Next-Lab

Next Generation Stakeholders and Next Level Ecosystem for Collaborative Science Education with Online Labs

Innovation Action in European Union's 2020 research and innovation programme

Grant Agreement no. 731685

Deliverable 4.4

Release of the consolidated underlying infrastructure (M24)

Editor	Denis Gillet (EPFL)
Date	21st December 2018
Dissemination Level	Public
Status	Final

The Next-Lab Consortium

Beneficiary Number	Beneficiary name	Beneficiary short name	Country
1	University Twente	UT	The Netherlands
2	École Polytechnique Fédérale de Lausanne	EPFL	Switzerland
3	IMC Information Multimedia Communication AG	IMC	Germany
4	EUN Partnership AISBL	EUN	Belgium
5	Ellinogermaniki Agogi Scholi Panagea Savva AE	EA	Greece
6	University of Cyprus	UCY	Cyprus
7	Universidad de la Iglesia de Deusto	UD	Spain
8	Tartu Ülikool	UTE	Estonia
9	Núcleo Interactivo de Astronomia Associacao	NUCLIO	Portugal
10	Ecole Normale Superieure de Lyon	ENS de Lyon	France
11	Turun Yliopisto	UTU	Finland
12	University of Leicester	ULEIC	United Kingdom

Contributors

Name	Institution
María Jesús Rodríguez-Triana, André Lemos Nogueira, Juan Carlos Farah, Hassan Abdul Ghaffar, Yves Piguet, Denis Gillet	EPFL
Pablo Orduna	DEUSTO
Internal Reviewers	
Margus Pedaste	UTE

Legal Notices

Executive Summary

As stated in the grant agreement, the work reported in this deliverable is related to Task 4.2 that deals with the underlying Next-Lab infrastructure. It includes the definition and the implementation of APIs enabling the loose coupling between the Next-Lab platforms, such as the authoring, the sharing and the tutoring platforms, for exchanging and disseminating apps, labs and spaces. It also includes the gateway enabling to federate 3rd parties science repositories offering additional online labs and open educational resources exploited by science teachers.

The main adaptations of this infrastructure during the second year of the project towards sustainability and higher Technical Readiness Level (Task 4.3) include a new Graasp API which is enabling the integration of apps and labs without the need to rely on the outdated shindig server and the opensocial wrapping, an improved printing service supporting the creation of eBook and integrated in a new distributed infrastructure, a new framework to develop and host modern labs and apps, as well as support to migrate the full ecosystem to https.

.

Table of Contents

1 Introduction

As stated in the grant agreement, the work reported in this deliverable is related to Task 4.2 that deals with the underlying Next-Lab infrastructure. It includes the definition and the implementation of **APIs** enabling the loose coupling between the Next-Lab platforms, such as the authoring, the sharing and the tutoring platforms, for exchanging and disseminating apps, labs and spaces. It also includes the **gateway** enabling to federate 3rd parties science repositories offering additional online labs and open educational resources exploited by science teachers.

The main adaptations of this infrastructure during the second year of the project towards sustainability and higher Technical Readiness Level (Task 4.3) include a new Graasp API which is enabling the integration of apps and labs without the need to rely on the outdated shindig server and the opensocial wrapping, an improved printing service supporting the creation of eBook and integrated in a new distributed infrastructure, a new framework to develop and host modern labs and apps, as well as support to migrate the full ecosystem to https.

2 New Graasp API

As of this moment, apps running on Graasp that need data persistency - due to the need of saving/fetching app related data - are relying and using, under the hood, a service called Shindig.

Since a while now, this service is no longer supported by the community and has been deprecated, which raises big concerns regarding sustainability.

To respond to the growing need of decommissioning this current solution, of building something simpler and scoped to our needs, and under Graasp's "umbrella", we built the new Graasp API.

The Graasp API responds to the major needs of apps running in a Graasp space/ILS context, by providing a clear and simple API that allows apps to:

- Save and retrieve user's inputs;
- Get data regarding the execution context - information about the space/ILS, current user, etc.;
- Create and fetch actions - data describing user interactions with the app;
- Transparently handle user management and permissions.

Although the need of replacing Shindig, and its functionalities, was the main reason that triggered the creation of Graasp API, its implementation has since seen some other developments and added features to also power other clients, namely Graasp Pages - a new web application for visualizing spaces and ILSs (discussed in more detail in D2.7).

Following this line, other future client applications might have their needs fulfilled with the new Graasp API as we continue its development. Furthermore, while designing the structures for storing data and action logs via the API, we discovered that different application providers had different requirements for the formats in which they store their data. To this end, we made the interface flexible enough to support any format a developer might require (e.g. xAPI, ActivityStreams), along with a field to specify the format used for clients that require reading the data. This flexibility allows our API to be usable by any application, with interoperability achievable by knowing how to transform from one data storage format to another. This responsibility is left to the developers, who will best know the needs for their respective applications.

Graasp API is also a service on its own, built from scratch in a different codebase, that only shares the data layer with Graasp - the MongoDB database.

This separation from Graasp's main codebase (which is considerably large) allowed us, among other things, to implement the new API using more recent technologies. Also, since it was thought through, from the start, as a solution to run in the cloud (in our case we are using AWS - Amazon Web Services), it can now benefit from out-of-the-box redundancy and scalability, which the existing Shindig solution cannot praise about - it’s a single service, running in a single local machine.

To support and facilitate the adoption and usage of Graasp API, for current and future app developers, we also provide comprehensive documentation on how to use it. This documentation, which can be found at http://developers.graasp.eu, is open-source, meaning that developers can also contribute and help in improving it (more about the Developers Portal in section 4.6).

