

Improvements to the G-Lorep Federation of Learning Object Repositories

Federico Sabbatini, Sergio Tasso(✉), Simonetta Pallottelli, and Osvaldo Gervasi

Department of Mathematics and Computer Science,
University of Perugia, via Vanvitelli 1, 06123 Perugia, Italy
federicosabbatini96@gmail.com,
{sergio.tasso,simonetta.pallottelli,osvaldo.gervasi}@unipg.it

Abstract. The G-Lorep project of the European Chemistry Thematic Network (ECTN), based on a federation of distributed repositories of Molecular Science Learning Objects, leverages at present a "hybrid" centralized/distributed architecture in which the central node hosts a shared database. The shared database deals only with the task of managing metadata in order to synchronize the information made available to the federation members at regular time intervals. To avoid security problems and reach a better code updating, the project has been migrated from *Drupal* CMS to *Laravel* framework. All modules have been written from the start, also a REST API server used as interface to the shared database is implemented and related performances are evaluated.

Keywords: Cloud · Distributed systems · Learning objects · Repositories · Security · Synchronization

1 Introduction

G-Lorep [1] is a technology aimed at facilitating the sharing of *Learning Objects* (LOs) among distributed repositories (whose typical scheme is illustrated in Fig. 1). For this reason G-Lorep leverages the assembling of a federation of repositories among the members of a scientific community wishing to share their LOs and to offer them to other community members for further development. Each federate in its domain has the same web application with its own local database in order to make users feel, even visually, to be using the same environment even when they are operating on a different federate.

The network structure adopted by G-Lorep is hybrid. This permits each website to work stand-alone like (with no need to communicate with the other servers) and to use a central database to get information on the LOs made available on the network. Accordingly, each federate can contact autonomously the central node whose tasks are only to distribute metadata and to allow each member of the federation to update the learning objects through a simple operation of synchronization occurring at regular time intervals [2]. In particular, when an "update" request is received, the member refreshes its own database adding the

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12255, pp. 526–537, 2020.
https://doi.org/10.1007/978-3-030-58820-5_39

new data received, creating an image of the LO equal to the original one and providing a link to the federate of origin [3,4].

In the G-Lorep initial implementation the central node was placed on a server physically located inside the University of Perugia. This, while guaranteeing a more direct control of the system, was reducing its fault tolerance in the case of Internet communication break-up causing miscommunication among the federates. An additional weakness of the initial implementation was the impossibility of guaranteeing that good practices of security (like the adoption of *HTTPS*, the automatic updating of *Drupal* [13] and Linux, the configuring of *SSH* and Firewall) would be adopted.

Accordingly, the paper focuses on the full restyling of the G-Lorep obtained by migrating from *Drupal* CMS to *Laravel* [12] framework as discussed in Sect. 2 where the new G-Lorep federate implementation is described and Sect. 3 where the new approach to shared database through a REST API server is detailed. In Sect. 4 is described how G-Lorep will be distributed through a interactive script that installs *Docker* [14] images, finally in Sect. 5 some conclusions are drawn and some directions of related future work are outlined.

