A SUMMARY PROTOCOL ON HOW TO DEVELOP A REMOTE LABORATORY

Luis de la Torre[1]*,
Milan Matijević[2],
Đorđe Seničić[3],
Maja Milojević[4],
Marko Tanasković[5]

[1]UNED, Calle Juan del Rosal, Madrid, Spain

[2]Faculty of Engineering at University of Kragujevac, Kragujevac, Serbia

[3]Cognipix D.O.O. Belgrade, Belgrade, Serbia

[4]University of Belgrade – Faculty of Physical Chemistry, Belgrade, Serbia

[5]Singidunum University, Belgrade, Serbia

Abstract:

Web Laboratories (weblabs) have proven to be a fantastic tool for teaching and learning, especially, as a complement to traditional hands-on labs in technical, engineering and science fields. Unfortunately, designing and implementing a weblab is usually not an easy nor a quick task, making this kind of resources difficult to find in many university courses. Therefore, an effort to educate our educators on how to create weblabs, so they can add them to their courses, is required. The Erasmus+ KA RELAB project is aware of this lack, and so, one of its main objectives is to address this problem. This paper is a result of the work done within the RELAB project. Here, we provide a summary protocol that makes it easier for teachers wanting to develop their weblabs to start doing so.

Keywords:

Remote labs, lab work, online education, engineering education.

INTRODUCTION

Experimentation plays an essential role in engineering and scientific education. Whereas traditional hands-on labs offer students opportunities for experimentation with real systems (they provide "actual experience"), they involve high costs associated with equipment, space, and staff for maintenance [1]. To minimize those costs, remote labs, or weblabs, use real setups which can be used at a distance (see Figure 1). In addition to reducing costs, weblabs the following benefits [2]:

1. Availability: VRLs can be used from anywhere at anytime, thus they support students geographically scattered, who besides are conditioned to different time zones.
2. Accessibility for handicapped people.
3. Observability: lab sessions can be watched by many people or even recorded.
4. Safety: VRLs can be a better alternative to hands-on labs for dangerous experimentation.

Correspondence:
Luis de la Torre

e-mail:
ldelatorre@dia.uned.es

DOI: 10.15308/Sinteza-2022-163-168

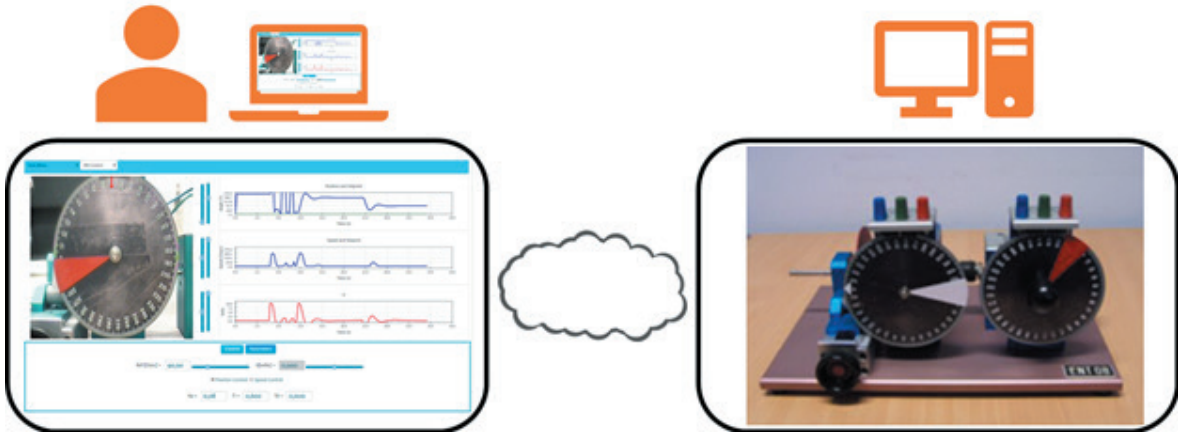

Figure 1 – Basic architecture of a weblab.