2.1 API calls

The main API entities and corresponding endpoints available to the app developer are:

- **app-instance**: represents the application in the space/ILS, and contains the app configuration, its location (URL), the application name, etc.

HTTP Method	**API Endpoint path**	**Purpose**
GET	/app-instances/<id>	get app-instance
PATCH	/app-instances/<id>	update app-instance

- **app-instance-resource**: bound to both the app-instance and a user, it holds the data generated when the user interacts with the app.

GET	/app-instance-resources	get app-instance-resources
GET	/app-instance-resources/<id>	get app-instance-resource
POST	/app-instance-resources	create app-instance-resource
PATCH	/app-instance-resources/<id>	update app-instance-resource
DELETE	/app-instance-resources/<id>	delete app-instance-resource

- **action**: a record of metadata concerning the user interaction with the app (example: "user clicked on button A")

POST	/actions	create action
GET	/actions/<id>	get action
GET	/actions	get actions

3 New Graasp distributed architecture

The Figure 1 below provides an overview of the Graasp architecture with part of the services relying on the Amazon distributed service architecture, including the printing service to generate pdf versions of the inquiry learning spaces and eBooks, as well as the new app and lab repository.

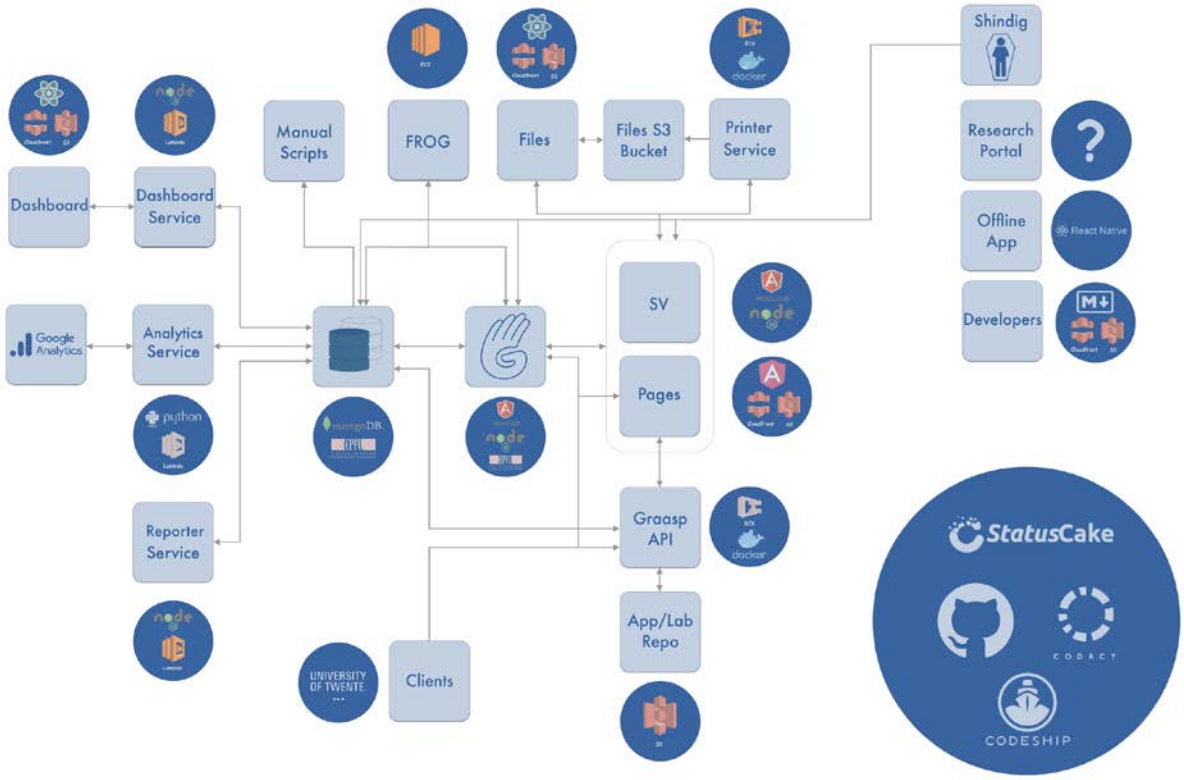


Figure 1. The current distributed Graasp architecture with its main services

In the center of the diagram we find the core Graasp servers along with the MongoDB instances hosted at EPFL. A number of new peripheral platforms are connected to this core component including most notably the following:

- **Pages**: the new viewer for Graasp regular or learning spaces that provides wider screen size for students and a better vertical navigation scheme between the phases. It also supports apps and labs not relying on the outdated shindig technology;
- **Graasp Dashboard**: the new analytics service that provided administrative figures from Google's analytics service as well as from the Graasp database through locally ran scripts. This dashboard is only accessible by authorized partners (National Expert Center (NEC) representatives and ambassadors) and has hence it own user management system;
- **Graasp Offline App**: an offline viewer that enables students to use their phones/tablets to view and share spaces. This part is developed in the framework of the GO-GA project to support requirements for implementation in under-connected areas. It is not only useful for schools in Africa, but also for most poorly connected schools in Europe;
- **Graasp and third party apps and labs**: consisting of Graasp servers along with apps and labs provided by clients. This new service emerges from the need to host apps and labs collections with all their dependencies in the Graasp.eu domain when learning spaces are integrated for off-line usage. This part is also developed in the context of the GO-GA project. Details are provided in Section 4 below.