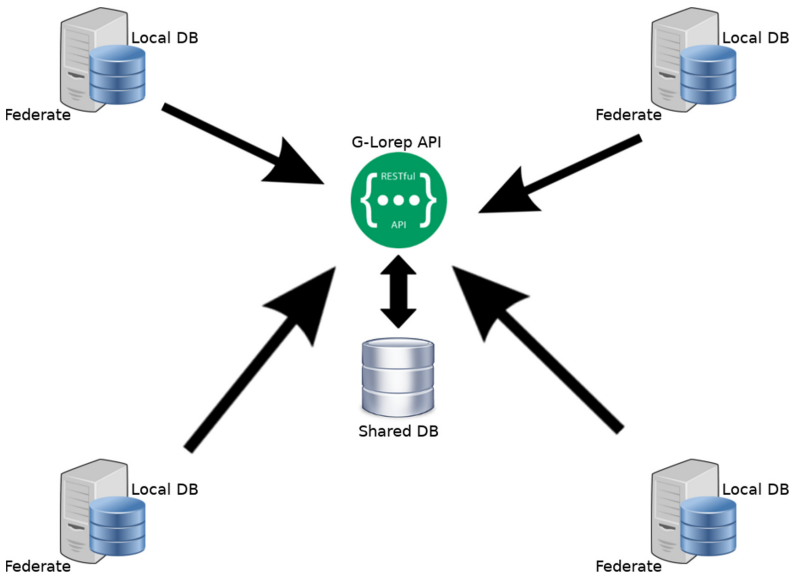

Fig. 1. New structure of the G-Lorep project federation

2 G-Lorep Migration to *Laravel*

2.1 How G-Lorep Was Structured

The distribution of the old version of G-Lorep was structured in three main components:

- the shared database which aims to store only metadata of each learning object;
- the *php_socket_io* located in each federate machine whose task is performing the synchronization between his host and the shared database;
- the latest *Drupal* 7 version with installed modules created to manipulate learning objects.

In each of these components there are security problems if we have a look to the future. *Drupal* 7 will be abandoned in november 2021, so this can cause some entry points for future attacks; furthermore actually there is not an automatic update system, so each webmaster should perform updates of his/her own platform; moreover it is easy to think *Drupal* being a really famous CMS put it at the center of interests of attackers that mass hacking websites created with CMS exploiting feature that often are not used. In the case of G-Lorep the unique functions used are modules and user management. Looking in this way there are no vantages in keeping updated G-Lorep project over this platform. So it has been decided to migrate the system to *Laravel*, one of the most famous PHP frameworks. It permits to easily maintain the code, keeping it more safe, because it will be harder to become a target of attackers. Furthermore, adding new features taking advantage of powerful functions of *Laravel* will be easier. During this migration new features have been implemented and they will be described in the next subsection.

If an attacker gains access to a federate, the *php_socket_io* daemon and shared database can be a security issue. The daemon can be considered like a client of the shared database, in fact it allows the direct access to the database and normally, via its software permits, each federate can manipulate its own learning objects over the shared database. But if an attacker edits the deamon or steals database credentials, he/she can full manipulate the shared database, and this is a serious problem. Taking advantage of the cronjob system directly managed via *Laravel* and creating a RESTFUL API server that acts like an interface to the shared database one can protect the central node from malicious attackers. This synchronization system will be described in the next section.

2.2 The New G-Lorep

The principal difference between old and new G-Lorep it's its structure. In *Drupal* modules it is needed to handle all parts of platform from scratch, from the database to the query functions, from the data manipulation to their visualization via HTML. In *Laravel* these phases are well separated following the MVC (**M**odel **V**iew **C**ontroller) pattern (Fig. 2).

Migrations and Models are the main features of *Laravel*. With migrations one can create, drop and alter existing tables of a database without writing SQL commands. Models work like an interface to the database, using them you can build customized queries which can be performed many times. For example an automatic `join` between two or more tables can be handled just calling a specific function declared in one of their classes. Furthermore *Laravel* has many predefined functions like `TableName::find($id)`, which returns a single row filtered by its primary key (in pure PHP is needed to write full SQL query, then run that, finally extract the first row from the returned array if its result is not null) or `TableName::firstOrCreate`, which returns a record matching specific conditions, and if it does not exist it will be created than returned.

In G-Lorep a massive usage of these features have been adopted to perform `joins` between a table containing the list of IDs of all learning objects (and others information) and relative table, that can be local learning objects or other federate learning objects. Furthermore, adding future features to the G-Lorep project will be really easy because using Migrations and Models it is just needed to edit their functions instead of navigating full website code searching where to apply modifications.

Routes and Middlewares keep coding funny and at the same time secure. Using routes you can communicate to a server. When a client is visiting a certain specific URL, the server must be up and running. Building structured routes is easy, because you can create groups to aggregate them, for example all URIs like `/learning_object/edit`, `/learning_object/show` and `/learning_object/new` can be merged into a group called `/learning_object/` and `edit`, `show` and `new` can be treated like its children; it is just like a switch that distributes packages between hosts of a network instead of linking each device directly to the router.

