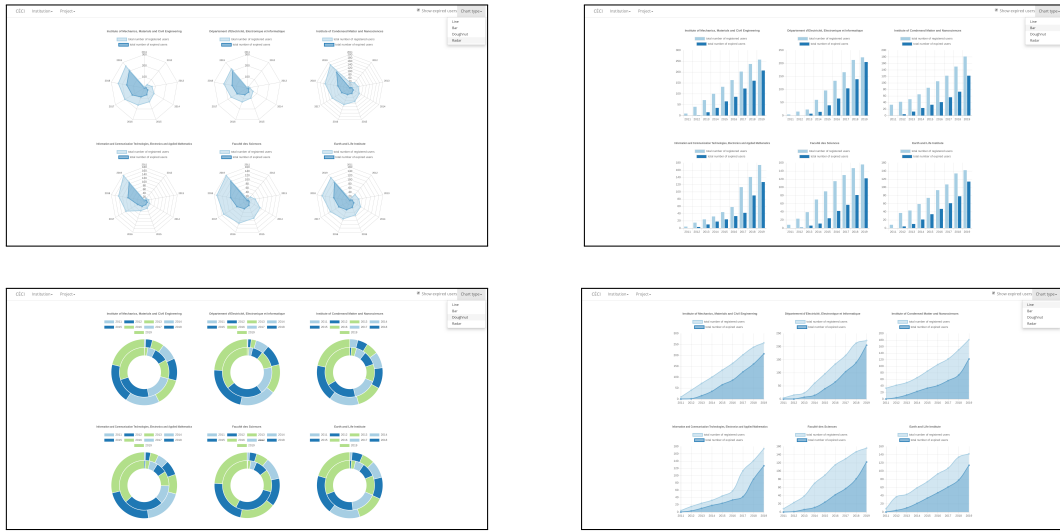


Figure 4.4: Layout of the different types of charts of the page `institutes.html`.

- **starting_year/end_year:** Start and end dates of the period over which the data will be collected and covered by the report.
- **display_introduction_section:** This option is used to display the introduction sections of the university and project pages of the report. The latter includes additional graphs such as the chart of the number of active users from each university and the number of projects according to the different categories.
- **color:** This option allows CÉCI administrators to define a color scheme for universities or all poles/institutes belonging to a certain university. To do

so, administrators can choose a theme to apply to the charts from a list of predefined color schemes and specify it in the configuration file. In Figure 4.5, we can see the section of the report indicating the institutions with the most registered accounts and in which color themes have been applied for the institutions belonging to Université catholique de Louvain, Université de Liège and Université libre de Bruxelles.



Figure 4.5: Layout of the top section of the page `institution.html` in which color themes have been applied for institutions belonging to the UCLouvain (red), ULiège (yellow) and ULB (blue).

4.5 Summary

In this chapter, we have implemented a tool to generate reports using information extracted from the database of the Relog application with the Jinja2 web template engine and the `Chart.js` library. We have also made it possible for CÉCI administrators to easily parameterize the different reports to generate. Although there is always room for improvement, this implementation meets the requirements originally planned.

Chapter 5

Contribution 3: Security Audit

Web applications are particularly exposed to the risk of cyber-attacks [30], which can pose a significant threat to the latter, such as the disclosure of secure or confidential information. In an application like Relog, which provides access to high-performance computing resources, the impact of a potential attack would be all the greater.

In the case of a vulnerability in the Relog application that could be exploited by attackers, the latter would have access not only to the various CÉCI clusters but also to confidential information present on them. To illustrate this, we can take the example of a cyber-attack such as SQL injections or Cross-site Request Forgery (CSRF) in which an attacker would succeed in replacing a user's email address with one to which he has access. The attacker could then ask the CÉCI administrators to generate a new SSH key associated with the account from the CÉCI site, and this SSH key will be sent directly to the new email address. Finally, by connecting to the CÉCI clusters using his new key, the attacker will have access to all the information that has been previously stored by the user on the clusters.

For this reason, we decided, in agreement with CISM, to devote part of this thesis to a security audit of the Relog application. In this chapter, we present the results of this audit, which focused on four themes. The first was an analysis of SQL injection risks in which we were unable to identify any potential flaws. The second focused on cross-site request forgery attacks and allowed us to highlight the lack of protection of the application, such as the use of anti-CSRF tokens, to prevent this type of threat. The third was an evaluation of the risks associated with cross-site scripting attacks. The last one consisted of using web application vulnerability scanners to detect possible flaws in the Relog application. This allowed us to detect the risk of clickjacking attacks, which could be easily prevented by the use of the `X-Frame-Options` header by the web server.

5.1 SQL Injection

A SQL injection attack consists of the insertion of a SQL query via the input data by an attacker on a website to perform an action on the database, such as reading or modifying information, and for which he would not normally have been authorized [31]. The use of object-relational mapping is a good way to protect against SQL injection. `SQLAlchemy`, which has been chosen as ORM for the `Relog` application, is not an exception to the rule. Indeed, when used properly, the different methods that `SQLAlchemy` offers us to execute our queries are designed to avoid SQL injection attacks. However, like all libraries, `SQLAlchemy` has vulnerabilities.

5.1.1 SQL Injection in SQLAlchemy

`SQLAlchemy` provides an SQL expression language [32] as an API to execute our queries using `Python` constructs. The latter uses the `bind parameters` method to pass the different data to the database, and thus, any special character of the user-generated values will be automatically quoted by the `SQLEngine` object [33]. When using SQL expression language, SQL injection attacks are almost impossible.

`SQLAlchemy` also allows executing raw SQL statements which, contrary to the use of SQL expression language, can cause security vulnerabilities. However, the `Relog` application does not use this method to execute its queries. To find potential flaws that could lead to SQL injection attacks, we will have to analyze the known vulnerabilities of `SQLAlchemy`.

5.1.2 Security Vulnerabilities of SQLAlchemy

In this subsection, we will analyze the known vulnerabilities of the object-relational mapper `SQLAlchemy` and see the impact they can have on the `Relog` application. On the site *CVEdetails* [34], which is a free CVE security vulnerability database source, we can see two known vulnerabilities for version 1.1.14 of `SQLAlchemy` and the latter are of SQL injection type (see CVE-2019-7548 [35] and CVE-2019-7164 [36]).

The first vulnerability is defined as *"SQLAlchemy has SQL Injection when the group_by parameter can be controlled"* [35] and has an impact on the confidentiality as it allows access to information to which a user would not have access rights. To illustrate this, we will make an HTTP request with a POST method to return the list of available CÉCI institutions. The endpoint we are going to use to access the resources is the following *https://login.myceci-hpc.be/_rpc/institution/read* and is accessible to the public. The `Relog` application uses parameters in the request

body to process and return the right information to the client. On Listing 5.1, we have an example of data in the request body that can be sent by a client to the Relog application.

```
1 {
2   "args": [],
3   "kwargs": {
4     "fields": ["id", "name", "parent", "levels"],
5     "filter_by": {},
6     "limit": null,
7     "order_by": ["id"]
8   }
9 }
```

Listing 5.1: Example of data in the HTTP request body that can be received by the Relog application.

In the example above, the property **fields** allows us to indicate the different attributes of the institutions we want to retrieve, and the property **order_by** is used to specify that we want the results to be sorted according to the attribute **id**. On Listing 5.2, we can find the response returned by the server to the query in which we can see the list of institutions ordered by their **id**.

```
1 [
2   [
3     1,
4     "Laboratoire de Biologie des Organismes Marins et Biomimétisme",
5     106,
6     null
7   ],
8   [
9     2,
10    "Institut de Recherche Interdisciplinaire en Biologie humaine et moléculaire ",
11    30,
12    null
13  ],
14  ...
15 ]
```

Listing 5.2: Response of the Relog application to the request 5.1.

After having made slight modifications to the back-end of the Relog application, we will exploit the vulnerability to retrieve the name of the user with **id** 5 from the **user** table. To achieve this, we will make the same query as before, but we will replace the **order_by** property by a SQL CASE statement (see Listing 5.3).

```

1 {
2   "args": [],
3   "kwargs": {"fields": ["id", "name", "parent", "levels"],
4   "filter_by": {},
5   "limit": null,
6   "order_by":
7     [
8       "CASE
9         WHEN (
10           SELECT unicode(substr(email,1,1))
11           FROM user
12           WHERE id='5'
13         )=108
14         THEN name
15         ELSE id
16       END"
17     ]
18 }
19 }

```

Listing 5.3: Example of data that can be used to exploit the CVE-2019-7548 vulnerability.

Depending on the result returned, we can find out if our condition has been verified. In the previous example, we wanted to know if the user having the identifier 5 has as value for the first letter of his email the small Latin letter "L" evaluated at 108 by the `unicode` function. The email of the latter being *laurent.*****@uclouvain.be*, the application has returned a JSON of the different institutions ordered alphabetically according to their names (see Listing 5.4). If our condition is not verified, we just have to try other values until we get an alphabetically ordered response. Finally, by testing each letter individually, we can find the complete value of the attribute.

```

1 {
2   [
3     [
4       299,
5       "ANAST - Systemes de transport et constructions navales",
6       201,
7       null
8     ],
9     [
10      9,
11      "Administration Systeme",
12      51,
13      null
14    ],
15    [
16      135,
17      "Aeroelasticity and Experimental Aerodynamics",
18      187,
19      null
20    ],
21    ...
22  ]
23 }

```

Listing 5.4: Response of the Relog application to the request 5.3 with institutions ordered alphabetically according to the `name` attribute.

To verify if this vulnerability can be exploited, we have to look at the source code of the Relog application. The only function that uses the `order_by` function

offered by the Query API of `SQLAlchemy` is the method `read`. The latter is a generic function and is used by the back-end of the application each time a user wants to perform read access to resources stored in the database.

```

1  @classmethod
2  def read(cls, fields, filter_by={}, order_by=None, ilike=None, limit=10):
3      session = CTX.get('session')
4
5      query = session.query(cls).filter_by(**filter_by)
6
7      if order_by:
8          order = []
9          for n in order_by:
10             if isinstance(n, list):
11                 item = getattr(cls, n[0])
12                 if n[1] == 'desc':
13                     item = item.desc()
14             else:
15                 item = getattr(cls, n)
16             order.append(item)
17             query = query.order_by(*(order))
18  ...

```

Listing 5.5: Part of the code of the `read` method of the file `models.py`.

On Listing 5.5, we can see that the values in the `order_by` field that are present in the body of the HTTP request are not directly passed as parameters to the `query.order_by` function. Instead, they are retrieved directly from our various models using the `getattr` function. As a result, a value that is not an attribute of the model corresponding to the resource requested by the user would raise an `AttributeError`. It is therefore impossible to exploit this vulnerability to access the different data of the Relog application.

Concerning the CVE-2019-7548 vulnerability, this one is defined as *"SQLAlchemy has SQL Injection when the group_by parameter can be controlled"* [36]. This vulnerability is quite similar to the previous case, however, the Relog application does not use the `group_by` function making the vulnerability non-exploitable. Nevertheless, in order to prevent any potential risk if the application were to evolve, we recommend checking the input data provided by users, as it is currently the case, and to regularly update the version of `SQLAlchemy` used by the application.

5.1.3 SQL Penetration Testing

Finally, to ensure that no SQL injection flaws can be exploited, we have used `sqlmap` [37] which is an open source penetration testing tool written in Python that automates the process of detecting SQL injection vulnerabilities.

To test the Relog application, we have targeted different URLs with POST methods. For each of our tests, the results showed that none of the request body

parameters appeared to be injectable. In Appendix D.1, we can see an example of a test we performed with the `sqlmap` tool.

Although caution should be exercised with the results obtained with the testing tools, as they are incomplete and may have weaknesses. These results reinforce the assumption that the chances of success of SQL injection attacks on the Relog application are unlikely.

5.2 Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF [38]) is an attack that forces an innocent victim to unintentionally perform actions, such as submitting a request, on a web application to which he has privileged access and is currently authenticated. Although in recent years, the popularity of CSRF-type attacks has tended to decline due to the increase in anti-CSRF protection [39], this type of attack remains common.

5.2.1 CSRF Vulnerability in Relog

CSRF attacks exploit the fact that browsers automatically include any identifiers associated with the site, such as session cookies, to inherit the victim's privileges on a particular site. In the Relog application, the authentication of the CÉCI administrators is done through client certificate authentication. This authentication method does not prevent the possibility of CSRF attacks since some browsers automatically add client-side certificates. As illustrated on Listing 5.6, if we make a simple cross-origin request from our browser to the Relog server in which we delete the user with `id 2884`, we can see that the request will be accepted by the server and that the client certificate will be automatically added.

```
1 | POST /_rpc/user/delete HTTP/1.1
2 | Host: login.myceci-hpc.be:449
3 | Origin: https://www.w3schools.com
4 | Referer: https://www.w3schools.com/xml/ajax_xmlhttprequest_send.asp
5 |
6 | {"args":["2884"],"kwargs":{}}
```

Listing 5.6: HTTP request sent to the Relog application originating from *www.w3schools.com*.