Despite their advantages, few teachers and institutions actually use weblabs as a common way to complement their traditional hands-on activities [3]. One of the main reasons for this is that weblabs have been traditionally hard and time consuming to develop. Moreover, they require a mix of skills (programming, communications, hardware setup, etc.) which makes it difficult for single persons or small teams to cover.

Therefore, this paper presents a comprehensive protocol with the steps to follow and the tools to use when facing the challenge of developing a weblab for the first time. This does not pretend to be a full step-by-step guide, nor to cover all the possible architectures, solutions and tools that can be used and applied for the task. For example, desktop-type access for weblabs (which have their own difficulties and risks) is not considered in this work. Instead, it just pretends to give a reasonable starting point for those wanting to develop and deploy their first weblab.

2. SUMMARY PROTOCOL

We can divide the process of designing, developing and using weblabs into four main considerations [4]: 1) security and communication issues, 2) the server-side development, implementation and/or integration, 2) the client-side development and implementation, and 4) the deployment of the weblab app into an online course. The next subsections tackle each of them.

2.1. SECURITY & COMMUNICATIONS

Usually neglected, security and communications issues with weblabs are extremely important, and, sometimes, difficult to address. It is also the first thing that should be considered when developing a weblab, as the decisions made in this phase will likely affect how to develop the other ones.

The following are the most common sources of threats related to webLabs, and the risks associated with them, followed by the basic recommendations to avoid them:

- *Risk*: Equipment (cameras, computers, switches, etc.) use public IPs. When the lab devices use public IPs, they are visible and reachable to anyone. While this is a very handful solution for exposing the lab services and making them accessible, it also carries a huge risk, as any user or bot can try to access such devices. *Recommendation*: Use private IPs within a VLAN/VPN that is protected and allows you to control who may access which resources within the private network.
- *Risk*: WebLab services do not use an authentication system. Sometimes, weblabs are left open, so anyone can access them. This may be done either by design or by mistake, but in both cases, malicious users may take control of the lab and make bad use of it, potentially harming the equipment, monopolizing it and preventing others from using it. Or simply, access to cameras that should not be available to everyone at any time. *Recommendation*: Always use an authentication system at the communications protocol level. Using authentication for accessing a webpage is not

enough, as users may be able to access the webpage at some point, load a web app and keep it open and running if no security is implemented at the communications protocol level. Users may even be able to download the app and keep using it whenever they want, without having to log in and authenticate in the webpage never again.

- *Risk*: Services not running over HTTPS. Even when an authentication mechanism is in place, if such mechanism does not rely on HTTPS, security is at risk due to the use of unencrypted transport of the credentials. *Recommendation*: Always install an SSL certificate to enable the use of HTTPS and encrypted communications.

2.2. SERVER-SIDE DEVELOPMENT, IMPLEMENTATION AND/OR INTEGRATION

The server-side development for a weblab usually has to solve connecting a computer (whether a desktop computer or a single board computer) with real hardware, in the form of actuators, sensors, etc. This can be done through various hardware (Arduino, Raspberry, Data Acquisition Cards, PLCs, etc) and software (LabVIEW, Beckhoff, Matlab, Python, etc.) solutions. This work does not address the hardware part, as this is extremely dependent on the type of experiment to be prepared and the resources that are available or at reach. For the software part, we focus on two well-known solutions (LabVIEW and Python), both equally valid and easy to connect to a web-app in HTML5.

The use of LabVIEW is recommended for those who have a license and can use the software, as there are good chances the lab hardware that needs to be connected for the weblab already has a LabVIEW interface and/or API. Python is recommended for any other situation, as it presents a high degree of interoperability and connectivity, making it ideal for most applications. Two common architecture situations and lab setups supported by these two implementations of RIP are illustrated in Figure 2. Both scenarios can be addressed using a middleware that enables the communication between a client application (see next section) and any software in the lab operating the experimental setup. This middleware is called RIP [5] and can be downloaded for free in both of its implementations: LabVIEW (https://github.com/UNEDLabs/rip-labview-server) and Python (https://github.com/UNEDLabs/rip-python-server). While the installation and configuration to make them work with each particular lab setup and the software used to operate it, differ for these two, instructions can be found when downloading RIP. In any case, once the installation and configuration is completed, both versions would start exposing your lab inputs and outputs as web services, using the exact same protocol. Listing 1 presents a generic JSON example response, provided by RIP, to an HTTP request.