With middlewares, rules (and other operations) to access a certain route are described and applied without efforts. For example a programmer can forget to check if a user is logged or not to let him/her access a certain resource. Using middleware the rule is matched before the function is called, in this way the programmer does not need to bother himself/herself checking anything. Definitively middlewares can be applied to whole groups, for example can be created a middleware that checks if a visitor is a logged user to the whole group `/your_profile/` inheriting it to its children `show` and `edit` too.

In G-Lorep a huge usage of a middleware which checks if a user is logged as author has been adopted because many functions related to learning objects like downloading of attachments or creation of new LOs are permitted only to authorized users.

Controllers are the *heart* of *Laravel* because functions are written inside them. They are called via routes, then a specified function is executed; with a function you can run queries via Models or Eloquent query builder of *Laravel*, then they return results like JSON, HTML, or better **views**. Using the `with()` function a controller can pass not only results of queries, but any kind of variable like

informative messages to the called view. Then the view will have the variable declared by itself without the need to call any other function.

Views are returned from controller, they are pieces of HTML (and PHP code). In *Laravel* they are written using *Blade* templates. *Blade* is the simple, yet powerful templating engine provided with *Laravel*. Unlike other popular PHP templating engines, *Blade* does not restrict the programmer from using plain PHP code in the views. In fact, all *Blade* views are compiled into plain PHP code and cached until they are modified, meaning *Blade* adds essentially zero overhead to the application. In this way you can add generated variable from controller into the HTML without writing PHP code or worry about Cross Site Scripting (XSS) attacks. Moreover other PHP, JavaScript or HTML files can be incorporated into a view just using few characters, indeed you can print information, error or warning messages to the user without writing the code every time. A programmer can create and import a different *Blade* file containing code checking if a passed variable through controller is set and in that case it prints nice looking HTML messages.

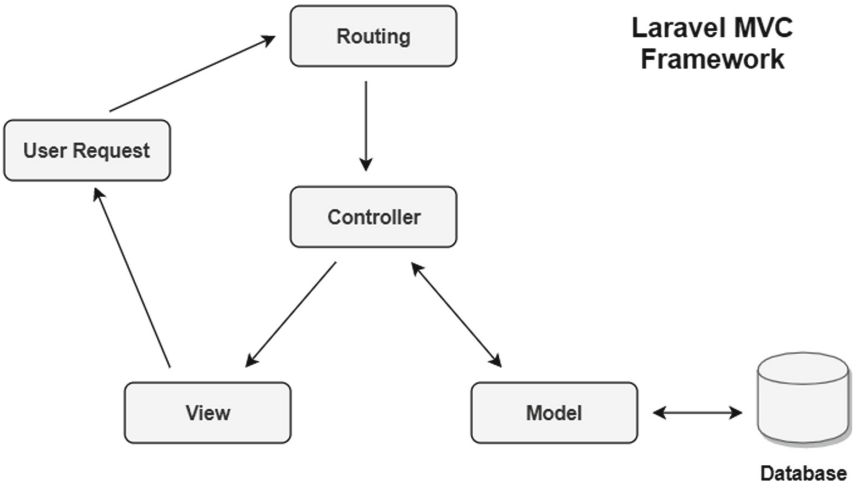

Fig. 2. *Laravel* model view controller representation

2.3 New Features in G-Lorep

During the migration many features have been directly implemented in the new platform. These features are added to the existing ones of the old version. In the old version of G-Lorep learning objects are stored following standards of Learning Object Metadata (IEEE LOM), but in this version many information are automatically compiled. For example, in the old version through writing title and description of a LO you could already obtain a relative suggested category

based on provided terms thanks to *Taxonomy Assistant* [5]. In the new version this mechanism has been improved to make it more accurate (Fig. 3). Moreover it has been used to automatically suggest synonyms words for the description and keywords to obtaining a best cataloging of the learning object that joins the already existing differentiation by category [6].