When we analyze the source code of the Relog application, the latter does not seem to be protected against CSRF type attacks. Indeed, as we will see in Subsection 5.4.1, the application does not use anti-CSRF tokens [40] and does not

compare the origin of incoming requests via the **Origin** or the **Referer** headers to know if the latter are reliable.

Another important point is that the Relog server does not check that the **Content-Type** header of the POST requests it receives is of type "application/json". This can make CSRF type attacks very simple to implement. To understand this, it is necessary to grasp the concept of Same-Origin Policy (SOP [41]). The latter can be broadly defined as a web browser security mechanism that prevents scripts on one web page from accessing data on another web page. The same-origin policy also allows restricting the requests that can be exchanged between sites of different origin to "simple requests" [42]. For this type of requests, the only values allowed for the **Content-Type** header are "application/x-www-form-urlencoded", "multipart/form-data", and "text/plain". To return to the previous example, an attacker could simply use the `XMLHttpRequest` object to send a JSON to the server using a "simple request" whose HTTP **Content-Type** header would have been set to "text/plain" 5.7. In such a case, the request will be sent directly to the server without the same-origin policy being able to prevent it.

```
1 <body>
2 <script>
3   function request() {
4     var xmlhttp = new XMLHttpRequest();
5     xmlhttp.open("POST", "https://login.myceci-hpc.be:449/_rpc/user/delete");
6     xmlhttp.setRequestHeader("Content-Type", "text/plain");
7     xmlhttp.send('{ "args": [2884], "kwargs": {} }');
8   }
9   request();
10 </script>
11 </body>
```

Listing 5.7: Example of code that could be used in the case of a CSRF attack to remove a user from the Relog application.

5.2.2 Mitigation of CSRF Risks

Finally, it should be noted that the risk associated with potential CSRF attacks is reduced due to the fact that the number of CÉCI administrators is relatively small. Moreover, these administrators have a high level of expertise, which decreases the chances of success of the techniques commonly used to get a user to make HTTP requests against his will, such as phishing.

5.3 Cross-Site Scripting

Cross-Site Scripting (XSS [43]) attacks are one of the most serious threats to web applications. In 2020, the Open Web Application Security Project (OWASP), an online community working to improve software security through freely available

materials, ranked this type of attack 7th in its top 10 most critical security risks for web applications [44]. In this type of attack, scripts are injected into the content of a web page, causing actions on the web browsers of the different users visiting the page. If an attack of this type were to be carried out on the Relog application, this could have negative consequences. We can take the following example of an attacker who, via the project creation form, would inject the code seen in the previous section 5.7 in the project description. If an administrator validates this project, this script could be executed via his browser, and thus cause the unintentional deletion of a user. In this section, we will analyze the cross-site scripting vulnerabilities of the Relog application.

5.3.1 Non-Persistent XSS

Non-persistent cross-site scripting attacks occur when inputs provided by a user are returned in the HTML pages to the user's browser by a web server and without the inputs having been sanitized. The Relog application does not render user-provided input directly to the browser. Even if query strings of URLs are used, for example, to check ACL keys, these are not rendered in the browser. Furthermore, the Relog application does not display directly search terms or possible information encoded in the HTTP headers. To find XSS vulnerabilities that can be exploited in the application, we will have to focus on the persistent type.

5.3.2 Persistent XSS

As explained in the Background chapter, the Relog application uses the `Knockout.js` library to build its web interface. To do this, a clear separation is defined between the different view models containing the UI's data and the data to be displayed. To connect the user interface of the Relog application to its underlying data model, the `Knockout`'s declarative binding system is used. All data from the application will first be assigned to the view models before being displayed in the user interface using the declarative bindings.

`Knockout.js` has many built-in bindings [45] to connect the view models to target DOM elements such as:

- **Text binding:** To display the text from a parameter of our view model in the associated DOM element.
- **HTML binding:** To display the HTML specified by a parameter of our view model in the associated DOM element.

- **Value binding:** To link a property on our view model with the associated DOM element, this binding is generally used with form elements.

Regarding the possibility of script injection of the different bindings, this depends on the type used. For example, we can read in the `Knockout.js` documentation [46] that text binding sets the text value to the associated DOM element using a text node and therefore is safe regarding script injection risks. On the other hand, when we use HTML binding, the latter sets the element's content using `innerHTML` and, as a result, may be subject to XSS attacks.

When we analyze the different views of the Relog application contained in the different HTML files, we can see that in order to display the different data fetched from the application, the text binding is mainly used. Nevertheless, there is one exception in the `project.html` file where an HTML binding is used to display information about the category of a selected project. The parameter specified in the binding refers to the computed observable `category_info` of the `project` model and depends on the data in the `project_category` table. The latter does not directly contain data stored by the users of the Relog application, and therefore, does not offer the possibility of cross-site scripting attacks.

Finally, we can then consider that due to the appropriate utilization of the different `Knockout`'s declarative bindings, the Relog application seems prevents all potential XSS attacks.

5.4 Vulnerability Scanning

While most security vulnerabilities can be found manually, there are other technical solutions to test web applications in a simpler and more automated way. In this section, we will focus on the web application security scanner.

A web application security scanner is a program that scans web applications through the end-user interface to identify security vulnerabilities [47]. In the following, we will perform a security audit of the Relog application using the w3af [48], Vega [49], and OWSAP Zed Attack Proxy (ZAP) [50] web application security scanners.

5.4.1 Scan Results

The results of the different web application security scanners were quite similar overall with alert severities ranging from low to medium. In Appendix D.2, we can

see one of the scanning reports we have obtained using the ZAP security tool. We will review some of the vulnerabilities detected and see what impact they could have on the Relog application.

Clickjacking Vulnerability

This vulnerability presents the highest level of risk among the vulnerabilities we have obtained. It has been detected by both w3af and ZAP software and has been assessed at a medium severity risk. This type of attack occurs when an attacker uses transparent layers to trick a user to click on a button or a link that is different from what the latter perceives [51].

This vulnerability could have an impact on the Relog application as an attacker could trick a CÉCI administrator into emitting an undesired action. To illustrate this, we can take the case of a malicious site using a transparent `iframe` to load the board page of the CÉCI web site reserved for its administrators. If a CÉCI administrator loads this page and clicks on it, the latter could issue an unwanted action, such as confirming or deleting a user.

To solve this vulnerability, the solution proposed by the scanning tools is to configure the web server to include an `X-Frame-Options` header. As we can see on the list available in the OWASP clickjacking defense cheat sheet [52], other methods are nevertheless possible, such as using the Content Security Policy (CSP) `frame-ancestors` directive header.

Missing Content-Type Header

This vulnerability has been detected by the ZAP security scanner and is because the `X-Content-Type-Options` response header was not set to "nosniff" to prevent the browser from doing MIME-type sniffing. Indeed, not using this header can potentially cause risks of MIME-sniffing attacks when a website allows users to upload content that can be downloaded and rendered as HTML to other users.

The risk of this vulnerability is very low since users of the Relog application cannot upload content such as images, stylesheets, or `JavaScript` files. It is, however, recommended to include this header and to assign it the value "nosniff" [53].

Absence of Anti-CSRF Tokens

This vulnerability is due to the fact that no anti-CSRF token has been found in the HTML submission forms of the Relog application. As explained previously, the

Relog application is not fully protected against CSRF attacks.

Password Field With Auto-Complete

This alert was issued by the Vega scanner since a form that included a password input field was detected without the `autocomplete` attribute set to off. If the latter is not disabled, some browsers may remember the user credentials entered in the form and store them locally, a potential attacker who can access the victim's computer could then steal this information [54].

To resolve this alert, we have to add the `autocomplete` attribute set to off for the input fields `passphrase` and `repeat_passphrase` of the `user.html` page.

5.5 Summary

In this chapter, we have carried out an analysis of the potential vulnerabilities of the Relog application in relation to three types of common attacks. Although we must remain cautious in the conclusions to be drawn as we have only covered a small part of the various security risks that exist, we have come to the following findings:

- The chances of success of SQL injection type attacks seem unlikely on the Relog application mainly due to the fact that the application only uses the API offered by `SQLAlchemy` to execute its SQL queries and that the known vulnerabilities of `SQLAlchemy` cannot be exploited.
- XSS type attacks do not seem to be possible due to the proper use of the `Knockout.js` library by the Relog application.
- The Relog application seems vulnerable to cross-site request forgery attacks. Checking the `Content-Type` header for POST requests received by the Relog application could easily limit the risk of this type of attack without fully preventing it.
- Vulnerability scanner tools did not find any serious flaws. The highest risk is clickjacking attacks. To prevent this, we recommend to include the `X-Frame-Options` header.

Chapter 6

Conclusion

During this thesis, we had the opportunity to collaborate with the CISM and the CÉCI to improve the application used to manage access to the different clusters among the French-speaking universities in Belgium. To do so, we worked on three different features that each required a good general understanding of the application, going from the front-end libraries like `Knockout.js` to the back-end framework used as `bottle`. Through these contributions, we have been able to improve the application with the following.

Firstly, by allowing the Relog application to manage the creation of new types of projects other than the projects intended for the Tier-1 level infrastructure 'Zenobe', and which we hope can be deployed across the different CÉCI clusters in the near future.

Secondly, by developing a tool to extract demographic data from the application in order to generate reports that allow a graphical and interactive visualization of this information. These reports will make it easier for CÉCI administrators to access and analyze this data.

And finally, by analyzing some of the different security risks that exist, we hope to have helped the CÉCI to detect some potential vulnerabilities in the Relog application.

However, having covered only some of the features to be implemented in the application's GitLab, and in a world where the need for high-performance computing facilities is constantly evolving, we are aware that further improvements can and should be made in the future. Nevertheless, we hope that this thesis has been as beneficial to CISM as it has been to us.

Appendix A

Contributions

These contributions consisted of bug fixes or feature requests. We resolved four of them, which are described in the following sections.

A.1 Default Username Size

The first fix was a small bug in **JavaScript** to get into the code. The challenge of this correction was not the line to correct itself but rather where to modify it. In the application, when a user enters their first and last name, a default suggested login is built from the first letter of his name and the last name. Sometimes it has been reported that the login could be greater than 8 characters when the user name was greater than or equal to 8 characters. While the rule for the login is from 4 to 8 characters, the login therefore appears in red so that the user has to modify it.

The simple fix to this issue was to truncate the automatically-built login to make it fit in the 8 characters long. We just use the **substring** function as you can see in Listing [A.1](#) below:

Listing A.1: Fixed default username size bug

```
1 | this.login((this.first_name()[0]  
2 |           + this.last_name()).substring(0,8))
```

We can notice two things here:

1. "[0]" is used to select the initial of the first name and;
2. "this.login()" uses the **Observable** feature in the **Knockout** framework which means that to write a new value in an **Observable**, we must call it

and pass the new value as a parameter.

A.2 Message in Web UI Misleading

This bug has been reported by more than one user and is the smallest, but impacts many users and also aims to clarify the understanding of the application when a user creates or updates a guest account. In fact, when creating a new guest account, an email is only sent to the guest when the supervisor has completed the form.

This bug consists of a green box which says: Emails have been sent to *"username of the new guest"* and *"username of the supervisor inviting the guest"*. While in fact, only the supervisor receives the email when this button is clicked. When the supervisor has completed the form, the guest will then receive an email.

In order to resolve this bug, the code was very simple and straightforward via Listing A.2:

Instead of printing:

```
1 | 'msg': 'Emails have been sent to %s and %s' % (  
2 |     email_guest, email)
```

We are now printing:

Listing A.2: Resolve Web UI misleading

```
1 | 'msg': 'An email has been sent to %s' % (email)
```

A.3 Create a New Email Template

In Chapter 2, we explained that the Relog application manages projects. This problem consists of an inconsistency when an email template sent to update a project is the same as the one sent when creating a project. It was reported by several Project Investigators who did not understand why, according to the e-mails they received, a previously created project was recreated instead of being updated.

With this issue, we had to make two commits. The first consisted of a new email template in the `emails.py` file which did not change during the last version A.3:

Listing A.3: Resolve New email template email part

```
1 'project_updated': '''
2 Dear {user.first_name} {user.last_name},
3
4     The {project_type.name} project "{project.acronym}" has
      been updated.
5
6 {signature}
7
8 '''
```

We made a modification in the `models.py` file which consists in a modification when sending the email (sending a `project_updated` model instead of a `project_ready` model). But we forgot to add a check, indicated in Listing A.3 in the first line, which checks if the project is not in the `old_project` list because otherwise it was reported on the GitLab that the same new email template would be send every time a project was created or updated.