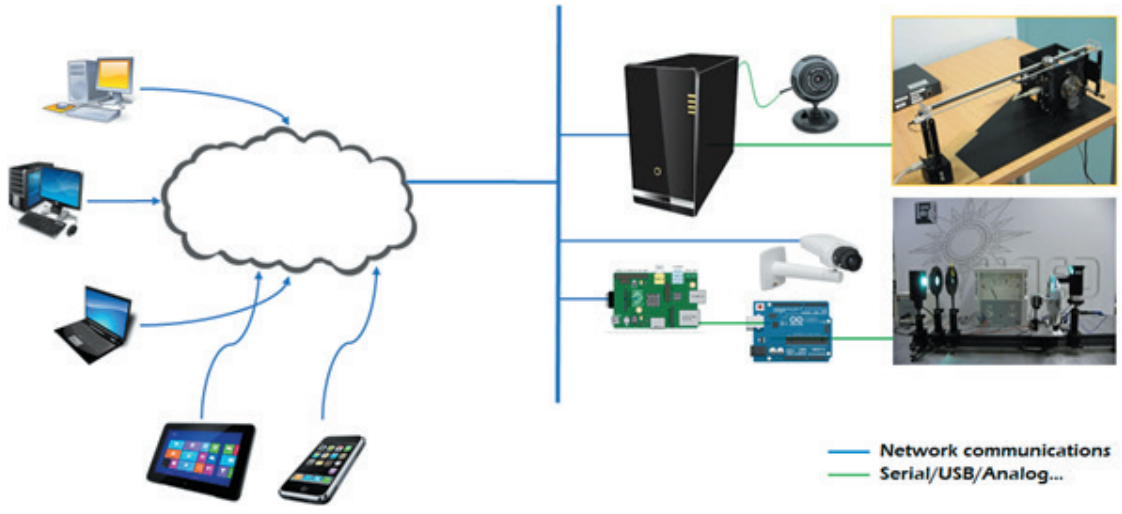

Figure 2 – Two typical weblab implementations supported by RIP.

```
{
  "outputs": {
    "list": [
      {
        "name": "output_1",
        "description": "description_output_1",
        "type": "type_output_1",
        "min": "min_output_1",
        "max": "max_output_1",
        "precision": "precision_output_1"
      },
      // ... ,
      {
        "name": "output_N",
        "description": "description_output_N",
        "type": "type_output_N",
        "min": "min_output_N",
        "max": "max_output_N",
        "precision": "precision_output_N"
      }
    ],
    "methods": [ { //... } ]
  },
  "inputs": {
    "list": [
      {
        "name": "input_1",
        "description": "description_input_1",
        "type": "type_input_1",
        "min": "min_input_1",
        "max": "max_input_1",
        "precision": "precision_input_1"
      },
      //... ,
      {
        "name": "input_M",
        "description": "description_input_M",
        "type": "type_input_M",
        "min": "min_input_M",
        "max": "max_input_M",
         "precision": "precision_input_M"
       }
    ],
    "methods": [ { //... } ]
  }
}
```

Listing 1 - An example of RIP response to an HTTP request.

For the sake of simplicity and briefness, the methods in the outputs and inputs fields are not included. However, they contain all the information a client needs to create REST calls to: 1) subscribe for getting updates on the values of the outputs, and 2) send requests for writing/modifying the inputs. Interested readers can find all the details in [5]. This information, and the available methods, published as webservices by RIP, to write a new value in an input variable and read the value of an output variable, is all a client app needs to actually operate the lab from the web and thus, transform the traditional lab into a weblab.