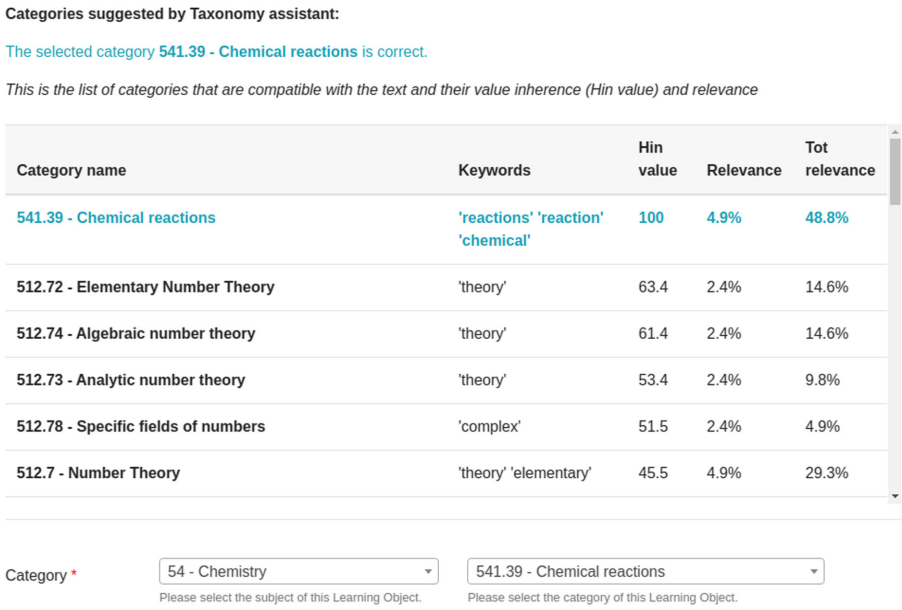

Fig. 3. New layout of *Taxonomy Assistant*

Computer Science. G-Lorep was initially designed to catalogue only Chemical learning objects, later it was extended to manage learning objects of other scientific subjects too, so Mathematics categories were added to the platform. From now on it will include Computer Science, too.

Management of Attachments. In G-Lorep *Drupal* version, if someone knew the structure of the website he/she could download attachments without the need to authenticate himself/herself. In the new version files are provided via a function instead of directly provide their URL. Furthermore files are renamed with a secret random string keeping them safe from being downloaded without using the required function, that is protected via the middleware for the authentication and authorization. In this way files are inaccessible and can be downloaded only from authorized logged users calling a certain route with a middleware.

Links as Attachment. Before the migrations you could not directly include a link as attachment. For example, if an author would like to add a not listed YouTube video as LO, he/she should have to create a `txt` file, write inside the YouTube link, then add this file as attachment to the object. Later a reader of that learning object should open the attachment in a tab of the browser, than copy and paste the link in a new tab. It was just a workaround to accomplish that, but now it is directly implemented permitting to give a name to a certain URL, too.

New Management of Keywords. As explained before, keywords are now automatically generated. A user can create a learning object adding suggested keywords or typing new ones by himself/herself. Most used are directly showed like a popup in the input box. Thanks to this feature, now keywords are effectively used, and the search engine has been improved adding a research via keywords, in addition to the already existent search via category, title, description, authors and other metadata.

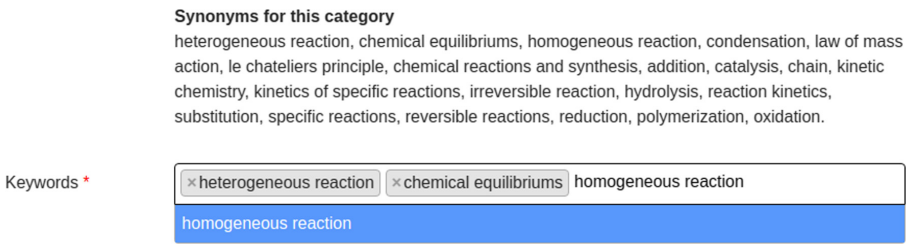

Fig. 4. New management of keywords

Visible Statistics for All Users. Now statistics of usage of learning objects, like views of the single LO or clicks and downloads of their attachments are fully logged and showed to users. Thanks to this feature tabs can be added in the learning object feed that shows the most interacted objects ordering them by views or interactions to their attachments.