Listing A.4: Resolve New email template model part

```
1 if not old_project:
2     emails.trigger_mail('project_ready', investigator,
      project=self)
3 else:
4     emails.trigger_mail('project_updated', investigator,
      project=self)
```

A.4 SSH2 Key Format Management

This issue was the most technical because before starting to code, it was mandatory to understand the difference between the `OpenSSH` [55] key format and the `SSH2` key format which are two major versions of the Secure Shell protocol (SSH [6]). `Relog` currently only supports the `OpenSSH` key format in the form for guest validation.

The `OpenSSH` key format is the one expected on Linux machines while some guests use an `SSH2` key format, the one that is generated by `Putty` [56] for instance. We show an example of each of these keys in Table A.1:

Type of key	Example
OpenSSH	<code>ssh-rsa AAAAB3NzaC1yc2EAAAABJQreg[...]iI02Jf409hn</code>
SSH2	<code>-- BEGIN SSH2 PUBLIC KEY -- Comment: "ceci" AAAAB3NzaC1yc2EAA[...]3fLSNBYw== -- END SSH2 PUBLIC KEY --</code>

Table A.1: Table illustrating the 2 different types of SSH keys

One simple way to convert a key from the SSH2 key format to the OpenSSH is by the `ssh-keygen` command:

```
ssh-keygen -i -f ssh2key.pub
```

Where the file `ssh2key.pub` is the file containing the SSH2 key and the resulting OpenSSH key is returned to the standard output.

But with Relog, the user passes this public key in the guest validation form, so we had to make changes on two sides, the Front-End (in JavaScript) and the Back-End (in Python). For the Front End, we just had to add the possibility to put an SSH2 key format in the form by modifying a `if-else` clause, among others, via the code below:

Listing A.5: Resolve SSH2 key format Front-End part

```
1 | if ((reader.result.slice(0, 3) == 'ssh')
2 | || (reader.result.slice(0, 15) == '---- BEGIN SSH2'))
```

On the Back-End side, we had to make these changes:

1. Check if the SSH key passed is of type SSH2.
2. Select the key itself between the markers (`-- BEGIN SSH2 [...] --END SSH2`).
3. Add the keyword "`ssh-rsa`" before the key.
4. Place this string in the `ssh_public_key` variable.
5. Then finalize by adding this new key to the user.

Listing A.6: Resolve SSH2 key format Back-End part

```
1 if self.ssh_public_key[0:15] == "---- BEGIN SSH2":
2     key_data = ''.join(self.ssh_public_key.splitlines()
3                           [2:-1])
4     key_data = "ssh-rsa " + key_data
5     self.ssh_public_key = key_data + "\n"
6 user['sshPublicKey'] = str(self.ssh_public_key)
```

While studying our request, we noticed that when the key was pushed from the SSH2 format, we forgot to add a "\n" to the line 4 at the end of it. Otherwise, the `ldapsearch` function will store it in plain format instead of `base64` encoding.

A.5 Summary

In conclusion, thanks to these bug fixes and feature requests, we were able to familiarize ourselves with the application and we now have a better understanding of the application code.

All these fixes have been reviewed by the developer in charge and are now merged into the production application and used by all Relog users.

Appendix B

Documentation: Adding new project types

B.1 User Documentation

Update to Handle Multiple Project Types: User Documentation

This documentation is related to issue 04 which was opened on the CÉCI Gitlab, namely *"Update to handle other types of projects than the Tier-1 projects"*. This section is intended for the users of the Relog application.

Changes Made

The changes that have been made to the Relog application concern the addition of new types of projects. The Relog application can now support several types of projects other than the already existing *tier1* type.

Project Creation Form

With regard to the form to be filled in when creating a project, we can now choose the type of project we want to create in the **Identify the project** section and the fields in the **Provide a description** remain the same regardless of the type of project. Only the fields of the **Setup details** change and the latter adapts dynamically to match the selected type.

CÉCI

New project "meac"

Help

1. Identify the project

Project Type

TIER1

Title

Lorem ipsum Duis

Acronym

meac

The acronym must be between 4 and 8 characters long and can only contain lowercase letters

Project Category

Catégorie 1

2. Provide a description

Short description of the project

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

Please provide as much information as you can such as the objective of the research, the funding source, partners, and a description of the resources you will need. (max 10000 chars)

3. Setup the details

Requested permanent disk space

100

Maximum shared disk space in GB the project will need

Requested temporary disk space

2000

Maximum temporary disk space in GB the project will need

Requested CPU hours

500000

Number of CPU hours the project will consume

Expiration date

2023-05-07

Example: 2018-12-31

Users

UserProject

aeda

Q

aeda

Lorem ipsum

Name

Lorem ipsum

Acronym

aeda

Investigator

sauvenee@uclouvain.be

Description

Praesent et pulvinar magna, in auctor nunc. Mauris dictum commodo arcu, nec dignissim nulla faucibus et. Nam venenatis iacus a mauris finibus efficitur id in magna. Maecenas non ligula vel ante eleifend feugiat. Donec leo tortor, mollis sit amet turpis ut, congue mollis mi.

Maecenas sit amet sapien neque. Vivamus mollis tristique orci in porta. Sed ante magna, faucibus id malesuada nec, dapibus sit amet velit. Integer

CPU hours

500000

Disk space

100

Scratch Space

100

Expiration Date

2023-05-02

Project Category

Catégorie 1

Users

nsauvene

Add user

Status

Draft

Waiting

Confirmed

Done

Actions

Save

Delete

Project secret url: /project/116?k=c3157d39-4fac-4532-8a60-ae06902d28e3

UserProject

abidev

Q

abidev

Name

Lorem Ipsum Dolor Sith Atmet

Acronym

abidev

Investigator

jyraty@uliege.be

Description

CPU hours

500000

Disk space

50

Scratch Space

100

Expiration Date

2020-08-13

Project Category

Catégorie 1

Users

jyraty

Add user

Status

Draft

Waiting

Confirmed

Done

Actions

Save

Delete

Mark the project as ready to push in LDAP

Project secret url: /project/83?k=060fb96a-9ce5-43f0-aa1a-e1011e2a0729

After

UserProject

aemo

Commodo elit at imperdiet

Name

Commodo elit at imperdiet

Acronym

aemo

Investigator

sauvenee@uclouvain.be

CPU hours

500000

Disk space

100

Scratch Space

200

Users

nsauvene

Add user

Description

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc lobortis mattis aliquam faucibus. Urna molestie at elementum eu facilisis. Sed id semper risus in hendrerit. Nulla malesuada pellentesque elit eget gravida cum sociis natoque penatibus.

At erat pellentesque adipiscing commodo elit at imperdiet dui accumsan. Libero volutpat sed cras

Expiration Date

2023-05-07

Project Type

TIER1

Project Category

Catégorie 1

Status

Draft

Waiting

Confirmed

Done

Actions

Save

Delete

Project secret url: /project/1217k=976e8f85-e91d-47c3-af44-e12f8e8f792e

Administrators now have the possibility to see the type of project selected.

UserProject

meac

meac

Lorem ipsum Duis

Name

Lorem ipsum Duis

Acronym

meac

Investigator

sauvenee@uclouvain.be

Scratch Space

2000

Users

nsauvene

Add user

Description

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

Expiration Date

2023-05-07

Project Type

scratch

Project Category

Catégorie 1

Status

Draft

Waiting

Confirmed

Done

Actions

Save

Delete

Project secret url: /project/1207k=61a69594-4362-4f0e-9672-72502919496c

Administrators can only modify attributes that relate to the selected project type.

Administrators cannot change the type of project selected.

UserProject

Name

Acronym

Investigator

Description

Expiration Date

Users

Status

Actions

Project secret url: /project/118?k=2b745004-738d-4773-81ff-fd8f087f9aa2

If a project does not yet have an associated type, administrators cannot mark the project as ready to push in ldap.

Board Policy

Concerning the new policies for updating the status of the projects in the *board* section, these have been established as follows:

- Board transitions remain the same.
- Administrators do not have the possibility to change the type of project regardless of its current status.
- If a project does not yet have an associated type (this one has just been created), administrators cannot mark the project as ready to push in ldap. The purpose of this is to prevent a project from being pushed into the ldap if its type field is *NULL*.
- When a user issues a project update request, the project type cannot be changed.
- The modification of a project type can only be done using the command-line interface.

Create new Types of Projects

It is possible to create new project types by executing the *create_project_type.py* script located at the root of the application.

```
create_project_type.py name eula start_uid [--cpu_time] [--disk_space] [--scratch_space]
```

Options	Description
name	Name of the new project type.
eula	Link to the eula of the new project type.
start_uid	start uid of the counter associated with the new project type.
--field	Set the field mandatory for the new project type.

note: Currently the creation of a new project type will copy and associate to the new type the different categories already defined for the first project type.

LDAP

The description of the POSIX group differs according to the different types of projects. When a project is pushed in the ldap, the application gets the record from the *project_type* table whose id corresponds to the attribute type of the current project. The keys *cpu_time*, *disk_space* and *scratch_space* are inserted only if the values of these attributes are set to true in the record that was just retrieved.

Below, we can see an example of an entry corresponding to a project for which only the *disk_space* field is required by the project type.

```
# 5000205, groups, ceci-hpc.be
dn: gidNumber=5000205,ou=groups,dc=ceci-hpc,dc=be
objectClass: posixGroup
gidNumber: 5000205
cn: test
description::  acronym: test
                category: Categorie 1
                confirmed_by: nicolas.sauvenee@student.uclouvain.be
                create_date: '2019-11-30'
                disk_space: 200
                expiration_date: '2022-11-29'
                investigator_email: testuser@uclouvain.be
                investigator_login: testuser
                name: test
                project_description: my test
                type: example2
memberUid: testuser
```

B.2 Developer Documentation

Update to Handle Multiple Project Types: Developer Documentation

This documentation is related to issue 04 which was opened on the CÉCI Gitlab, namely "*Update to handle other types of projects than the Tier-1 projects*".

The changes that have been introduced by this issue relate to the addition and the support of new types of projects. The main topics covered in this documentation concern the code changes and software testing made.

Code Change

This section will present the major changes that have been made to the codes of the Relog application. Some less significant code changes are deliberately not covered.

Back-End

The *ProjectType* class of the *models* file has been adapted to allow the definition of several types of projects with different mandatory fields. For this purpose, the attributes *disk_space*, *cpu_time* and *scratch_space* have been added to this class. The fields are of the *boolean* type and their default values are set to true. The *counter* attribute has been introduced to identify the *ProjectCounter* specific to the project type to be incremented. Finally, the default value of the *disk_space*, *scratch_space* and *cpu_time* attributes of the *Project* class has been removed.

```
class ProjectType(Base):
    __tablename__ = 'project_type'
    __table_args__ = (UniqueConstraint('name', 'eula', 'counter'))

    name = Column(String)
    eula = Column(String)
    counter = Column(Integer, nullable=False)
    disk_space = Column(Boolean, default=True, nullable=False)
    cpu_time = Column(Boolean, default=True, nullable=False)
    scratch_space = Column(Boolean, default=True, nullable=False)
```

The *incrementTypeSpecific* method has been added, when a project has to be pushed into LDAP, the *pid_number* is generated according to the counter specific to the current project type. As before, the latter will be used to define the attribute *gidNumber*.

```
class ProjectCounter(CounterBase):
    __tablename__ = 'projectid_counter'

    @classmethod
    def incrementTypeSpecific(cls, counter_type):
        session = CTX.get('session')
        row = session.query(cls)\
            .filter(cls.id == counter_type).first()
        row.last_uid += 1
        return row.last_uid
```

```

...

def push_ldif(self):
    logger.info('Push ldif for project %s' % self.acronym)
    session = CTX.get('session')
    project_type = session.query(clsProjectType)\
        .filter_by(id=self.type).first()
    if self.pid_number is None:
        self.pid_number =
ProjectCounter.incrementTypeSpecific(project_type.counter)
    ...