2.3. CLIENT-SIDE DEVELOPMENT & IMPLEMENTATION

The client-side development tackles with creating the HTML5 web application that would allow students to interact with the remote lab assets. This usually includes buttons, sliders, graphs, input fields and/or a video stream of what is going on in the lab room.

Any HTML + Javascript + CSS programming could give a weblab HTML5 app as a result, but the amount of work to program this from scratch is considerable. Other solutions, such as using Unity or similar web-friendly engines, are also possible, but the same problem remains. Here, we propose the use of EjsS [6] for this task, as it is a tool specially designed to make the task easy and, furthermore, anyone can download it and use it for free.

Web applications developed in EjsS can be either simulations or remote lab interfaces. When the app is a remote lab interface, the communication with the lab hardware/software can be easily implemented using a RIP client, which is already included in EjsS as an EjsS element. Readers needing a tutorial on how to use EjsS can find one at [7].

As a very brief introduction to EjsS elements, it is enough to say that these elements are added to an EjsS app by choosing the Model tab and then dragging one element icon from the palette (in the right part of the EjsS editor) into the model elements list (in the left part of the editor). Then, it is enough to double-click the new element to set its properties. Once created, users can call the element's methods in their app code to operate with the element. In the case of the RIP element, the three main methods to be used are:

- `connect()`: To connect the client web app with the RIP server and enable communication with the lab hardware. It is required to call this method and wait for its response before using the following two. It is important to note that RIP does not provide any authentication mechanism and therefore, this security aspect must be addressed somehow, usually at the VPN level (see section 2.1).
- `[value_i, value_k] = get([output_i, …, output_k])`: To obtain the values of the specified output variables.
- `set([input_l, …, input_m], [value_l, …, value_m])`: To write the specified new values in the specified input variable.

As simple as they are, these three methods provide all the necessary functionalities to communicate with the lab and thus, build a web app that is interactive and allows users to work with the lab equipment remotely. Once the web app is finished, EjsS provides an option to put all the files associated with this HTML5 app (.html, .js and .css files) inside a single .zip file and the only task that remains is to deploy it into an online course so that students can use it.

2.4. DEPLOYMENT

While the work required on the server side is a must in order to setup a weblab, it is the client web application the one that needs to get deployed in an online course to make it accessible to the students and enable them to perform their lab tasks.

As with all the previous tasks, this can be done in several ways. Here, we focus on using a Moodle plugin called EJSApp (which can be downloaded for free from [8]) to enable their easy integration into this Learning Management System (LMS). While this seems to limit its use solely to the case when the institution/teacher uses this LMS, this is not true. First of all, Moodle is open-source software that can be downloaded and used for free. Therefore, anyone could adopt it. Equally important, however, is that any activity embedded into a Moodle course (and this includes weblabs integrated through EJSApp) can be shared with other LMS through the Learning Tools Interoperability (LTI) standard [9], supported by all current major LMS. Therefore, once a weblab is embedded in a Moodle system, it can be shared with virtually any institutional platform and used from their online courses. The process for sharing a weblab from Moodle with another LMS follows the LTI standard for sharing any other learning tool, and instructions about how to do this can be found in the Moodle docs or in the target LMS platform documentation.

Once EJSApp is installed in Moodle (the instructions for this can be found in the plugin's download webpage or in the Moodle docs) embedding a web lab created with EjsS in Moodle becomes extremely easy. Figure 3 shows the web form to add or edit an activity with an EjsS app (EJSApp). The only two fields that need to be filled are the name of the activity and the .zip file that encapsulates the EjsS application.