Multilingual. A system that manages localization of the federate has been provided in the new platform. Now a client can select the wished language for the navigation of the website. This also allows to keep the editing of texts easy, because there is no need for a programmer to search the view or the controller that uses a certain text (maybe it has more occurrences too) and edit it, just apply modifications to localization files is now required.

Maintaining G-Lorep Updated into All the Federation. Finally, with this last but not least feature all federates website can be keept updated without concerning each webmaster to manually update it. It is just necessary debugging a new version into a development environment, then the updating to the new version of G-Lorep will be automatically propagated to all the networks of federation without paying attention to eventual incompatibility between security updates and installed modules for managing learning objects like in the old G-Lorep based on *Drupal*.

3 The New Architecture of G-Lorep Synchronization

At regular interval of time a synchronization between each federate and shared database is performed. Using this hybrid approach, if the shared database goes down only the synchronization between federates is compromised, still permitting the navigation of every single federate, visualization of foreign LOs of previous synchronizations will be still accessible. Similarly if a federate goes offline only the access to its attachments and the synchronization between its learning objects and the rest of federation is temporary compromised. Then when the server comes again up the synchronization will be performed and attachments will be again accessible.

3.1 The Central Node Architecture

In the first version of G-Lorep the shared database was located in a virtual machine inside the Department of Mathematics and Computer Science of University of Perugia. Last year the shared database migrated into the Cloud taking advantage of *PaaS* services like *Amazon RDS* [1]. This solution permitted to reach a great level of resiliency and availability and a good level of security (thanking the whitelist of authorized IP addresses and the infrastructure of Amazon), because the safety of the service was full managed by third parties. As said in the previous section, however despite it keeps the central node secure from external attacks, it does not protect it from authorized hacked federates, permitting in this way malicious queries.

The solutions was creating a RESTFUL API server that acts like an interface to the shared database. In this way the only authorized host accessing to the database is the one owning the API, then each federate needs to contact the central node asking for new learning objects or sending its own updated tables.

Access to the RESTFUL API server is permitted only to authenticated federate behind a white-listed IP address, using an architecture of this type permits to add a additional security layer because it is not possible anymore run customized queries.

The central node is written in Python3 using *Flask* [15] as framework to manage routes, then *SQLAlchemy* [16] is used to perform queries.

Flask has been choosen because it is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It is not a great solution for websites

because using it standalone it does not permit to manage layouts, but in the G-Lorep case only JSON requests and responses are required, furthermore only few routes are needed to upload and download learning objects composed by metadata, information about attachments and eventually the relation tables between learning objects. So *Flask* represented for G-Lorep a great solution, easy to code and to maintain.

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language; it is used by famous organizations like OpenStack Project. Like *Laravel* it allows to use models to create table and perform queries, so it represented a great opportunity to write clearer code without the need to write pure SQL.

The approach to the shared database is quite similar to the last version: the federate sends new or updated learning objects to the server; the server checks if the ID of a certain learning object is already in the database and if a record is returned it is deleted, then each learning object is added to the shared database keeping trace of which federate sent it. Deleting then adding updated learning objects instead of directly update them permits to assign them a new incremental ID by the central node to each received learning object. In this way a federate in the synchronization phase just needs to ask the G-Lorep synchronizer to return learning objects having the ID bigger than the received ID of the latest synchronization. A similar approach could be reached using timestamps, but sometimes it can happen that databases or web servers are in different timezones, and often they are not set to use the Unix timestamp. In this way the timestamp is not used for the synchronization of the federation, avoiding problems of this type.