```

When the *description* of the POSIX group is defined using a dictionary, the record of the table *project_type* whose id corresponds to the type attribute of the project that is pushed into the LDAP is obtained. The keys *cpu_time*, *disk_space* and *scratch_space* are inserted only if the values of these attributes are set to true in the record just retrieved. If the value of one of the attributes of the *project_type* record is set to true but the corresponding project field is not defined, a *MissingAttributeError* is raised.

```

description_dict = {
    'name': str(self.name.encode("UTF-8")),
    'acronym': str(self.acronym.lower()),
    'create_date': self.create_date.strftime(DATE_FORMAT),
    'confirmed_by': str(self.confirmed_by),
    'project_description': str(self.description.encode("UTF-8")),
    'investigator_email': str(investigator.email),
    'investigator_login': str(investigator.login),
    'category': str(ascii(category.name)),
    'type': str(type.name),
}

optional_fields = ['cpu_time', 'disk_space', 'scratch_space']
for field in optional_fields:
    if getattr(project_type, field):
        attr_value = getattr(self, field)
        if attr_value is None:
            raise MissingAttributeError('Mandatory attribut %s is set to None' %
(field))
        description_dict[field] = attr_value

```

Front-End

relog.js

The observables *available_types_list*, *_specific_type_fields* and the *specific_fields* computed observable have been added to the *Project* model. The observable *available_types_list* contains all project types stored in an array and is used in the drop-down list type of the *project.html* file. The observable *specific_type_fields* is used to store the type of the selected project.

```
// Fields added to the class Project of the relog.js file
this.available_types_list = ko.observable();
this.specific_type_fields = ko.observable({});

ko.computed(this.specific_fields, this);
```

The attributes *disk_space*, *cpu_time* and *scratch_space* have been added to the fields to be retrieved in the *load* function of the Project class. Therefore, the project types contained in the observable *available_types* will include these new fields.

```
// get available types
var prm = load_prm.then(function() {
    return rpc('project_type', 'read', [], {
        'fields': ['id', 'name', 'eula', 'disk_space', 'cpu_time',
        'scratch_space'],
        'order_by': ['id'],
    });
}).bind(this));
prm = prm.then(function(res) {
    var avt = {};
    for (var pos in res) {
        var id = res[pos][0];
        avt[id] = {
            'id': id,
            'name': res[pos][1],
            'eula': res[pos][2],
            'disk_space': res[pos][3],
            'cpu_time': res[pos][4],
            'scratch_space': res[pos][5]
        };
    }
    this.available_types(avt);
}).bind(this));
```

The *specific_fields* computed observable depends on the observable *available_types* and *type* and consequently, the *specific_type_fields* will automatically be updated whenever any of these dependencies change.

```
Project.prototype.specific_fields = function() {
    if (!this.id() || !this.available_types() || !this.type()) {
        return;
    }

    var active_fields = this.available_types();
    this.specific_type_fields(active_fields[this.type()]);
};
```

Now, the different fields of a project that will be written will depend on whether the fields *cpu_time*, *scratch_space* and *disk_space* are required by its project type defined in the *specific_type_fields*.

```
Project.prototype.write_fields = function() {
    var fields = [
```

```

        'name', 'acronym', 'timestamp', 'expiration_date', 'description',
        'type', 'category'
    ];

    var specific_fields = this.specific_type_fields();
    if (specific_fields) {
        var type_specific_fields = Object.keys(specific_fields)
            .filter( key => specific_fields[key] === true);
        fields = fields.concat(type_specific_fields);
    }
    if (this.role() == 'admin') {
        var extra = ['investigator'];
        fields = fields.concat(extra);
    }
    return fields;
};

```

To prevent a project that does not yet have an associated type to be marked by an administrator as ready to be pushed in the LDAP. The computed observable *enable_save* had to be modified.

```

this.enable_save = ko.computed(function() {
    if (this.errors().acronym || this.errors().expiration_date) {
        return false;
    }
    if (this.role() == 'user' && this.status() != 'draft'){
        return false;
    }
    /* if no associated type and the state of new status is not draft,
       saving option is disabled */
    if (!this.type() && this.new_status() && this.new_status() != 'draft'){
        return false;
    }
    return true;
}, this);

```

HTML files

Regarding the *project.html* and the *board.html* files. The visible binding system provided by knockout.js is used to make the associated DOM element for each of these *div* tags hidden or visible according to the type of the project. Therefore, the DOM element for each of these *div* tags becomes visible (invisible) if the value of the key corresponding to the field in the observable *specific_type_fields* is true (false).

```

<!-- board.html -->
<div class="col-md-6 col-sm-6">
    <div data-bind="visible: specific_type_fields().cpu_time">
        <div class="form-group" data-bind="css: 'has-' + errors().cpu_time">
            <label class="control-label" for="cpu_time">
                CPU hours
            </label>
            <input type="number" min="0" class="form-control focus_popover"
                id="cpu_time" placeholder="500000"
                data-bind="value: cpu_time, valueUpdate:'keyup'"/>
        </div>
    </div>

```

```

        </div>
    </div>
    <div data-bind="visible: specific_type_fields().disk_space">
        <div class="form-group" data-bind="css: 'has-' + errors().disk_space">
            <label class="control-label" for="disk_space">
                Disk space
            </label>
            <input type="number" min="0" class="form-control focus_popover"
                id="disk_space" placeholder="100"
                data-bind="value: disk_space, valueUpdate:'keyup'"/>
        </div>
    </div>
    <div data-bind="visible: specific_type_fields().scratch_space">
        <div class="form-group" data-bind="css: 'has-' +
errors().scratch_space">
            <label class="control-label" for="scratch_space">
                Scratch Space
            </label>
            <input type="number" min="0" class="form-control focus_popover"
                id="scratch_space" placeholder="100"
                data-bind="value: scratch_space, valueUpdate:'keyup'"/>
        </div>
    </div>
</div>

```

Unit Testing

This section will cover the unit tests that have been implemented in regards to issue 04.

Tech Stack

- Pytest (4.6.9)
- Factory Boy (2.8.1)

Installation

- Install pytest:

```
pip install pytest
```

Using pip and Python 2.7, the version that will be installed will automatically be the last pytest 4.6.X and not the 5.0 or later versions.

- Install Factory Boy:

```
pip install factory_boy==2.8.1
```

- To launch the tests, run the command below at the root of the *Relog* application:

```
py.test tests/unit
```

Test Scope

The parts of code that are covered by the unit tests concern the *Project* class of the *models.py* file, and more particularly, the methods *set_action* and *set_status*. These are tested with multiple data, actions, and roles. The tests are intended to run on the *Relog* application once the latter handles multiple project types.

Testing Approach

Concerning the approach that is used for unit testing, this one is composed of different steps.

Action	Characteristic
pre-populate in memory db	For all tests
Begin transaction	For each test
Generate data with factories	For each test
Execute logic of the test	For each test
Rollback transaction	For each test

Before executing all the unit tests, an in-memory database is first generated. This database will be fed only with basic data that are necessary and common to all the tests. For example, we include some users and admins as well as different institutions. The python module that performs this function can be found in the *prepopulate_db.py* file.

For each unit test, we will run them in isolated transactions and apply a rollback once they are completed. The *pytest* fixture responsible for this can be found below and is in the *confest.py* file.

```
@pytest.yield_fixture(scope='function')
def test_session(ctx):
    conn = ctx.engine.connect()
    transaction = conn.begin()
    with ctx.session(acl_key=None, admin_email=None, conn=conn) as session:
        setup_factories(session)
        yield session
    transaction.rollback()
```

Note: to allow this, the signature of the *_session* function of the class context had to be modified. An argument has been added to the latter, it is a connection with a default value of *None* with which the new session will be associated. This option is only used for unit tests and is the only modification made outside the *unit* test file.

```
def _session(self, acl_key=None, admin_email=None, cli=None, conn=None):
    if conn is None:
        session = self.sessionmaker()
    else:
        session = self.sessionmaker(bind=conn)
```

The factory-boy framework is used for the creation of the test data. The different factories of our models that have been defined for this can be found in the *models_factory.py* file.

Finally, we can find below an example of unit test present in the file *test_project_set_status_done.py*

```
def test_set_status_done_random_projects(test_session):
    ctx = models.CTX
    user = test_session.query(User).first()
    admin = test_session.query(Admin).first()

    nbr_projects = randint(START_RANDOM, END_RANDOM)
    projects, counter, project_type = init_confirmed_projects(
        nbr_projects,
        user,
        admin,
        START_UID)

    ctx.set('role', 'admin')
    ctx.set('admin', admin)
    for project in projects:
        if not is_project_valid_according_type(project, project_type):
            with pytest.raises(Exception):
                project.set_status("done")
        else:
            project.set_status("done")
            assert project.confirmed_by == admin.email
            assert project.pid_number == counter.last_uid
    projects_status_done = test_session.query(Project)\
        .filter_by(status="done")\
        .all()
    assert START_UID + len(projects_status_done) == counter.last_uid
```

Unit Tests Structure

```
tests
├── config.json
├── happy.py
└── unit
    ├── conftest.py
    ├── prepoluate_db.py
    ├── models_factory.py
    ├── test_project_set_action_create.py
    ├── test_project_set_action_update.py
    ├── test_project_set_status_done.py
    └── test_project_set_status.py
```

All unit tests are located in the *unit* folder of the *tests* folder. In the latter, we find the 4 modules containing our tests:

- **test_project_status_done.py**: in this one, we can find the specific tests for issue 04, we check for example by creating multiple projects with random values and multiple project types with different mandatory fields, that a push in the ldap can only be done for projects

respecting the conditions imposed by their project types and that for the others, an error is returned.

- **test_project_status.py**: contains various tests concerning the set status other than *done*, for instance, that a project draft cannot be validated if a user with a public role has an invalid *acl_key*, that a confirmed project can only be done for the roles admin and cli,...
- **test_project_set_action_create.py**: file in which we can find the tests related to the action *create_project*, we find for example the tests verifying that initializing a project with an acronym already taken by a team prevents the creation of the project, that initializing a project whose acronym is already existing also avoids the creation of the latter,...
- **test_project_set_action_update.py**: finally, we test various scenarios when an update action is issued, in particular, that a project not yet pushed in the ldap cannot be updated, that only the project investigator has the right to issue an update for his project.

Output

When the tests are run, tests that successfully passed will be indicated using the dot notation. If all our tests are passed, the following result should appear.

```
[relog@myserv335 relog-fork]$ py.test tests/unit/
===== test session starts =====
platform linux2 -- Python 2.7.5, pytest-4.6.9, py-1.8.1, pluggy-0.13.1
rootdir: /home/relog/relog-fork
plugins: cov-2.8.1
collected 21 items

tests/unit/test_project_set_action_create.py ..... [ 23%]
tests/unit/test_project_set_action_update.py .... [ 42%]
tests/unit/test_project_set_status.py ..... [ 80%]
tests/unit/test_project_set_status_done.py .... [100%]

===== 21 passed in 5.42 seconds =====
```

On the other hand, failed tests will be indicated using the 'F' notation. In the example below, the test *test_set_status_confirmed_role_admin* failed and we can easily see the intermediate values in the assertion to help to understand the reason for the failure.

```
[relog@myserv335 relog-fork]$ py.test tests/unit/
===== test session starts =====
platform linux2 -- Python 2.7.5, pytest-4.6.9, py-1.8.1, pluggy-0.13.1
rootdir: /home/relog/relog-fork
plugins: cov-2.8.1
collected 21 items

tests/unit/test_project_set_action_create.py ..... [ 23%]
tests/unit/test_project_set_action_update.py .... [ 42%]
tests/unit/test_project_set_status.py ...F.... [ 80%]
tests/unit/test_project_set_status_done.py .... [100%]

===== FAILURES =====
_____ test_set_status_confirmed_role_admin _____

init_draft_project = (<project(1)>, <user(1)>, <sqlalchemy.orm.session.Session
object at 0x7f88ff215450>)
```

```
def test_set_status_confirmed_role_admin(init_draft_project):
    project, active_user, session = init_draft_project
    admin = session.query(Admin).first()
    ctx = models.CTX
    ctx.set('role', 'admin')
    ctx.set('admin', admin)
    result = project.set_status("confirmed")
>    assert result is False
E    assert True is False

tests/unit/test_project_set_status.py:63: AssertionError
===== 1 failed, 20 passed in 4.03 seconds =====
```

B.3 Administrator Documentation

Update to Handle Multiple Project Types: Administrator Documentation

This documentation is related to issue 04 which was opened on the CÉCI Gitlab, namely "*Update to handle other types of projects than the Tier-1 projects*".