Furthermore, there are a number of backend services that support the aforementioned platforms and the Graasp core components including but not limited to the following:

- **Graasp API**: the new service described in Section 2 that allows the integration of third party apps and labs without having to pass through the old shindig server;
- **Printer**: service that allows the students to print a space as a pdf document, a epub eBook, or a png image;
- **Analytics and reporter:** services used to populate our MongoDB instances with figures from Google Analytics and formatting them in form of accumulated reports;
- **FROG**: third party apps that offer collaboration between the users. Interesting examples include a chat application, a PDF annotator, and a spreadsheet tool, where users can collaborate in real time.

4 New labs & apps development framework

The new Graasp API introduced in Section 2 enables the emergence of a new ecosystem of apps and labs that do not depend on the deprecated Shindig server. In this section we detail the new features, frameworks and infrastructures that have been implemented in order to bolster the Graasp and Golabz ecosystem, and foster a community of app and lab developers.

4.1 Apps and Labs

Apps and labs are essentially standalone HTML5 web pages that can be exploited independently or embedded into a Graasp learning space. In principle, apps and labs can be hosted in any infrastructure. Nevertheless, in order to provide a broad set of functionalities whilst guaranteeing an adequate level of security we require that any app or lab that wants to interact with the Graasp API be hosted under the graasp.eu domain. This motivated us to design a framework to support app and lab developers, allowing them to submit their apps and labs to our cloud infrastructure, from which they will be accessible to Graasp users and the Golabz repository.

4.2 Cloud Infrastructure

Our cloud infrastructure for apps and labs consists of four Amazon Web Services (AWS) Simple Storage Service (S3) buckets, which are essential drives hosted in the Amazon Infrastructure. These four buckets correspond to the following:

1. Graasp Applications in Development
2. Graasp Applications in Production
3. Graasp Labs in Development
4. Graasp Labs in Production

The distinction between development and production is to allow developers to fully test their apps and labs using the Graasp development environment (dev.graasp.eu). The distinction between apps and labs is to align ourselves with the way these learning resources are organized and exposed to end users both on Graasp and Golabz.

Each of these buckets are cached and served by AWS' Cloudfront content delivery network (CDN). This CDN creates copies of the resources in the bucket and deploys them to data centres across the world, reducing the time required to do the round-trips necessary for the end clients to fetch the content.

4.3 Starter Kits

Our goal in implementing this framework for apps and labs was also to reduce the barrier to entry for developers to create learning resources that would be out-of-the-box compatible with the Graasp infrastructure. With this in mind, we deployed two starter kits, available in the React and Angular JavaScript frameworks. Developers can use these starter kits as a solid base for the apps or labs that they would like to develop. The starter kit provides the scaffolding needed to follow programming best practices and plug into the Graasp API. Its features include, but are not limited to the following:

- AWS Setup Scripts
- Code Style Linting
- Testing
- Code Coverage
- Internationalization
- Sample Viewing Modes (Student, Teacher, Print, Offline, etc.)
- Sample Graasp API Calls
- Continuous Integration and Deployment
- Offline Support
- Semantic Versioning

These starter kits are open source and are freely available on GitHub. As a proof-of-concept, we have already developed and deployed a few sample apps and labs built using the React version of the starter kit (see Fig. 2).