3.2 The Federate Synchronization Architecture

The federate synchronization architecture is easy to understand (Fig. 5):

- when a new LO is created or an old one is edited, the timestamp of that LO inside the local database is updated
- at regular intervals of time the cronjob of each federate runs sending metadata of learning objects to the central node. These metadata have the timestamp greater than the one of the last synchronization
- the central node saves learning objects and adds the ID of the authenticated federate who sent those LOs to the metadata
- if LOs are successfully stored the federate updates the timestamp of the last synchronization
- the local node checks the biggest ID among the foreign LOs received in the previous synchronization and it asks the central node for a list of LOs having the ID bigger than that number
- the central node sends requested metadata of LOs owned by different federates
- finally the received data are saved and showed inside of the federate website.

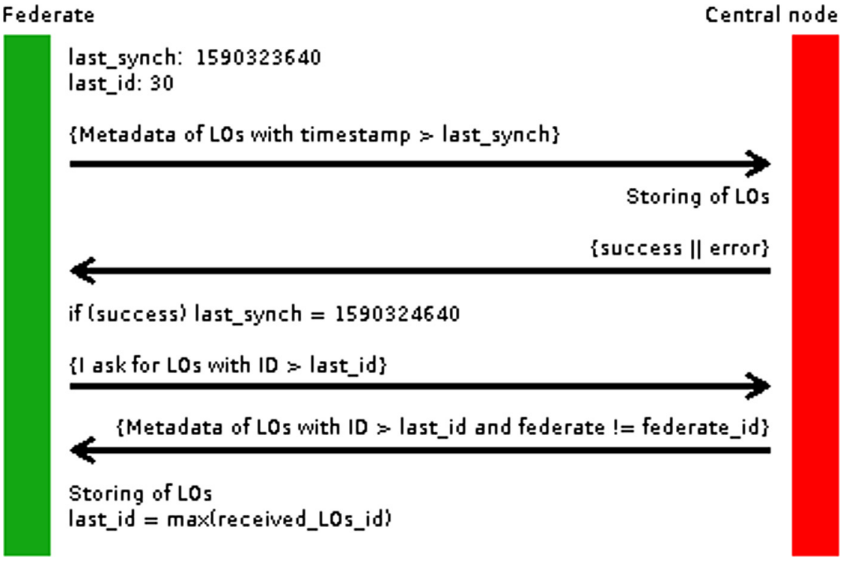

Fig. 5. The federate synchronization architecture

4 Distribution of G-Lorep

This version of G-Lorep has been studied to be easy to install but at the same time easy to customize.

The provided script automatically installs `Docker` and `Docker-compose`, then adds the current user to the `docker` group. After reloading, the privileges of the user latest version of project is downloaded and unpacked in the home directory, then configurations file for Docker-compose are created. A Docker image of G-Lorep is created starting from the official `php-fpm-alpine` with installed dependencies like PHP extensions and composer, this G-Lorep Docker image is the *PHP-FPM* server.

Through `mysql:5.7` and `nginx:alpine` images database and web server are installed, and through Docker-compose are managed to work together, in particular the installed Nginx will use the G-Lorep container as `fastcgi_pass`. *FastCGI* is a PHP protocol used by the web server; in the initialization phase the web server using *FastCGI* creates a process, then when a request arrives the server send the request to this created process, the *FastCGI* process then execs PHP code and returns the output (generally in HTML or JSON) to the web server which in turn will send it to the client, in this way the number of PHP process is optimized because only in the startup phase processes are created. *PHP-FPM* (**F**astCGI **P**rocess **M**anager) is a more recent alternative `version` of *FastCGI*, it implements additive functionality useful to websites with high traffic. *PHP-FPM* works similar to *PHP FastCGI* and it is focused on process optimization too, the main difference is that with *PHP FastCGI* the process is created directly from the web server, instead with *PHP-FPM* a different

service is created to manage by itself PHP executions. *PHP-FPM* creates *pools* for replying PHP requests and generated process are *childs* of the Process Manager which can be separately managed from the web server. The use of this approach permits to obtain a better service high-availability because processes restart will impact only FPM pools and not the web server anymore.

Finally the user will be prompted to optionally install a SSL certificate provided by Let's Encrypt and *HTTPS* is automatically activated. Let's Encrypt is a free, automated, and open certificate authority (CA) provided by the Internet Security Research Group (ISRG), run for the public's benefit. It permits to create or update automatically in few seconds secure and free certificates.