Installation

Pull Modifications

After the changes have been integrated on the master branch of the *Relog* repository, login to `serv335` and pull the modifications:

```
ssh serv335
sudo su - relog
cd relog
git pull
```

Database Update

Before restarting the service, it is necessary to update the database and more particularly the schema of the table *project_type*:

- Create a new table in which we add the columns *disk_space*, *cpu_time*, and *scratch_space*. These attributes indicate whether a field is required for a specific project type. The new table also has the column *counter*. The latter indicates the project_counter that corresponds to the project type.
- Insert the already existing project type *tier1* in the new table. The fields *disk_space*, *cpu_time*, *scratch_space* will be set to true and the specific counter has to be the counter with id 1.

```
CREATE TABLE project_type_tmp (
  `id`      INTEGER NOT NULL,
  `timestamp` TIMESTAMP,
  `name`    VARCHAR UNIQUE,
  `eula`    VARCHAR,
  `disk_space` INTEGER DEFAULT 1
    CHECK (disk_space IN (0,1)),
  `cpu_time` INTEGER DEFAULT 1
    CHECK (cpu_time IN (0,1)),
  `scratch_space` INTEGER DEFAULT 1
    CHECK (scratch_space IN (0,1)),
  `counter` INTEGER NOT NULL UNIQUE,
  PRIMARY KEY(id)
);
INSERT INTO project_type_tmp (id, timestamp, name, eula, counter) SELECT *, 1
FROM project_type;
DROP TABLE project_type;
ALTER TABLE project_type_tmp RENAME TO project_type;
```

note: It is preferable to make a backup of the database before updating the table *project_type*.

Restart Service

Once the above operations have been done, restart the *relog* service.

```
service relog restart
```

Final Remark

It is possible that projects not corresponding to the Tier-1 type have been manually inserted in the LDAP, it is important to check when a new project type is created that the value of the *last_uid* attribute of the associated counter does not overlap with the *gidNumber* values of existing projects in the LDAP.

B.4 Integration Testing

Sheet1

Test Name	Type	Steps	Objective	Expected Result	Result	Prepared Date
T_001	Manual	-Open CÉCI Project management -Request the creation of a 'project'	Verify customer is able to initiate the creation of a new project	The project details must be inserted in the database	Passed	09/05/20
T_002	Manual	-Open CÉCI Project management -Request the creation of a 'project' with the same acronym as the project just created	Check that a project with an existing acronym cannot be created	Project details should not be inserted in the database	Passed	09/05/20
T_003	Manual	-Connect to the vm -Run the script create_project_type.py	Check if a new project type can be created in command line interface	The new project type and its associated counter must be inserted in the database with the correct information	Passed	09/05/20
T_004	Manual	-Click on the link of the new project request -Select the new type of project that was created previously -Complete some fields and save the draft	Check that a project with a type different than tier1 can be registered correctly as a draft	The information that has been filled in should be included in the database and fields that are not required by the new project type must not be included	Passed	09/05/20
T_005	Manual	-Click on the link of the previously created project and which was saved as a draft	Check that the information that has been saved are correctly fetched	Information that has been previously saved as must be displayed	Passed	09/05/20
T_006	Manual	-Click on the link of the previously created project and which was saved as a draft -Change the type of the project to tier1	Check that the user can change the type of a project if the latter has the status of 'Draft'	The new type of project should be inserted into the database and its project category must be adapted	Passed	09/05/20
T_007	Manual	-Open the CÉCI administrator Board -Find the new project created -Change its description -Save as 'Draft'	Check that the CÉCI administrators can change the information of a project with a new type without modifying the different information related to its type	The information that has been saved by the administration must be recorded in the database and the data relating to the type of project must remain unchanged.	Passed	09/05/20
T_008	Manual	-Click on the link of the previously created project and which was saved as 'Draft' -Add a new user to the project -Save the draft	Check that the user can add a new project member for a type other than tier1	The new member should be added to the project	Passed	09/05/20

Sheet1

T_009	Manual	-Open the CÉCI administrator Board -Find the new project created -Delete the new registered user -Save	Check that the admin can delete a project member for a project of a type other than tier1	The new member should be deleted of the project	Passed	09/05/20
T_010	Manual	-Open the CÉCI administrator Board -Find the new project created -Change its category -Save	Check that the admin can change the category of a project of the type other than tier1	The category of the project should be saved and the new category id should reference the type of the project	Passed	09/05/20
T_011	Manual	-Click on the link of the previously created project and which was saved as a draft -Submit the form	Check that the user can submit a project of the type other than tier1	The administrator should receive a confirmation link and the status of the project should be marked as 'Waiting'.	Passed	09/05/20
T_012	Manual	-Open the CÉCI administrator Board -Find the submitted project -Try to change its type	Check that CÉCI administrators cannot change the type of a project	The administrator should not be able to change the project type	Passed	09/05/20
T_013	Manual	-Click on the validation link of the previously submitted project -Change the description of the project -Save the form	Check that CÉCI administrators can change the information of a project of the type other than tier1 in the validation form	The information that has been modified should be saved in the database	Passed	09/05/20
T_014	Manual	-Open the CÉCI administrator Board -Find the submitted project -Confirm it	Check that CÉCI administrators can confirm a project of the type other than tier1	The status of the project should be marked as 'Confirmed' and its attribute 'confirmed_by' should be the administrator who performed the action	Passed	09/05/20
T_015	Manual	-Wait for the batch job to trigger the confirm->done transition on projects	Check that confirmed project will be marked as 'Done' by the batch jobs and pushed into the LDAP	The status of the project should be mark as 'Done', its 'in_ldap' attribute to true and the value of its 'pid_number' should correspond to the counter of its associated type. The LDAP should be correctly updated	Passed	09/05/20
T_016	Manual	-Open CÉCI project management -Request a project update	Check that a project can be updated	The status of the project should be mark as 'Draft' and its action as 'update_project'	Passed	09/05/20

Sheet1

T_017	Manual	-Open the link to update the project -Change the category of the project -Save	Check that the changes made on the 'update project' form can be saved	The new category of the project should be saved	Passed	09/05/20
T_018	Manual	-Open the CÉCI administrator Board -Search for the project that has just been modified	Check that CÉCI administrators can see the changes when a project is updated	The new category of the project should be displayed and the project status should be 'Draft'	Passed	09/05/20
T_019	Manual	-Open the link to update the project -Send the form	Check that the modification of a project can be sent	The status of the project should be mark as 'Waiting'	Passed	09/05/20
T_020	Manual	-Open the admin validation form by clicking on the link received. -Confirm the update	Check that the modification of a project can be confirmed	The status of the project should be marked as 'Confirmed' or "Done", its 'pid_number' should be the same and the LDAP should be successfully updated with its new category if the status of the latter is marked as 'Done'	Passed	09/05/20
T_021	Manual	-Open CÉCI Project management -Perform new project registration	Verify customer is able to initiate the creation of a new project	Project details should be insert into the database	Passed	09/05/20
T_022	Manual	-Open the CÉCI administrator Board -Search for the new project created -Try to confirm it	Verify that CÉCI administrators cannot confirm a project with no type	The action button must be disabled when the new status of the project is not 'Draft'	Passed	09/05/20
T_023	Manual	-Click on the link of the new project request -Select the a type of project -Complete some fields and add a project member -Send the form	Verify that a user can confirm a project and add project members	The information that has been filled in should be included in the database , its status set to 'Waiting' and the registrations of the project members should be saved in the database	Passed	09/05/20
T_024	Manual	-Click on the link of the admin validation -Confirm the project	Verify that a project can be pushed into the LDAP with multiple project members	The status of the project should be marked as 'Confirmed' or 'Done',if the status is 'Done', the fields 'in_ldap', 'pid_number' and the LDAP must be updated	Passed	09/05/20
T_025	Manual	-Create a new account and join the project pushed previously in the LDAP -Push the user into the LDAP	Verify that a user can join a existing project that have been pushed into the LDAP	The LDAP should be updated with the new account	Passed	09/05/20

Sheet1

T_026	Manual	-Join an existing project with a CÉCI account -Add the project that has been pushed into the LDAP to the account projects -Confirm the request with an CÉCI administrator	Verify that a user can join a existing project that have been pushed into the LDAP by sending a request 'to join project'	The LDAP should be updated after some time	Passed	09/05/20
-------	--------	---	---	--	--------	----------

Appendix C

Documentation: Extract Demographic Information

C.1 Documentation

Extract_Demo

Description

Extract_Demo is a tool that has been developed as part of the **Relog** application. The primary purpose of this module is to extract demographic data from the database and subsequently generate reports with the collected data.

The demographic information that is collected is mainly related to the CÉCI institutions and more particularly the data on the number of registered and expired users as well as information on projects.

Extract Demo creates HTML reports using the web template engine *Jinja* and used the *Chart.js* open-source *JavaScript* for data visualization.

Tech Stack

- Jinja2 for Templating
- SQLAlchemy (1.1.14)
- Python (2.7)
- Reports
 - Chart.js (2.7.3)
 - Bootstrap (3.3.7)
 - jquery (2.2.4)

Getting Started

Installation

- Download the latest version of *Extract_Demo* on [GitLab](#).
 - Extract it to the *~/relog* folder.
- Install *Jinja2*.

```
pip install Jinja2==2.11.1
```

- Install *MarkupSafe* (optional if *Jinja2* was installed with *pip*).
- Extract the database from which a report is to be generated in the folder *~/relog/extract_demo/var*.
- Add Relog root directory to the *PYTHONPATH*

```
export PYTHONPATH="${PYTHONPATH}:/home/relog/relog"
```

Basic Example

In this section, we will provide a basic example of a report that can be generated from the *Extract_demo* tool.

The first thing to do is to configure the report to be generated using the file *extract_demo_config.json* (for more details see the *configuration* section). In the below example, the configuration file is as follows:

```
{
  "json_institutions_name": "data.json",
  "json_projects_name": "data_projects.json",
  "starting_year": 2011,
  "end_year": 2019,
  "university": {
    "document_title": "Universities - Registrations",
    "title": "Universities",
    "secondary_title": "Demographic Information",
    "display_introduction_section": true,
    "introduction_title": "About This Report",
    "introduction_text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque porta nisl dui, et vulputate sem aliquam gravida. Maecenas sollicitudin enim nulla, vitae consequat velit facilisis ac.",
    "additional_chart_type": {
      "bar": true,
      "doughnut": true,
      "radar": true
    },
    "color": {
      "default": "tableau.Blue20",
      "uc1": "tableau.Blue20",
      "ulg": "office.Blue6",
      "umons": "tableau.OrangeGold20"
    },
    "active_users_color": {
      "uc1": "#997119"
    }
  },
  ...
}
```

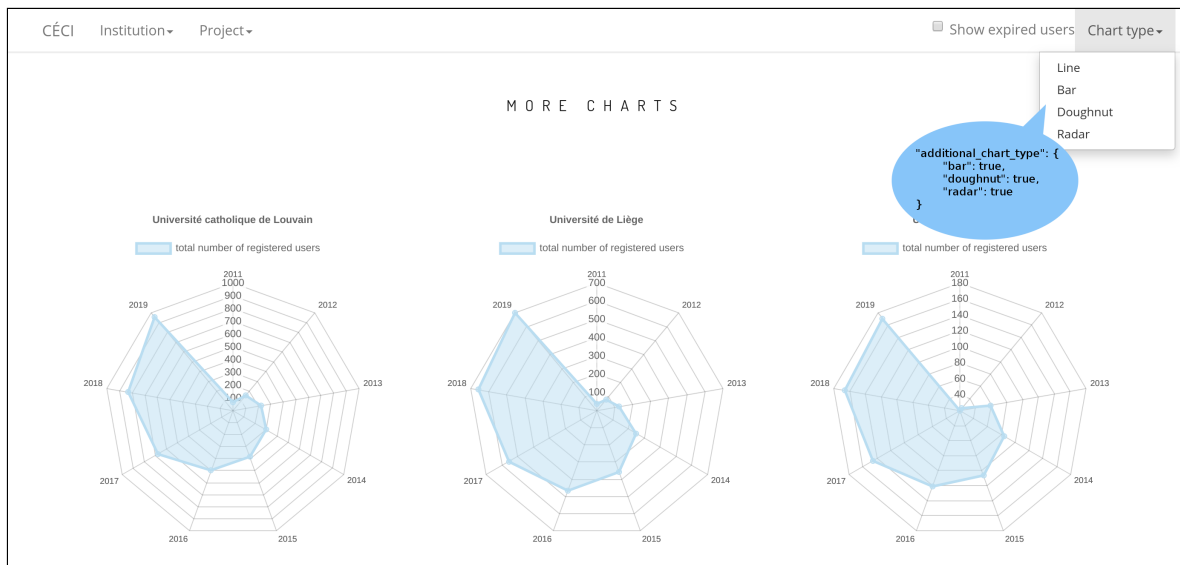
Once we have configured the report to be created, we can run the bash file *extract_demographic_info.sh*. After the program has finished running, the *report_demo_info.tar.gz* archive will be created. In this one, we will be able to find our report with the following structure.

```
.
├── assets
│   ├── bootstrap
│   ├── css
│   ├── fonts
│   └── js
├── universities.html
├── institutes.html
├── poles.html
└── projects.html
```