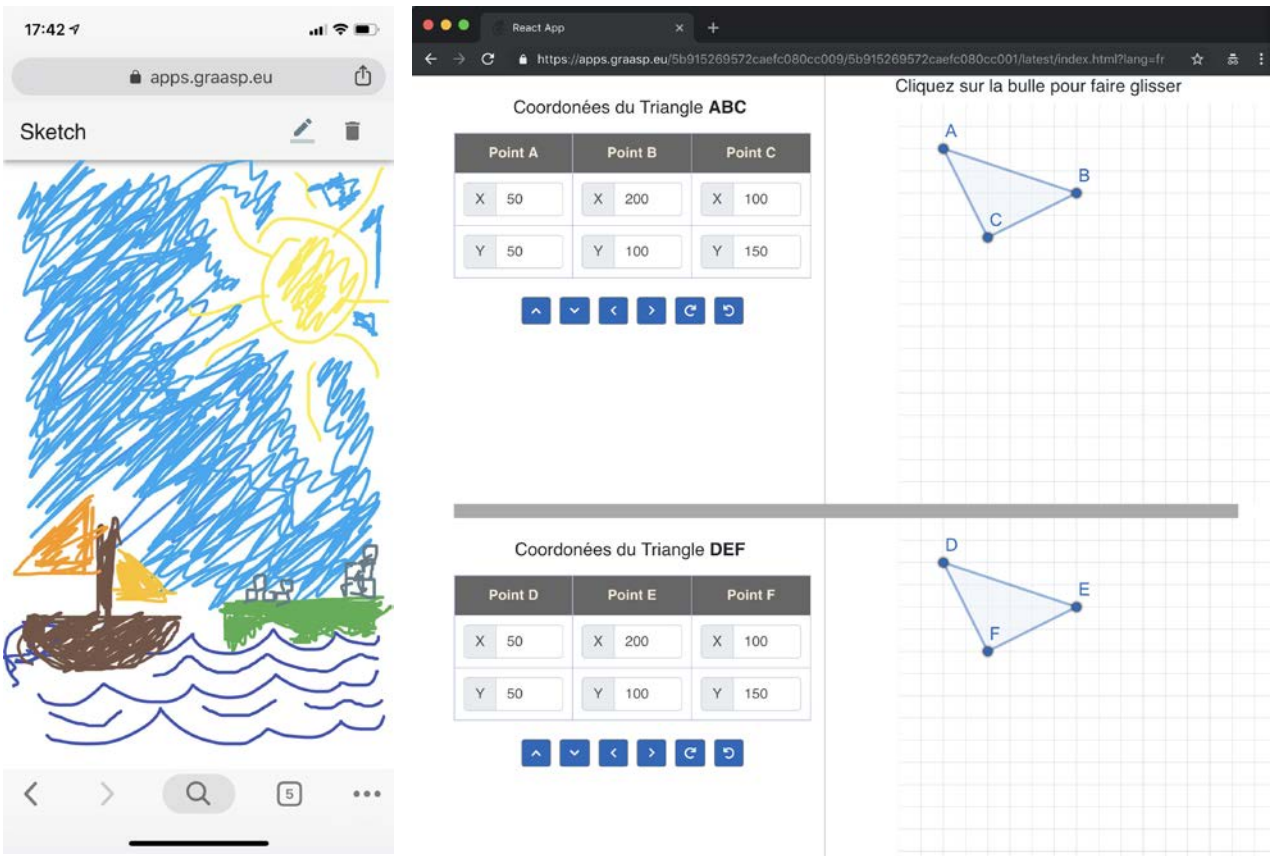


Figure 2: Left, the new mobile-friendly Graasp Sketch App is offline-ready, taking advantage of functionality included in the starter kit to bundle all of its resources into one file. Right, a new lab on similar triangles that uses the starter kit's built-in internationalization.

4.4. Dedicated Proxy

In addition to the cloud repositories for new apps and labs, we have deployed on AWS Cloudfront a dedicated reverse proxy and CDN for apps and labs hosted by the University of Twente, currently the main provider of scaffolding and learning analytics apps. The advantages of this Cloudfront service are twofold. Firstly, it allows using the domain utwente.graasp.eu as a proxy for the servers at Twente. This means that the current and future apps and labs hosted at the University of Twente can exploit the new Graasp API without any change in their core infrastructure. Secondly, since Cloudfront caches the content of applications across data centers worldwide, the time to fetch the content is reduced for clients located far from the host servers.

4.5 FROG Activities

Furthermore, the Graasp ecosystem now features an instance of FROG (Fabricating and Running Orchestration Graphs) running on our AWS infrastructure. FROG is a platform developed at the Computer-Human Interaction in Learning and Instruction (CHILI) lab at EPFL. We have worked closely with the team at CHILI to integrate their more than 30 learning activities, which are standalone applications including chat, a PDF annotation tool, a code editor, amongst others. FROG activities are collaborative, which allows for all of the students working in a learning space to interact with each other in real-time. FROG also allows us to give out-of-the-box support for H5P applications. Led by a team in Norway, H5P is an open-source initiative that empowers the community to build HTML5 applications for learning. These new integrations complement our current app repository and are currently being tested before being fully integrated into Graasp. Nonetheless, some of them are already available on the Graasp platform through a keyboard shortcut.

4.6 Developer Portal

One of the key motivations behind the implementation of the aforementioned features, frameworks and infrastructures is to empower current app and lab developers, lower the barrier to entry for those interested in contributing learning resources to the ecosystem, and encourage best practices and collaboration. To this purpose, we deployed a new developer portal (developers.graasp.eu), featuring documentation on how to get started with app and lab development, how to use the Graasp API, how to install the starter kit, support different views, internationalization, etc. This portal is open source and its repository is hosted on GitHub, meaning anyone can contribute, ask questions, discuss, and raise any issues they might encounter. This is a path towards a sustainable and vibrant developer community offering content for the Go-Lab ecosystem beyond the life of the Next-Lab and the Go-Ga H2020 innovation actions.

5 HTTPs enforcement

HTTPS is the standard secure version of HTTP: internally it uses the same HTTP protocol but all the information is encrypted so, if properly implemented, no actor between the final web server and

the final user can read (or modify) the information transmitted. This provides not only a higher level of security but also a higher level of privacy for the final user. While the protocol has been used for over two decades, until the last decade it was only broadly used in systems involving payments or where high level of privacy was required. For example, Google Mail did not start using https by default until 2010 and Facebook did not use https by default until 2013. Additionally, in 2014, Let's Encrypt was created with the support of different Internet organizations such as Akamai Technologies, Cisco Systems, Mozilla Foundation or the Electronic Frontier Foundation, with the goal of creating a more secure Internet, and since 2016 Let's Encrypt provides free https certificates in an automated fashion. Additionally, the way websites are displayed in web browsers have changed: in early 2017 a website with https would have a "Secure" message next to the URL; while in 2017 the "Not secure" message started to appear in pages where data was added; and at the time of this writing Google Chrome shows already "Not secure" to any website not using https.