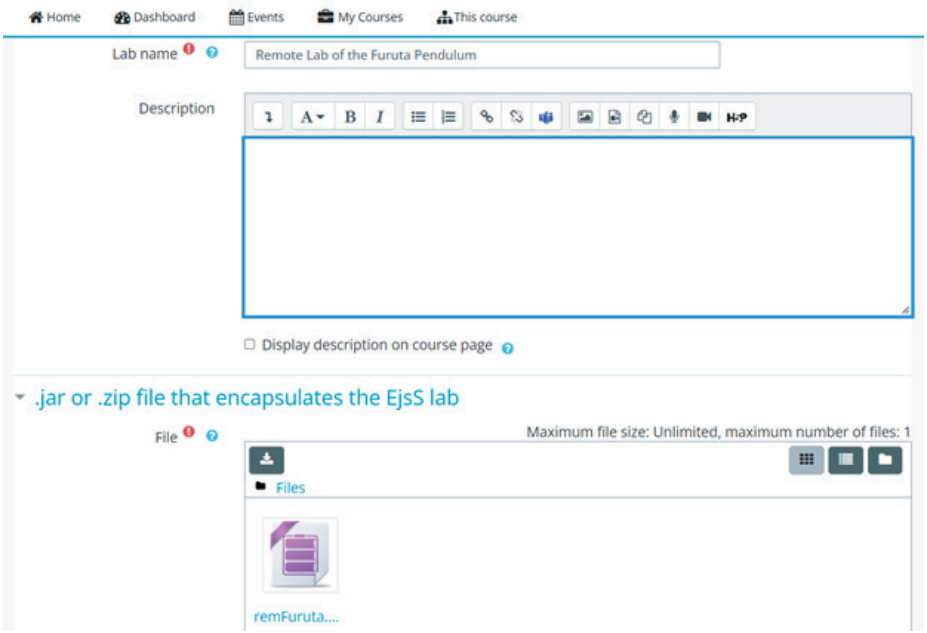

Figure 3 – Weblab deployment into Moodle through EJSApp.

3. CONCLUSION

This work summarizes the protocol and recommendations given in the RELAB's Intellectual Output 6. Following these guidelines, teachers can start getting familiarized with designing, creating and deploying a weblab, as well as with the software tools they can use to do so.

By making the learning curve more accessible and lowering the barriers for the entry point to develop weblabs, we expect to contribute to popularize this type of labs which have already demonstrated their usefulness and effectiveness in science and engineering education.

4. ACKNOWLEDGEMENTS

This work has been funded by the ERASMUS+ KA226 project 2020-1-RS01-KA226-HE-094550 "Repository of Open Educational Resources for Laboratory Support in Engineering and Natural Science – RELAB", which is gratefully acknowledged.

5. REFERENCES

[1] L. Bogosyan and S. Gomes, "Current trends in remote laboratories," *IEEE Trans. on Industrial Electronics*, vol. 56, no. 12, p. 4744–4756, 2009.

[2] C. Gravier, J. Fayolle, B. Bayard, M. Ates and J. Lardon, "State of the art about remote laboratories paradigms - foundations of ongoing mutations," *International Journal of Online Engineering*, vol. 4, 2008.

[3] R. Heradio, L. d. l. Torre and S. Dormido, "Virtual and remote labs in control education: A survey," *Annual Reviews in Control*, vol. 42, pp. 1-10, 2016.

[4] Erasmus+ KA 2 Project RELAB, I*ntellectual Output 6. Available online: https://relab.kg.ac.rs/publications/uned/weblab/*, 2022.

[5] L. d. l. Torre, J. Chacon and D. Chaos. [Online]. Available: https://github.com/UNEDLabs/rip-spec. [Accessed 31 03 2022].

[6] F. Esquembre, "Easy Javascript Simulations," [Online]. Available: https://www.um.es/fem/EjsWiki/. [Accessed 31 03 2022].

[7] L. d. l. Torre and R. Heradio, "EJSApp repository in GitHub," [Online]. Available: https://github.com/UNEDLabs/moodle-mod_ejsapp. [Accessed 31 03 2022].

[8] IMS Global, "Learning Tools Interoperability," [Online]. Available: https://www.imsglobal.org/activity/learning-tools-interoperability. [Accessed 31 03 2022].

[9] F. Esquembre, "Easy Javascript Simulations," [Online]. Available: https://www.um.es/fem/EjsWiki/uploads/Download/EjsS_Manual.pdf. [Accessed 03 31 2022].