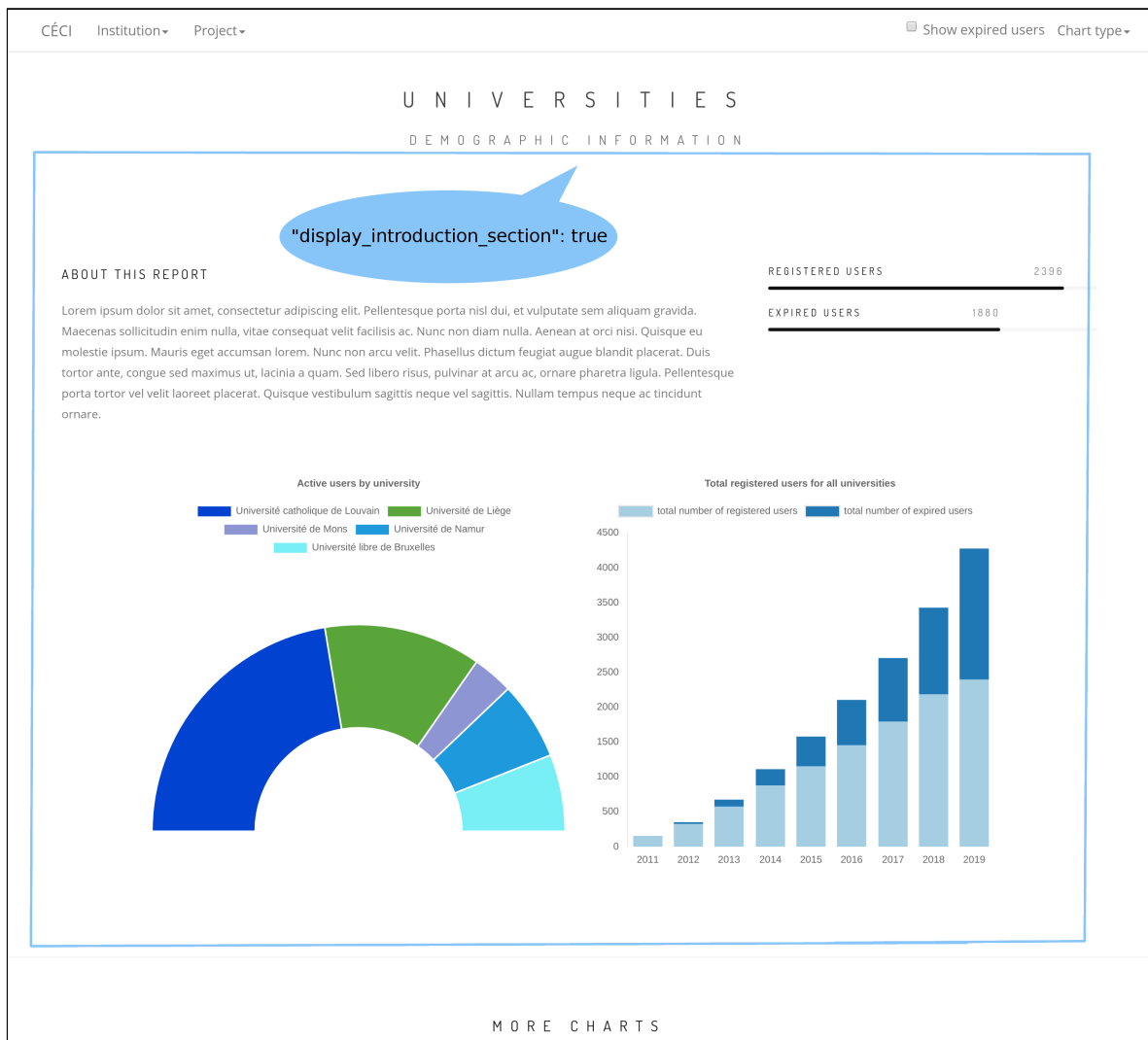
Configuration

In this section, we will detail the different possible options we can use to configure our report.

- **starting_year/end_year:** Start and end date of the period over which the data will be collected and covered by the report.
- **document_title:** Allows to define a title for the HTML document.
- **title/secondary_title:** Allows to define the main and secondary title of the introduction section, if the *display_introduction_section* option is not enabled, these will not be visible.
- **additional_chart_type:** The different properties defined inside this object allow to activate different types of graphs that the user will be able to choose to visualize the different information. When a specific type is activated, the user will be able to dynamically change the types of the charts using a *dropdown* defined in the *navbar* of the institutions' HTML pages. The line chart is the default type and this one cannot be deactivated.



- **display_introduction_section:** Allows to display for the universities and projects sections the additional charts, i.e. charts of active users for all universities, progress bars, projects by categories,...



- **color:** The different properties defined inside this object allows to define colour themes for the different charts. The effect of this color option may vary depending on whether it is used within the university section or within the institute/pole section.
 - When it is used to define a colour scheme for poles or institutes, the user has the choice to define a colour theme for all poles/institutes belonging to a certain university or for a specific pole/institute. The example below allows to define the *RdYlBu4* theme to any institute belonging to *UCLouvain* and to define the *Blue20* theme to the *Centre Spatial de Liege* institute.

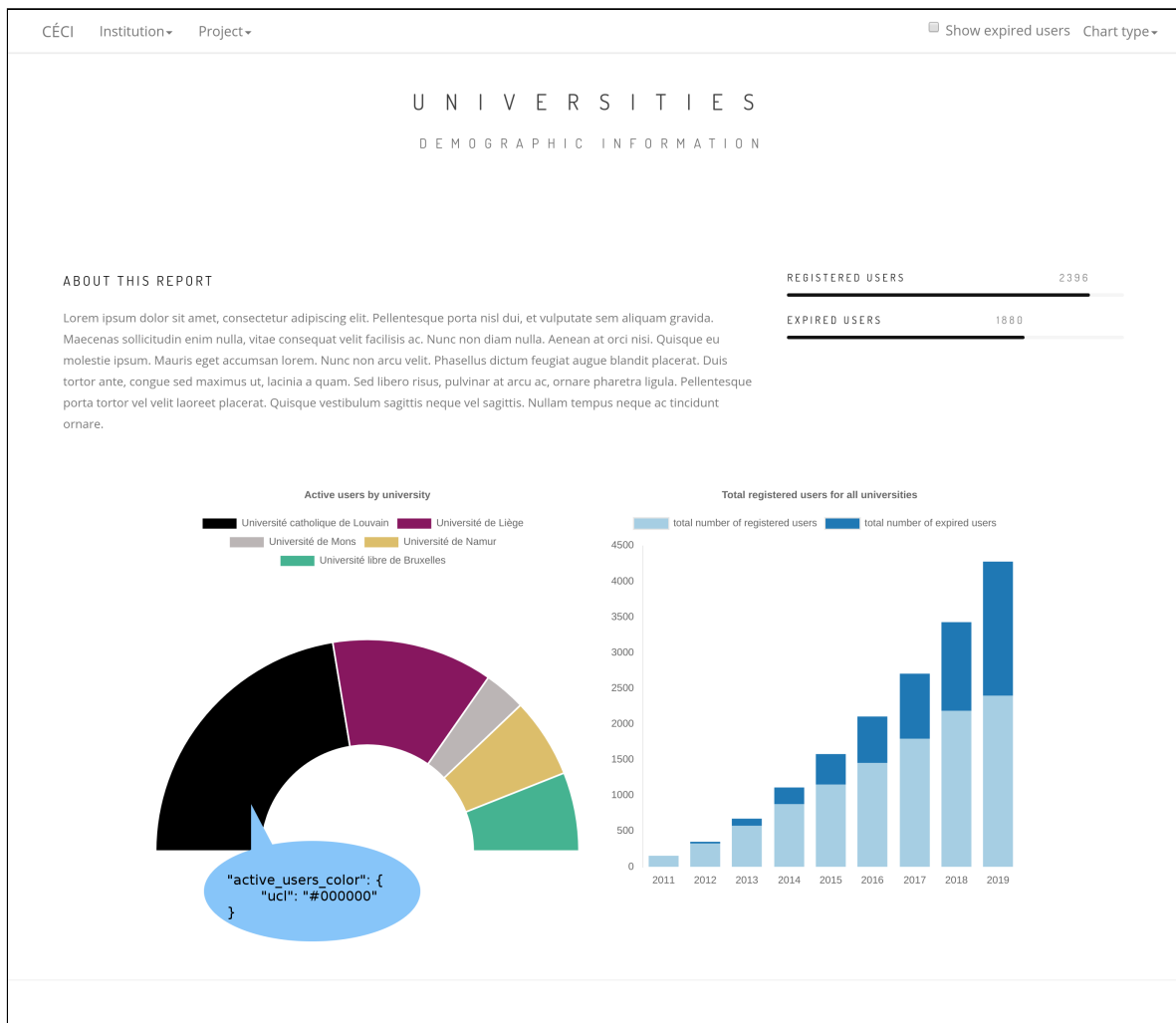
```
"color": {
  "default": "tableau.OrangeGold20",
  "uc1": "brewer.RdYlBu4",
  "cslulg": "tableau.Blue20"
}
```

- When it is used to define a colour theme for universities, the user must specify each university for which the colour theme is to be applied.



The complete list of color schemes that can be applied can be found on this [link](#).

- **active_users_color:** This option allows to set a color for a specific university concerning the chart of active users. Unlike the color option above, this one must be specified using an *RGB* value and is only available for the university section.



```
|   └─ projects
|       └─ projects_template.html
└─ var
    └─ db.sqlite
```

1. **extract_demo_config.json**: JSON configuration file, it allows to define the options and parameters desired before generating the report. It is also used to set the names for the different JSON files that will be extracted.
2. **loading_scripts**: Folder in which we can find the different scripts to extract the desired data from the database in JSON format. This is the data that will be used by the *main.py* script to generate the report.
3. **templates**: Folder in which we can find our different *Jinja* templates. These are defined using template inheritance with a skeleton that can be found in the *base_template.html* file and several macros defined in the *macros.html* file.
 - Concerning the *institutions* sub-folder, this one contains the templates *universities_template.html* and *low_level_institutions_template.html* which will be used to generate the files *universities.html* and *institutes.html/poles.html* respectively. In parallel to the base template file, the file *institutions_template.html* is used as the common template for the *institutions'* folder.
 - Regarding the *projects* folder, the latter contains the file *projects_template.html* which is the template that will be used to create the *projects.html* file.
4. **main.py**: This python script takes as input the different JSON files that have been generated in the folder *loading_scripts* and will use the different templates to produce the final report. This will be done by taking into account the different parameters set in our configuration file.
5. **extract_demographic_info.sh**: Shell executable file that serves as an entry point to the program and automates all of the above operations. Finally, this one will create an archive containing the different HTML files as well as the *assets* folder of our report.

JSON Structure

This section will detail the JSON schema that has been chosen to represent and extract the data related to the different institutions and projects of the CÉCI.

Institutions Schema

```
{
  "years": [2011, 2012, 2013],
  "all_universities": {
    "registered_users": [155, 325, 573],
    "expired_users": [0, 25, 102]
  },
  "universities": [
    {
      "abbreviation": "test",
      "demographic_info": {
        "registered_users": [64, 155, 224],
        "expired_users": [0, 14, 57],
      },
      "name": "Université catholique de Louvain"
    }
  ]
}
```

```

],
"institutes": [
  {
    "parent_abbreviation": "umons",
    "name": "Departement Mines-Geologie",
    "demographic_info": {
      "registered_users": [10, 21, 29],
      "expired_users": [0, 1, 3],
    },
    "abbreviation": "mingeo"
  }
],
"poles": [
  {
    "parent_abbreviation": "unamur",
    "name": "Unite de Geologie",
    "demographic_info": {
      "registered_users": [3, 3, 3],
      "expired_users": [0, 0, 0],
    },
    "abbreviation": "ugeologie"
  }
]
}

```

1. **root**: The root schema is used to represent data relating to CÉCI institutions. It required the arrays properties *years*, *universities*, *institutes*, and *poles*. The root also needs the object property *all_universities*.
2. **years**: This array represents the range of years for which data were collected. All values of the *registered_users* and *expired_users* keys in the entire JSON will refer to the years in this table.
3. **all_universities**: This object, which requires two arrays, is used to represent registered and expired users from all of CÉCI partner universities.
4. **universities/institutes/poles**: These arrays represent all of CÉCI's universities, institutes, and poles respectively. Each object included in these arrays has the property *demographic_info*. The latter has two properties:
 - **registered_users**: Cumulative sum of the number of registered users in the institution based on the years defined in the root's *years* property.
 - **expired_users**: Cumulative sum of the number of expired users in the institution based on the years defined in the root's *years* property.