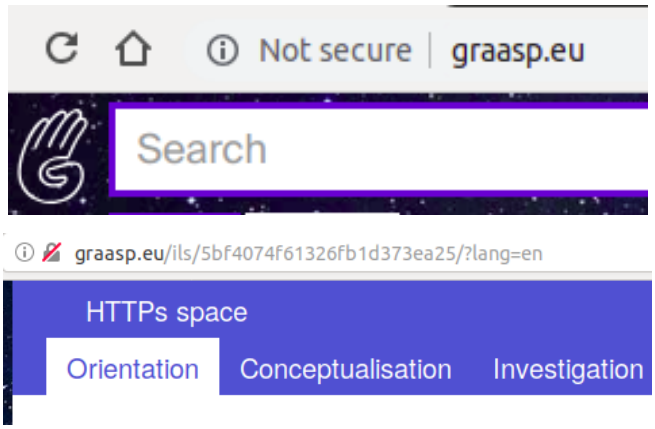


Figure 3: “Not secure” warning messages in Google Chrome (top; shown always) and Firefox (bottom; shown when the student can write any text, such as his/her login in the ILS)

In this context, from a technical perspective leading Internet technology providers have been enforcing the use of https in different ways that have affected Go-Lab and Next-Lab. Leading web browsers started in 2014 and 2015 to provide warnings if a website was using https but its internal contents (scripts, images) were http; what is called "mixed content". The rationale is that the user should be aware that even if the website being visited uses https, there is still information that can be read by third parties and therefore it is not secure. Short time after, web browsers started rejecting these requests by default, so nowadays it is not possible to have an https page which internally includes an http resource.

This is the main challenge in the Next-Lab https enforcement task. In Next-Lab there are 575 laboratories and 40 apps at the time of this writing. Some labs have more than one URL. And in total, 216 URLs in 67 different domains are not supporting https at the time of this writing. These numbers are probably slightly higher, since there might be websites using http internally that we cannot automatically detect. Given that these contents are displayed inside Graasp, if Graasp was using https by default, none of these labs and apps would appear.

For these reason, there are three alternatives that can be implement to circumvent this problem:

1. **Displaying a placeholder** with a message where the laboratory should appear, only for those laboratories using http-only. In the message it will be indicated that those contents are not available using https, but they can be open in a different window / pop-up. This breaks the usability cycle (students would not see the laboratory inside the ILS but outside) but it is the only completely safe (it always works) and automated (for any laboratory) solution.
2. **Implementing a https proxy**: It is technically possible to provide an https system, where users go to a https website (the proxy itself) and the proxy acts a man-in-the-middle requesting data to the original website. While this solution is automated (it can be used for any laboratory), and where it works, it is usable (students do not notice that the website is going through another system). It is not a safe solution: it may make the laboratory not work. For example, if the laboratory has an absolute URL inside one of its scripts (which unfortunately is common) using http, it might be impossible to guarantee that this call is going to work. Furthermore, there are labs that have conditions based on the domain and given that the domain where they are executed is different, they also fail. And it is not possible to have a solution that safely checks if the website is working under the proxy or not: for example one approach implemented was to compare screenshots of the website with and without proxy; but many laboratories have an random initial status that disables this feature. And even in the cases where it works, there are situations that cannot be automatically checked: maybe it seems to work but then there is a button that makes a special request which does not work in the proxy. There are more technical details in section 7 (Improved gateway).
3. **Contacting the 67 domain owners** to request them to use https. This is safe (if they do, it will work) and usable (users would see the laboratory as they see it now) but not automated (they must manually apply changes in their settings). Theoretically, while nowadays there are automated tools that make the process smooth and free of charge, some owners might face chain issues whenever they rely on http-only resources such as external resources or network cameras (without proxying them). We have not identified a case where a solution cannot be put in place and given that this process is manual, we will be able to address it case by case if it happens.

These alternatives and their limits are summarized in the table below.

Solution	Automatic	Safe	Usable: no action from student / teacher
Displaying placeholder	Yes	Yes	No
HTTPS proxy	Yes	No	Yes
Contacting owners	No	Yes	Yes

The approach taken in Next-Lab is the following:

1. As a first step, we are contacting the different app and lab providers to encourage them to support https. For the top providers (measured by number of apps or labs and by number of

uses), the messages include customized instructions for using Let's Encrypt in the particular web server and operating system they use.
2. In two months, for those top apps and labs not migrated to https by owners, the proxy solution will be evaluated in each of them manually. For those where the proxy works with all functionalities, the proxy solution will be used.
3. **In March 1st, 2019**, Next-Lab will only use https; relying on the placeholder message for those which have not been migrated and that could not be managed through the https proxy. New labs will by default put it the placeholder message but members of the project will be able to activate the proxy if it works.

6 Improved language support

The App Composer is the component dedicated to enabling teachers to provide translations of the different apps and labs in their languages. In most apps - those natively supporting certain APIs - it works automatically: once a teacher translates an app, the translation is automatically available for all students and teachers using it in that app. Furthermore, it automatically synchronizes with the other components of the ecosystem, reporting that this app is now also available in this language.

However, in Next-Lab there were two challenges to this system:
1. The removal of Shindig, and therefore, the underlying OpenSocial standard that was used by the App Composer to manage internationalization.
2. The ongoing trend of supporting offline apps and labs: an offline app / lab will not be able to retrieve the translations from the App Composer in an automated way.

These two challenges require a new solution that covers both the advantages of a fully automated system and the advantages of an offline system. This section covers this new solution and other changes involving improved language support.

6.1 New internationalization API

As mentioned above, a new internationalization API was required, which supports the following features:
1. Support for online retrieval of App Composer translations: the app can connect to the App Composer and obtain the latest translations of the terms.
2. Support for offline mode: if the app selects this option, it will only use those files indexed in the document. The App Composer provides all the translations of all the URL in different formats; so app developers can download (http://composer.golabz.eu/translator/dev/apps/) their translations and serve them in their own apps when using the offline mode. An automated script will be implemented in the New labs & apps development framework to automatically retrieve these translations.
3. Independence from Apache Shindig: the new API relies exclusively in pure HTML and JavaScript, providing the data using standard HTML tags. It supports both terms being translated as tags or calls being performed by the app developer to the API. In the case of tags being translated, a MutationObserver is used to check new contents automatically and translate them as they are created.
4. Multiple messages files, using two formats: a new JSON format and the legacy OpenSocial XML file.