5 Conclusion and Future Work

The G-Lorep federation of distributed repositories, previously based on *Drupal* CMS, has been redesigned and implemented on *Laravel* framework. As explained above, there are many advantages in this new version. In fact, in addition to the security aspects, new user features have been introduced and the federation itself has been equipped with a more efficient synchronization mechanism.

For years, the G-Lorep federation's organization of distributed repositories has been suitable to create, assemble, store and retrieve Chemistry Learning Objects in a cooperative way [7–10]. Now, it can also open up to two other areas such as Mathematics and Computer Science, even making use of a future general G-Lorep located in the Cloud, where everyone can insert their own learning objects regardless of their institution of origin [11].

Acknowledgements. The authors acknowledge ECTN (VEC standing committee) and the EC2E2N 2 LLP project for stimulating debates and providing partial financial support.

References

1. Tasso, S., Pallottelli, S., Gervasi, O., Sabbatini, F., Franzoni, V., Laganà, A.: Cloud and local servers for a federation of molecular science learning object repositories. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11624, pp. 359–373. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24311-1_26
2. Tasso, S., Pallottelli, S., Gervasi, O., Tanase, R., Rui, M.: Synchronized content and metadata management in a federation of distributed repositories of chemical learning objects. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10406, pp. 14–28. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62398-6_2
3. Tasso, S., Pallottelli, S., Bastianini, R., Lagana, A.: Federation of distributed and collaborative repositories and its application on science learning objects. In: Murgante, B., Gervasi, O., Iglesias, A., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2011. LNCS, vol. 6784, pp. 466–478. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21931-3_36

4. Tasso, S., Pallottelli, S., Rui, M., Laganá, A.: Learning objects efficient handling in a federation of science distributed repositories. In: Murgante, B., et al. (eds.) ICCSA 2014. LNCS, vol. 8579, pp. 615–626. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09144-0_42
5. Tasso, S., Pallottelli, S., Ferroni, M., Bastianini, R., Laganà, A.: Taxonomy management in a federation of distributed repositories: a chemistry use case. In: Murgante, B., et al. (eds.) ICCSA 2012. LNCS, vol. 7333, pp. 358–370. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31125-3_28
6. Pallottelli, S., Tasso, S., Rui, M., Laganà, A., Kozaris, I.: Exchange of learning objects between a learning management system and a federation of science distributed repositories. In: Gervasi, O., et al. (eds.) ICCSA 2015. LNCS, vol. 9156, pp. 371–383. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21407-8_27
7. Tasso, S., Pallottelli, S., Laganà, A.: Mobile device access to collaborative distributed repositories of chemistry learning objects. In: Gervasi, O., et al. (eds.) ICCSA 2016. LNCS, vol. 9786, pp. 443–454. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42085-1_34
8. Tasso, S., Pallottelli, S., Gervasi, O., Rui, M., Laganà, A.: Sharing learning objects between learning platforms and repositories. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10963, pp. 804–816. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95171-3_62
9. Franzoni, V., Tasso, S., Pallottelli, S., Perri, D.: Sharing linkable learning objects with the use of metadata and a taxonomy assistant for categorization. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11620, pp. 336–348. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24296-1_28
10. Laganà, A., Gervasi, O., Tasso, S., Perri, D., Franciosa, F.: The ECTN virtual education community prosumer model for promoting and assessing chemical knowledge. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10964, pp. 533–548. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95174-4_42
11. Laganà, A., Terstyanszky, G., Krüger, J.: Open molecular science for the open science cloud. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10406, pp. 29–43. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62398-6_3
12. Laravel.: https://laravel.com/. Accessed May 2020
13. Drupal.: https://www.drupal.org/. Accessed May 2020
14. Docker.: https://www.docker.com/. Accessed May 2020
15. Flask.: https://flask.palletsprojects.com/en/1.1.x/. Accessed May 2020
16. SQLAlchemy.: https://www.sqlalchemy.org/. Accessed May 2020