Projects Schema

```

{
  "total_projects": 102,
  "total_expired_projects": 66,
  "total_projects_by_category": [
    {
      "category": "Catégorie 1",
      "total_projects": 84
    },
  ],
}

```

```

{
  "category": "Catégorie 2",
  "total_projects": 4
},
{
  "category": "Catégorie 3",
  "total_projects": 1
}
],
"projects_by_status": [
  {
    "status": "done",
    "total": 26
  },
  {
    "status": "draft",
    "total": 73
  },
  {
    "status": "waiting",
    "total": 3
  }
],
"project_details": [
  {
    "cpu_time": 2500000,
    "status": "draft",
    "name": "Domain decomposition methods for time-harmonic wave problems",
    "acronym": "wavesddm",
    "scratch_space": 500,
    "number_registrations": 5,
    "disk_space": 50
  }
]
}

```

1. **root**: The root schema is used to represent data relating to CÉCI projects. It required the properties *project_details*, *projects_by_status* and *total_projects_by_category* which are of type array. The root also needs the integer properties *total_projects* and *total_expired_projects*.
2. **projects_by_status**: Total number of registered projects (expired and unexpired) according to their status.
3. **total_projects_by_category**: Total number of registered projects (expired and unexpired) by category.
4. **project_details**: This property is used to represent all the projects and their information that have been registered by CÉCI.

Appendix D

Security Audit

D.1 Sqlmap

D.1.1 Example of HTTP Request Tested

```
1 POST /_rpc/institution/read HTTP/1.1
2 Referer: https://login.myceci-hpc.be:443/_rpc/institution/read
3 Host: login.myceci-hpc.be
4 Content-type: application/json
5 Content-length: 144
6
7 {"args": [], "kwargs": {"fields": ["id", "name", "parent", "levels"], "filter_by": {}, "limit": null, "
8   order_by": []}}
```

D.1.2 Example of Request Generated by Sqlmap

```
1 HTTP request [#70]:
2 POST /_rpc/institution/read HTTP/1.1
3 Referer: https://login.myceci-hpc.be:443/_rpc/institution/read
4 Host: login.myceci-hpc.be
5 Content-type: application/json
6 Content-length: 142
7 Connection: close
8
9 {"args": [], "kwargs": {"fields": ["id WHERE 3966=3966 AND 4053=3286-- mkzP", "name", "parent", "
   levels"], "filter_by": {}, "limit": null, "order_by": []}}
```

D.1.3 Obtained Results

```

[19:04:46] [INFO] testing 'PostgreSQL time-based blind - Parameter replace (heavy query)'
[19:04:46] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind - Parameter replace (heavy queries)'
[19:04:46] [INFO] testing 'Oracle time-based blind - Parameter replace (DBMS_LOCK.SLEEP)'
[19:04:46] [INFO] testing 'Oracle time-based blind - Parameter replace (DBMS_PIPE.RECEIVE_MESSAGE)'
[19:04:46] [INFO] testing 'Oracle time-based blind - Parameter replace (heavy queries)'
[19:04:46] [INFO] testing 'SQLite > 2.0 time-based blind - Parameter replace (heavy query)'
[19:04:46] [INFO] testing 'Firebird time-based blind - Parameter replace (heavy query)'
[19:04:46] [INFO] testing 'SAP MaxDB time-based blind - Parameter replace (heavy query)'
[19:04:46] [INFO] testing 'IBM DB2 time-based blind - Parameter replace (heavy query)'
[19:04:46] [INFO] testing 'HSQLDB >= 1.7.2 time-based blind - Parameter replace (heavy query)'
[19:04:49] [INFO] testing 'HSQLDB > 2.0 time-based blind - Parameter replace (heavy query)'
[19:04:51] [INFO] testing 'Informix time-based blind - Parameter replace (heavy query)'
[19:04:51] [INFO] testing 'MySQL >= 5.0.12 time-based blind - ORDER BY, GROUP BY clause'
[19:04:51] [INFO] testing 'MySQL < 5.0.12 time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[19:04:51] [INFO] testing 'PostgreSQL > 8.1 time-based blind - ORDER BY, GROUP BY clause'
[19:04:52] [INFO] testing 'PostgreSQL time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[19:04:52] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind - ORDER BY clause (heavy query)'
[19:04:52] [INFO] testing 'Oracle time-based blind - ORDER BY, GROUP BY clause (DBMS_LOCK.SLEEP)'
[19:04:52] [INFO] testing 'Oracle time-based blind - ORDER BY, GROUP BY clause (DBMS_PIPE.RECEIVE_MESSAGE)'
[19:04:52] [INFO] testing 'Oracle time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[19:04:52] [INFO] testing 'HSQLDB >= 1.7.2 time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[19:04:52] [INFO] testing 'HSQLDB > 2.0 time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[19:04:52] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[19:04:56] [INFO] testing 'Generic UNION query (random number) - 1 to 10 columns'
[19:05:00] [INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[19:05:04] [INFO] testing 'MySQL UNION query (random number) - 1 to 10 columns'
[19:05:07] [WARNING] parameter 'User-Agent' does not seem to be injectable
[19:05:07] [CRITICAL] all tested parameters do not appear to be injectable. If you suspect that there is some kind of protection mechanism
d/or switch '--random-agent'
[19:05:07] [WARNING] HTTP error codes detected during run:
400 (Bad Request) - 8108 times, 403 (Forbidden) - 32444 times, 500 (Internal Server Error) - 8107 times

```

D.2 ZAP Scanning Report



ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	1
Low	6
Informational	2

Alert Detail

Medium (Medium)	X-Frame-Options Header Not Set
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	https://login.myceci-hpc.be/project/ceci%20est%20un%20test
Method	GET
Parameter	X-Frame-Options
URL	https://login.myceci-hpc.be/_rpc/_ctx/role?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	X-Frame-Options
URL	https://login.myceci-hpc.be/_rpc/_ctx/role?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	X-Frame-Options
URL	https://login.myceci-hpc.be/_rpc/_ctx/role
Method	POST
Parameter	X-Frame-Options
URL	https://login.myceci-hpc.be/project/134?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	GET
Parameter	X-Frame-Options
URL	https://login.myceci-hpc.be/init-project/
Method	GET
Parameter	X-Frame-Options
URL	https://login.myceci-hpc.be/user/2885?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	GET
Parameter	X-Frame-Options
Instances	7
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
CWE Id	16
WASC Id	15
Source ID	3

Low (Medium)	X-Content-Type-Options Header Missing
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	https://login.myceci-hpc.be/static/js/bootstrap-3.0.0.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/project_type/read
Method	POST

Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/project_type/read?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/project/134?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	GET
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/user/available_login?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/supervisor/get_domains?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/static/js/jquery-1.10.2.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/init-project/
Method	GET
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/user/get_projects?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/static/fonts/glyphicons-halflings-regular.woff
Method	GET
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/institution/read?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/_ctx/role?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/static/js/relog.js
Method	GET
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/project/read?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/static/js/knockout-2.3.0.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/static/css/custom.css
Method	GET
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/_ctx/role?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/project/read
Method	POST
Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/_rpc/project/get_users?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST

Parameter	X-Content-Type-Options
URL	https://login.myceci-hpc.be/static/css/bootstrap-3.0.0.min.css
Method	GET
Parameter	X-Content-Type-Options
Instances	25
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://www.owasp.org/index.php/List_of_useful_HTTP_headers
CWE Id	16
WASC Id	15
Source ID	3

Low (Medium)	Incomplete or No Cache-control and Pragma HTTP Header Set
Description	The cache-control and pragma HTTP header have not been set properly or are missing allowing the browser and proxies to cache content.
URL	https://login.myceci-hpc.be/project/ceci%20est%20un%20test
Method	GET
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/static/css/bootstrap-3.0.0.min.css
Method	GET
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/institution/read?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/init-project/
Method	GET
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/project/read
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/project_category/read?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/_ctx/role
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/project/get_users?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/_ctx/role?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/project_type/read
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/user/2885?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	GET
Parameter	Cache-Control

URL	https://login.myceci-hpc.be/project/134?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	GET
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/supervisor/get_domains?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/user/get_projects?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/user/available_login?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/user/read?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/project_type/read?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/project/read?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/static/css/custom.css
Method	GET
Parameter	Cache-Control
URL	https://login.myceci-hpc.be/_rpc/_ctx/role?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	Cache-Control
Instances	20
Solution	Whenever possible ensure the cache-control HTTP header is set with no-cache, no-store, must-revalidate; and that the pragma HTTP header is set with no-cache.
Reference	https://www.owasp.org/index.php/Session_Management_Cheat_Sheet#Web_Content_Caching
CWE Id	525
WASC Id	13
Source ID	3

Low (Medium)	Web Browser XSS Protection Not Enabled
Description	Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server
URL	https://login.myceci-hpc.be/_rpc/_ctx/role
Method	POST
Parameter	X-XSS-Protection
URL	https://login.myceci-hpc.be/project/134?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	GET
Parameter	X-XSS-Protection
URL	https://login.myceci-hpc.be/sitemap.xml
Method	GET
Parameter	X-XSS-Protection
URL	https://login.myceci-hpc.be/user/2885?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	GET
Parameter	X-XSS-Protection
URL	https://login.myceci-hpc.be/project/ceci%20est%20un%20test
Method	GET
Parameter	X-XSS-Protection

URL	https://login.myceci-hpc.be/_rpc/_ctx/role?k=d1e9834b-4c16-4f54-b1c7-a74c397a54f7
Method	POST
Parameter	X-XSS-Protection
URL	https://login.myceci-hpc.be/init-project/
Method	GET
Parameter	X-XSS-Protection
URL	https://login.myceci-hpc.be/robots.txt
Method	GET
Parameter	X-XSS-Protection
URL	https://login.myceci-hpc.be/_rpc/_ctx/role?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	POST
Parameter	X-XSS-Protection
URL	https://login.myceci-hpc.be/_rpc/project/get_users
Method	POST
Parameter	X-XSS-Protection
Instances	10
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Other information	The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it: X-XSS-Protection: 1; mode=block X-XSS-Protection: 1; report=http://www.example.com/xss The following values would disable it: X-XSS-Protection: 0 The X-XSS-Protection HTTP response header is currently supported on Internet Explorer, Chrome and Safari (WebKit). Note that this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length).
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://www.veracode.com/blog/2014/03/guidelines-for-setting-security-headers/
CWE Id	933
WASC Id	14
Source ID	3

Low (Medium)	Information Disclosure - Debug Error Messages
---------------------	--

Description	The response appeared to contain common error messages returned by platforms such as ASP.NET, and Web-servers such as IIS and Apache. You can configure the list of common debug messages.
URL	https://login.myceci-hpc.be/_rpc/project/get_users
Method	POST
Evidence	Internal Server Error
Instances	1
Solution	Disable debugging messages before pushing to production.
Reference	
CWE Id	200
WASC Id	13
Source ID	3

Low (Medium)	Application Error Disclosure
---------------------	-------------------------------------

Description	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
URL	https://login.myceci-hpc.be/_rpc/project/get_users
Method	POST
Evidence	HTTP/1.1 500 Internal Server Error
Instances	1

Solution	Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.
Reference	
CWE Id	200
WASC Id	13
Source ID	3
Low (Medium)	Absence of Anti-CSRF Tokens
Description	No Anti-CSRF tokens were found in a HTML submission form. A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf. CSRF attacks are effective in a number of situations, including: * The victim has an active session on the target site. * The victim is authenticated via HTTP auth on the target site. * The victim is on the same local network as the target site. CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.
URL	https://login.myceci-hpc.be/init-project/
Method	GET
Evidence	<form id="main_form" role="form">
URL	https://login.myceci-hpc.be/user/2885?k=83ce5572-c01c-49a5-ab7b-f4ed33fc2f88
Method	GET
Evidence	<form role="form">
Instances	2
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard. Phase: Implementation Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script. Phase: Architecture and Design Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). Note that this can be bypassed using XSS. Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation. Note that this can be bypassed using XSS. Use the ESAPI Session Management control. This control includes a component for CSRF. Do not use the GET method for any request that triggers a state change. Phase: Implementation Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.
Other information	No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret] was found in the following HTML form: [Form 1: "acronym"].
Reference	http://projects.webappsec.org/Cross-Site-Request-Forgery http://cwe.mitre.org/data/definitions/352.html