5. A JavaScript library for apps that use jQuery. Another library for Angular apps is planned. The link is: https://composer.golabz.eu/static/js/translations-jquery-1.0.js

The new API has been designed to be used not only by Next-Lab developers but also by external lab providers not supporting an internationalization system yet.

6.2 New statistics information

Given that the App Composer processes all the repository every few hours looking for changes in translations, it also now checks further information such as:

1. What labs and apps are not supported by https?
2. What labs are still using Adobe Flash and will become a problem when Adobe Flash is not supported anymore?
3. What labs have been failing for several days?

This information is available in the App Composer statistics page:

- https://composer.golabz.eu/translator/stats/

In particular, these three pages are updated every three hours containing such information:

- https://composer.golabz.eu/translator/stats/failing/ (failing labs and apps)
- https://composer.golabz.eu/translator/stats/ssl/ (labs and apps not supporting https)
- https://composer.golabz.eu/translator/stats/flash/ (labs and apps using Adobe Flash)

6.3 New supported languages

In Go-Lab, there was only support for languages and not subsets, dialects or different transcriptions of a language. This was found to be an important limitation in Next-Lab: for example, the Chinese language was supported but only using the Simplified Chinese (zh_CN) characters, and not the Traditional Chinese (zh_TW) which is in use in Taiwan nowadays. For this reason, all the components (Graasp, Gateway, Repository and App Composer) had to migrate to support also dialects. This opens also new possibilities of supporting other character sets (such as Serbian Cyrillic and Serbian Latin), and dialects of other languages, such as Peruvian Spanish or Brazilian Portuguese.

The App Composer has also increased the support for automatic translations, which are used to enable translators to translate faster by very often just confirming that an automatic translation is correct. In this sense, both Microsoft and Google have changed their existing APIs and also support for Deepl has been added. The current state of suggestions by language is available in the App Composer statistics page:

- https://composer.golabz.eu/translator/stats/suggestions

Finally, the App Composer also supports now Swahili, Vietnamese and Macedonian Slavic, reaching 49 languages supported by the App Composer.

7 Improved Gateway

The Gateway (Smart Gateway in Go-Lab) is a tool focused on providing access to online labs (virtual labs and remote labs) to the Go-Lab ecosystem. It relies on:

1. A set of plug-ins that support external lab repositories (such as PhET or Concord). In each system, the plug-in usually does:
    a. Automatic periodic retrieval of the languages supported by each laboratory. If a laboratory in PhET supports a new language (e.g., Estonian), the Gateway in few hours will detect this change and report it to the Go-Lab portal. This way, the Go-Lab portal will list this lab to users searching for certain terms and filtering by laboratories in Estonian.
    b. Automatically support all the laboratories in a particular repository. If Concord adds a new laboratory and later on the laboratory seems interesting for the community, that laboratory can be added. By then, given that the process is automatic, the laboratory is natively supported.
    c. Bypass authentication mechanisms in case it is needed. Most simulations do not need it; but certain remote laboratories require some form of username and password to access. Relying on the Gateway, teachers or students do not need to provide such credential.
2. A set of manually-added entries, where a non-technical person can add a simulation. If the language supports multiple languages, the tool allows scaling the laboratory and cropping it.

For each lab added (through the plug-in system or manually), the Gateway adapts it to the requirements of Graasp so it can be properly displayed. At the time of this writing, this is done using the OpenSocial standard and the Go-Lab libraries for reporting usage of each laboratory.

The remainder of this section are the specific new major features developed during the project in this component; not entering in minor details such as maintenance, its migration from a private server (at University of Deusto) to a cloud provider (OVH) or regular security upgrades.

7.1 New lab repositories

Different external lab repositories have been added using the plug-in system of the Gateway. These new repositories are:

Name	URL	# labs in golabz / # labs in provider
The Physics Aviary	https://github.com/gateway4labs/rlms_thephysicsaviary	82 / 130
Amrita Olabs	https://github.com/gateway4labs/rlms_amrita	26 / 124
Virtual Biology Lab	https://github.com/gateway4labs/rlms_virtualbiologylab	24 / 25
Vascak.cz	https://github.com/gateway4labs/rlms_vascak	11 / 206
ChemCollective	https://github.com/gateway4labs/rlms_chemcollective	2 / 38

7.2 Proxy (https)

As explained in the previous section *HTTPs enforcement*, one of the goals of the project is to support https in the whole ecosystem; but this is a problem with the Smart Gateway since it includes many resources which are only available in http (and not https). As noted, one of the efforts consists in providing an https proxy server, so automatically certain http-pages can be available under the https protocol.

To this end, the Gateway started supporting a https proxy. Essentially, any page using this schema is served using https:

- https://gateway.golabz.eu/proxy/<url>

Example:

- https://gateway.golabz.eu/proxy/http://www.walter-fendt.de/html5/phen/acceleration_en.htm

For the URL to work, it must be available in the Go-Lab repository for security purposes. For example, if a user attempts to use:

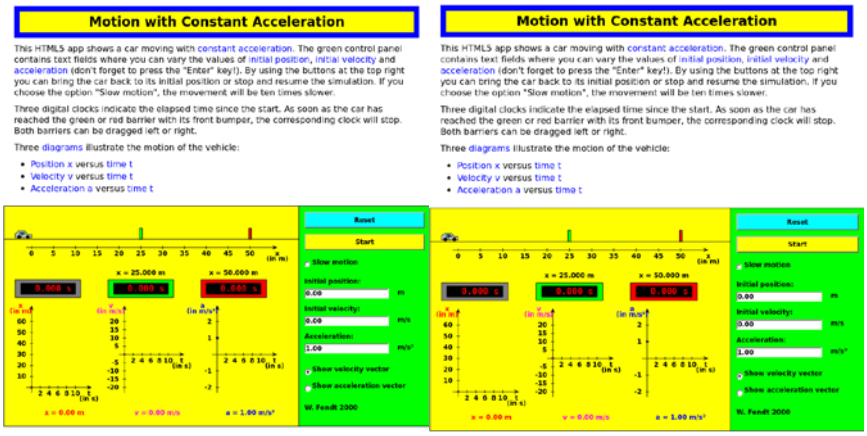
- https://gateway.golabz.eu/proxy/https://www.google.com

The proxy will disallow its usage; since that domain. This is important so the Gateway server does not become an anonymizing server for purposes unrelated to the project.

As mentioned in the *HTTPs enforcement* section, the proxy provides an automatic and usable solution; meaning that, when it works, the user does not need to do anything to get the contents served under https; and for a non-technical person, checking if the proxy works for a laboratory requires no development effort on each particular lab. However, the proxy server is not a safe solution since in some cases it works, in other cases it does not, and it needs a human effort to validate if it does work.

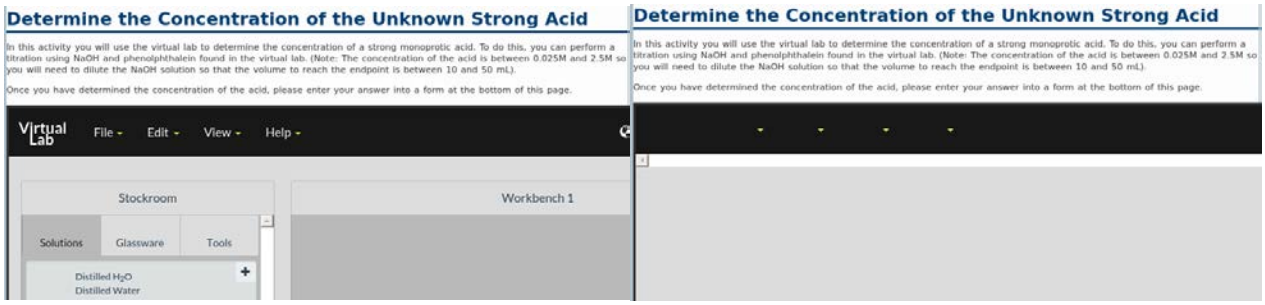
During the development of the project, different options to try to maximize this effort were evaluated. One of these options was to automatically take screenshots of the pages with and without proxy. The rationale was that in those labs where the screenshot is the same, the image would be different, and in those labs where the screenshot was different, the proxy was not working.

As a first example, these are two screenshots of the same laboratory, using proxy and not using proxy:



Original URL	http://www.walter-fendt.de/html5/phen/acceleration_en.htm
Proxy URL	https://gateway.golabz.eu/proxy/http://www.walter-fendt.de/html5/phen/acceleration_en.htm
Screenshot with proxy	http://composer.golabz.eu/static/proxy-images/dc8a31d591fd80da23fb85fc0c26a43d_proxy.png
Screenshot without proxy	http://composer.golabz.eu/static/proxy-images/dc8a31d591fd80da23fb85fc0c26a43d_non-proxy.png

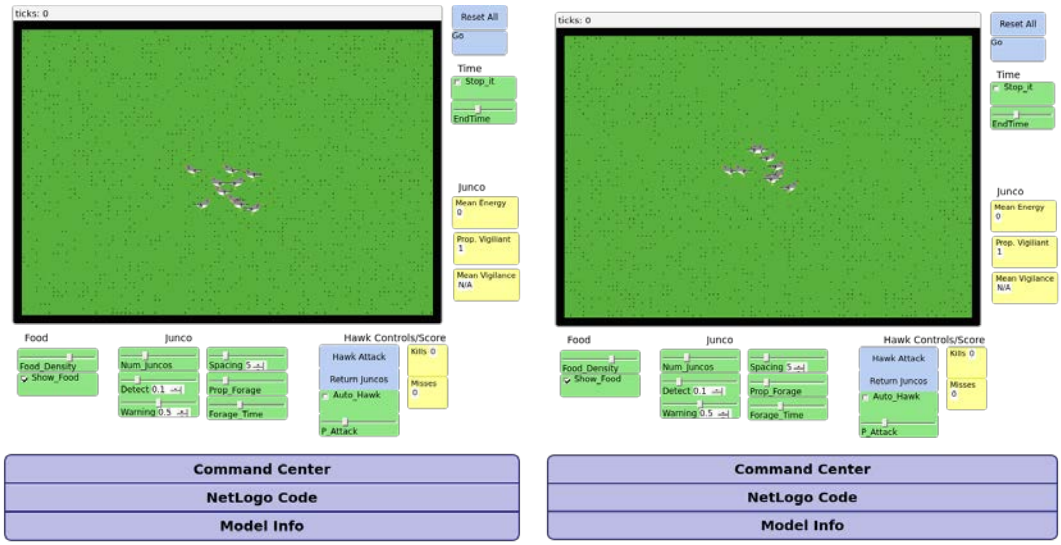
In this case, both screenshots are the same. However, this does not guarantee that the laboratory is indeed working properly. It may happen that when clicking on the Start button a script is called which does not work. Here is another example:



Original URL	http://chemcollective.org/activities/autograded/124
Proxy URL	https://gateway.golabz.eu/proxy/http://chemcollective.org/activities/autograded/124

Screenshot with proxy	http://composer.golabz.eu/static/proxy-images/3f46ee976918448ae49485a1e6cac1aa_proxy.png
Screenshot without proxy	http://composer.golabz.eu/static/proxy-images/3f46ee976918448ae49485a1e6cac1aa_non-proxy.png

In this case, the screenshot clearly differs because some scripts are not loaded properly because of the usage of https and http. However, that might not be the reason for the screenshot being different. For instance, in the last example the images differ since the animals are in different locations because the start situation is random. This happens more often that it was originally expected, especially with simulations including GIF files that, depending on the millisecond that the screenshot is taken, the result differs.



Original URL	http://virtualbiologylab.org/NetWebHTML_FilesJan2016/VigilanceBehaviorCollectiveModel.html
Proxy URL	https://gateway.golabz.eu/proxy/http://virtualbiologylab.org/NetWebHTML_FilesJan2016/VigilanceBehaviorCollectiveModel.html
Screenshot with proxy	http://composer.golabz.eu/static/proxy-images/4daa810097090084203ecf5ef37a64e5_proxy.png
Screenshot without proxy	http://composer.golabz.eu/static/proxy-images/4daa810097090084203ecf5ef37a64e5_non-proxy.png

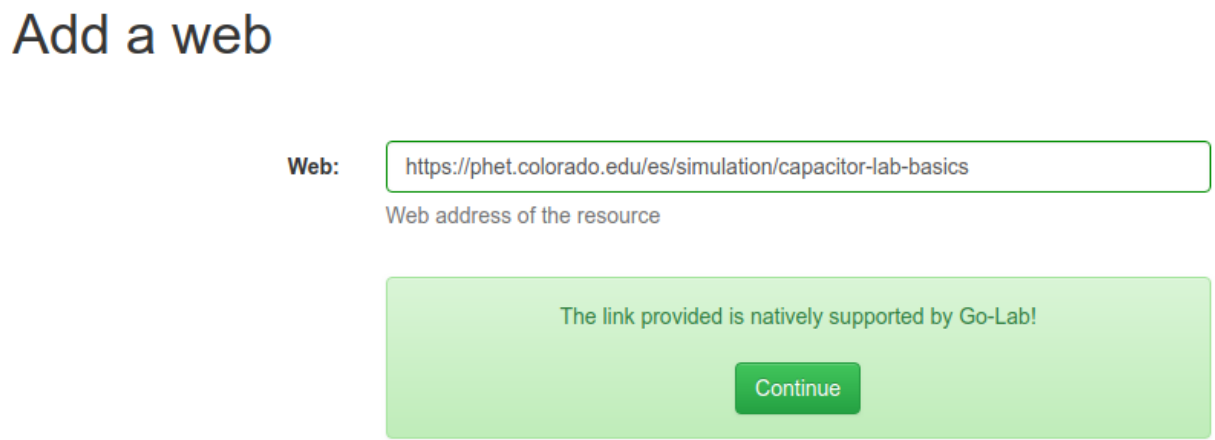
Given that it cannot be automatically guaranteed or discarded, this resource-consuming process of taking automatic screenshots is not performed anymore; and the approach, as described in the

previous section, is based on a manual work of checking what laboratories can be used through the proxy.

7.3 Embedder

In Go-Lab, the Smart Gateway had a plug-in based system to add external laboratories and export them as OpenSocial in an automatic fashion (checking automatically external systems such as PhET or QuVis for all their labs, languages, etc.). On the other hand, the App Composer counted with a component (the Embedder), which allowed non-technical users to add general external resources, crop them, scale them and include them in a way that the App Composer notified about its usage to the rest of the infrastructure.

In Next-Lab, the Embedder from the App Composer has been migrated into the Gateway. This allows a number of new features: the Gateway is now in charge of the integration of most laboratories, and both databases are crossed. The plug-in system now detects automatically that, given a link (e.g., https://phet.colorado.edu/en/simulation/capacitor-lab-basics) it checks if it can be managed by an existing plug-in; and if that is the case, it should be managed by the plug-in (so if PhET adds a new language in the future, the lab is automatically available in that language. The Embedder uses this feature so when non-technical people add a lab, they can configure it, but if it is already natively supported by a plug-in, the plug-in is used. This makes the process easier given that there is a single point of adding laboratories.



Additionally, the Embedder has new features such as automatically detecting if a resource can't be embedded due to certain headers (same-origin policies), check if it supports https or change if the document is Flash.

7.4 Usage

At the time of this writing, the repository counts with 575 labs which have 591 resources (some labs have more than one resource, such as two views of a laboratory). 90% of these resources are managed through the Gateway.

8 Conclusion and outlook

Thanks to the updated infrastructure described in this deliverable, the golabz ecosystem is now ready for a sustainable exploitation. Labs and apps can now be progressively converted to rely on the new Graasp API and be hosted when necessary in the new repository for offline exploitation.

The gateway also provides the necessary scheme to progressively migrate all labs to https to comply with the generalized security scheme of all modern browsers.

Thanks to the agile and redundant development and deployment schemes implemented, we can enforce a smooth transition from the previous to the new infrastructure in a way mostly transparent to the end users.