CWE Id	352
WASC Id	9
Source ID	3
Informational (Medium)	Information Disclosure - Suspicious Comments
Description	The response appears to contain suspicious comments which may help an attacker.
URL	https://login.myceci-hpc.be/static/js/relog.js
Method	GET
URL	https://login.myceci-hpc.be/static/js/jquery-1.10.2.min.js
Method	GET
URL	https://login.myceci-hpc.be/static/js/knockout-2.3.0.min.js
Method	GET
Instances	3
Solution	Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.
Other information	<pre> // Fill user fields var prm = rpc('user', 'set_action', args); rpc('user', 'get_domains').done(function(data) { this.user = options.user; this.select = function() { this.level.user.select_institution(this.level, this); var User = function() { if (login == User.last_login) { User.last_login = login; var prm = rpc('user', 'available_login', [this.id(), login]); 'user': this}); // Set user value var prm = rpc('user', 'delete', [this.id()]); return rpc('user', 'set_status', [this.id(), 'waiting'], return rpc('user', 'set_status', [this.id(), 'confirmed']); var msg = User.board_transition[key]; this.set_ssh_public_key = function(user, ev) { User.prototype.save = function() { rpc('user', 'set_projects', [this.id(), data]); User.prototype.kick = rpc_kick; User.last_login = null; User.prototype.state_buttons = [User.prototype._model = 'user'; User.prototype.load = function(id) { Level.root.user = this; inst.select(); return rpc('user', 'get_projects', [this.id()]); User.prototype.read_fields = function() { if (this.role() == 'admin') { User.prototype.write_fields = function() { if (this.role() == 'admin') { User.board_transition = { "done-draft": "Reset user account", "waiting-draft": "Reset user account", var user = new User(); </pre>

```
// read user
return user.load(id);
user.available_domains(domains);
ko.applyBindings(user, $('body')[0]);
if (this.role() == 'user' && this.status() != 'draft'){
this.add_user = function(user) {
user.registered(true);
var user = users[pos];
if (user.registered() || user.status != 'done') {
var login = user.login().toLowerCase();
res.push(user);
if (this.role() != 'admin') {
'user', 'available_login', [this.id(), acronym, 'project']
var user = users[pos];
'id': user.id,
'registered': user.registered(),
if (!this.investigator() || this.role() != 'admin') {
return rpc('user', 'read', [], {
if (this.role() == 'admin') {
if (this.role() == 'admin') {
"waiting-draft": "Reset user account",
if (this.model().key == 'user') {
'key': 'user',
'label': 'User',
// 'model' : User,
'new_record': function() {return new User();}
// Select item wrt hash
```

Reference	
CWE Id	200
WASC Id	13
Source ID	3

Informational (Low)	Timestamp Disclosure - Unix
Description	A timestamp was disclosed by the application/web server - Unix
URL	https://login.myceci-hpc.be/static/css/bootstrap-3.0.0.min.css
Method	GET
Evidence	00000000
URL	https://login.myceci-hpc.be/static/css/bootstrap-3.0.0.min.css
Method	GET
Evidence	80000000
URL	https://login.myceci-hpc.be/static/css/bootstrap-3.0.0.min.css
Method	GET
Evidence	428571429
URL	https://login.myceci-hpc.be/static/css/bootstrap-3.0.0.min.css
Method	GET
Evidence	1428571435
Instances	4
Solution	Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.
Other information	00000000, which evaluates to: 1970-01-01 01:00:00

Reference	https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure http://projects.webappsec.org/w/page/13246936/Information%20Leakage
CWE Id	200
WASC Id	13
Source ID	3

List of Figures

1.1	Photos taken during the visit of Data Center III at the UCLouvain [5].	5
2.1	Statistics on the number of accounts registered on the Relog application.	10
2.2	CÉCI login management workflow, adopted from the Relog documentation.	12
2.3	CÉCI's LDAP architecture, adopted from the Relog documentation.	13
3.1	Example of a possible layout for the project creation form.	23
4.1	Sequence diagram of our different module interactions.	31
4.2	Layout of the top section of the page <code>university.html</code>	34
4.3	Layout of the bottom section of the page <code>university.html</code>	35
4.4	Layout of the different types of charts of the page <code>institutes.html</code> .	37
4.5	Layout of the top section of the page <code>institution.html</code> in which color themes have been applied for institutions belonging to the UCLouvain (red), ULiège (yellow) and ULB (blue).	38

List of Tables

A.1	Table illustrating the 2 different types of SSH keys	54
-----	--	----

Listings

2.1	Class <code>Institution</code> including directives to describe the database table <code>institution</code> of the Relog application to which it will be mapped.	14
2.2	Examples of routes defined in the Relog application using the <code>Bottle</code> 's <code>route</code> decorator.	15
2.3	Simplified example of the use of the <code>Knockout.js</code> library by the Relog application.	16
3.1	Example of modification made to the <code>project.html</code> file.	22
3.2	<code>Session</code> function of the module <code>confest.py</code>	26
3.3	Factory of the <code>ProjectType</code> class.	26
3.4	Example of tests present in our <code>test_project_set_status</code> module.	27
4.1	Code part present in our <code>base_template.html</code> file.	33
4.2	Directory tree structure of our project.	33
4.3	Content example of our configuration file.	36
5.1	Example of data in the HTTP request body that can be received by the Relog application.	41
5.2	Response of the Relog application to the request 5.1.	41
5.3	Example of data that can be used to exploit the <code>CVE-2019-7548</code> vulnerability.	42
5.4	Response of the Relog application to the request 5.3 with institutions ordered alphabetically according to the <code>name</code> attribute.	42
5.5	Part of the code of the <code>read</code> method of the file <code>models.py</code>	43
5.6	HTTP request sent to the Relog application originating from <i>www.w3schools.com</i>	44
5.7	Example of code that could be used in the case of a CSRF attack to remove a user from the Relog application.	45
A.1	Fixed default username size bug	51
A.2	Resolve Web UI misleading	52
A.3	Resolve New email template email part	53
A.4	Resolve New email template model part	53
A.5	Resolve SSH2 key format Front-End part	54
A.6	Resolve SSH2 key format Back-End part	55

Bibliography

- [1] CISM. Center for high performance computing and mass storage. <http://www.cism.ucl.ac.be>. Last accessed 25 October 2019.
- [2] Wikipedia. High-performance computing. <https://en.wikipedia.org/wiki/Supercomputer>. Last accessed 25 October 2019.
- [3] High performance computing. <https://uclouvain.be/en/research/cism/high-performance-computing.html>. Last accessed 10 may 2020.
- [4] CÉCI. Consortium des Équipements de calcul intensif. <http://www.cec-hpc.be>. Last accessed 9 November 2019.
- [5] UCL. Data center management (dciii). <https://uclouvain.be/en/research/cism/data-center-iii.html>. Last accessed 30 November 2019.
- [6] Tatu Ylonen. Secure shell documentation. <https://www.ssh.com/>. Last accessed 26 December 2019.
- [7] GitLab. Documentation. <https://about.gitlab.com/>. Last accessed 30 November 2019.
- [8] LDAP. Lightweight directory access protocol documentation. <https://ldap.com/>. Last accessed 30 November 2019.
- [9] Bottle. Python web framework. <https://bottlepy.org/docs/dev/>. Last accessed 30 November 2019.
- [10] Knockout. Documentation. <https://knockoutjs.com>. Last accessed 30 November 2019.
- [11] Mvvm and view models. <https://knockoutjs.com/documentation/observables.html>. Last accessed 21 May 2019.
- [12] SQLite. Documentation. <https://www.sqlite.org/index.html>. Last accessed 30 November 2019.

- [13] SQLAlchemy. Documentation. <https://docs.sqlalchemy.org/en/latest/>. Last accessed 30 November 2019.
- [14] François Damien. Slufl2. <https://github.com/damienfrancois/sluf12>, 2018.
- [15] Python’s web framework benchmarks. <https://klen.github.io/py-frameworks-bench/>. Last accessed 05 January 2019.
- [16] knockout/knockout. <https://github.com/knockout/knockout>. Last accessed 07 may 2020.
- [17] PRACE aisbl. Prace in a few words. <http://www.prace-ri.eu/prace-in-a-few-words/>, Sep 2019. Last accessed 10 December 2019.
- [18] Cenaero. Accessing the infrastructure. https://tier1.cenaero.be/en/accessing_infra. Last accessed 10 December 2019.
- [19] Cenaero. Cenaero - simulation technologies for aeronautics. <http://www.cenaero.be/>. Last accessed 9 January 2019.
- [20] CÉCI. Disk space. https://support.cec-hpc.be/doc/_contents/ManagingFiles/Storage.html. Last accessed 08 December 2019.
- [21] Gidnumber. <https://ldapwiki.com/wiki/GidNumber>. Last accessed 08 December 2019.
- [22] Jpreese. Best practices for writing unit tests. <https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices>. Last accessed 03 May 2020.
- [23] Sql transaction. <https://www.w3resource.com/sql/controlling-transactions.php>. Last accessed 17 May 2020.
- [24] factory-boy. <https://factoryboy.readthedocs.io/en/latest/>.
- [25] Ceci clusters usage graphs. <http://www.cec-hpc.be/graphs.html>. Last accessed 07 May 2020.
- [26] Numpy. <https://numpy.org/>. Last accessed 07 May 2020.
- [27] Visualization with python. <https://matplotlib.org/>. Last accessed 07 May 2020.
- [28] Jinja. <https://jinja.palletsprojects.com/en/2.11.x/>. Last accessed 07 may 2020.

- [29] Chart.js. <https://www.chartjs.org/>. Last accessed 07 May 2020.
- [30] Positive Technologies. Attacks on web applications: 2018 in review. <https://www.ptsecurity.com/ww-en/analytics/web-application-attacks-2019/>, Feb 2020. Last accessed 10 June 2020.
- [31] Sql injection. https://owasp.org/www-community/attacks/SQL_Injection. Last accessed 10 June 2020.
- [32] Sqlalchemy 1.3 documentation. <https://docs.sqlalchemy.org/en/13/core/tutorial.html>. Last accessed 12 May 2020.
- [33] A step-by-step sqlalchemy tutorial. <http://www.rmunnn.com/sqlalchemy-tutorial/tutorial.html>. Last accessed 10 May 2020.
- [34] Current cvss score distribution for all vulnerabilities. <https://www.cvedetails.com/>. Last accessed 07 May 2020.
- [35] Vulnerability details : Cve-2019-7548. <https://www.cvedetails.com/cve/CVE-2019-7548/>. Last accessed 10 May 2020.
- [36] Vulnerability details : Cve-2019-7164. <https://www.cvedetails.com/cve/CVE-2019-7164/>. Last accessed 10 May 2020.
- [37] sqlmap. <https://github.com/sqlmapproject/sqlmap>. Last accessed 20 May 2020.
- [38] Cross site request forgery (csrf). <https://owasp.org/www-community/attacks/csrf>. Last accessed 20 May 2020.
- [39] Owasp top 10 2007-2017: The fall of csrf. <https://nvisium.com/blog/2017/11/30/owasp-top-10-2007-2017-the-fall-of-csrf.html>. Last accessed 10 June 2020.
- [40] Cross-site request forgery prevention cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html. Last accessed 29 may 2020.
- [41] Same-origin policy (sop). <https://portswigger.net/web-security/cors/same-origin-policy>. Last accessed 01 June 2020.
- [42] Cross-origin resource sharing (cors). <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>. Last accessed 01 June 2020.

- [43] Cross site scripting (xss). <https://owasp.org/www-community/attacks/xss/>. Last accessed 03 June 2020.
- [44] Owasp top ten. <https://owasp.org/www-project-top-ten/>. Last accessed 03 June 2020.
- [45] Getting started. <https://knockoutjs.com/documentation/introduction.html>. Last accessed 11 June 2020.
- [46] Getting started. <https://knockoutjs.com/documentation/text-binding.html>. Last accessed 14 May 2020.
- [47] Vulnerability scanning tools. https://owasp.org/www-community/Vulnerability_Scanning_Tools. Last accessed 27 May 2020.
- [48] Open source web application security scanner. Last accessed 27 May 2020.
- [49] Vega vulnerability scanner. <https://subgraph.com/vega/index.en.html>. Last accessed 02 June 2020.
- [50] Zap. <https://www.zaproxy.org/>. Last accessed 01 June 2020.
- [51] Clickjacking. <https://owasp.org/www-community/attacks/Clickjacking>. Last accessed 28 May 2020.
- [52] Clickjacking defense cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html. Last accessed 27 May 2020.
- [53] Security headers. <https://owasp.org/www-community/Security-Headers>. Last accessed 28 May 2020.
- [54] Autocomplete enabled (password field). <https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/autocomplete-enabled-password-field/>. Last accessed 04 June 2020.
- [55] OpenBSD Project. Openssh documentation. <https://www.openssh.com/>. Last accessed 26 December 2019.
- [56] Simon Tatham. Putty documentation. <https://www.putty.org/>. Last accessed 26 December 2019.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl