18/12/2021

REIABL

IO 6: TUTORIAL OF TECHNICAL DESIGN AND IMPLEMENTATION – DEPLOYMENT

Co-funded by
the European Union

1. FEATURES OF EjsS WEB APPLICATIONS EMBEDDED IN THE MOODLE PLATFORM WITH EJSAPP

Web applications developed in EjsS (it is strongly recommended to always use the latest available version of EjsS, which you can obtain here: https://gitlab.com/EjsS/tool) that are embedded in the Moodle Learning Management System environment as EJSApp activities may acquire the following features and functionalities:

- Adding a new learning activity in a Moodle course based on web apps (EJSApp).
- Web apps multi-lingual support integration with Moodle.
- Saving files (.json, .blk, .rec, .txt, .jpg ...), from the web app to Moodle's private files repository and reading/loading them, from this repository, to the web app.
- Loading initial states for a web app (files in. json format).
- Recording and reproducing interactions with web apps (files in .rec format).
- Enabling the use of a visual programming language to interact and communicate with web apps.
- Configuring a web app to use it as a remote interface that connects to real hardware.

This manual describes how to obtain all the integration features listed above. Chapter 2 of this document explains how web applications should be prepared in EjsS to get the proper operation in Moodle, while Chapter 3 shows how to configure Moodle EJSApp activities. If you need help with Moodle activities in general at a user level, remember this platform is just an extension of Moodle, so a visit to Moodle's official documentation really helps: https://docs.moodle.org/en/Activities

2. WEB APP PREPARATION IN EjsS FOR THEIR USE IN MOODLE

First, it is worth mentioning that this guide is limited to the preparation of EjsS applications in JavaScript mode, since the Java version is in disuse. Therefore, the first thing is to check that EjsS has been started in this mode and, if not, to do so. Figure 1 shows the *Basic* tab in EjsS' console window, from which the JavaScript programming language can be selected.

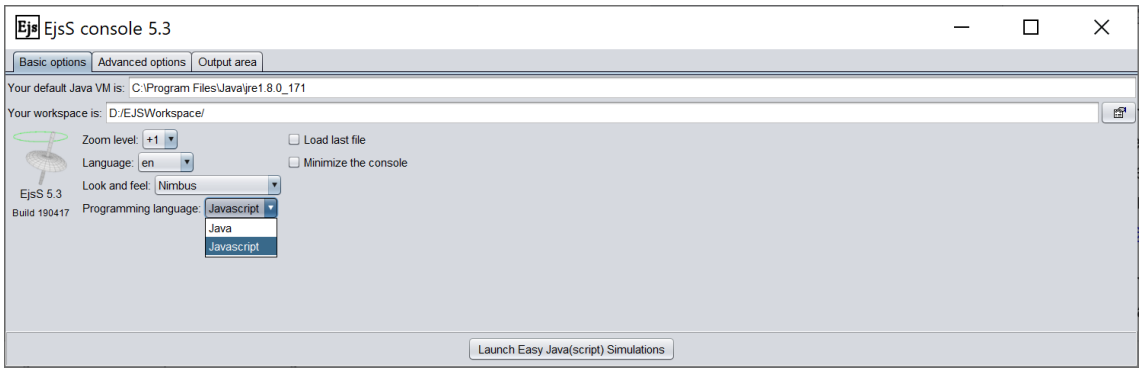

Figure 1. Setting JavaScript as default mode/programming language to work with EjsS

If you need a tutorial for EjsS, a manual to start learning about how to use this program can be found here: https://www.um.es/fem/EjsWiki/uploads/Download/EjsS_Manual.pdf While Chapter 2 is dedicated to the Java mode of EjsS and Chapter 4 focuses on converting Java apps to JavaScript apps, you can safely skip these two chapters and read the rest of the document.

2.1. Multi-language support

Language setting in EjsS web applications integrated into Moodle with EJSApp is automatically linked to the selected language in a Moodle web environment when the web app has been prepared using the native multi-language support offered by EjsS. To prepare an application in EjsS with this support for multiple languages, just follow the instructions given in this video-tutorial: https://www.um.es/fem/Ejs/WebCasts/en/EJSTranslationFacilities/media/EJSTranslationFacilities.mp4

Important note: It is a requirement that the default language of the web app (that is, the language of the texts that are written in the elements of the EjsS View) is English. Then, additional translations can be added. The opposite (making, for example, Spanish the default language and adding English translation later) makes the latter not to be displayed in Moodle.

2.2. Saving and loading files

It is important to follow some design guidelines of the user interfaces that are similar for all applications. The file saving/reading menu has been traditionally placed at the top of applications, as shown in Figure 2.

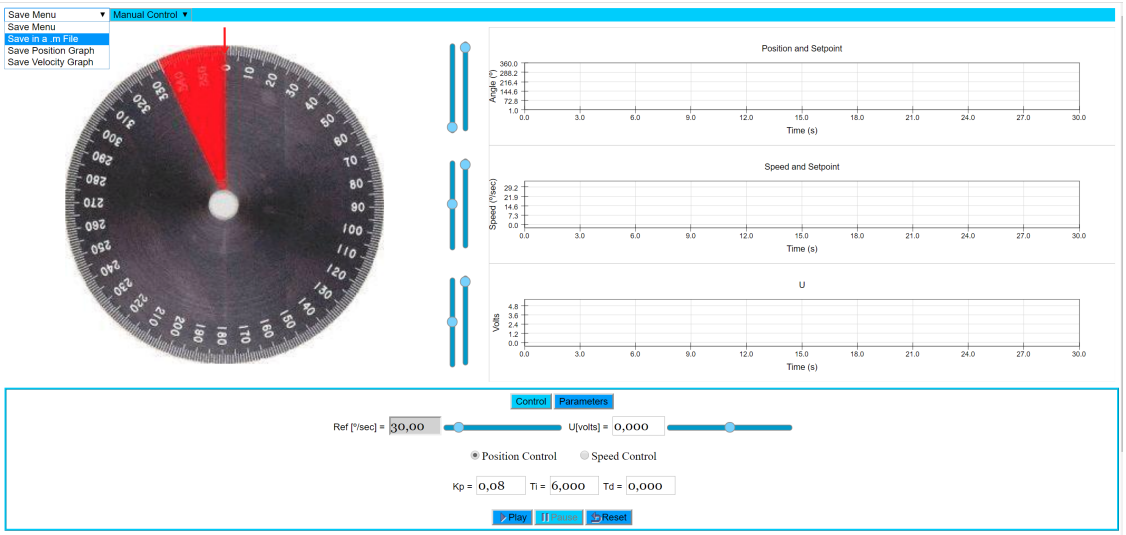

Figure 2. Menu and buttons to save/load files

Table 1 offers a list of EjsS native methods to save and load different type of files. These methods can be used in the actions invoked when the top menu options are selected, to save and load files.

Table 1. Methods for saving and loading files

There are four methods that allow sending a file from the EjsS web app to Moodle's private files repository:

- Method *_saveText(String FileName, String FileType, String FileContent)* , to save a text file that will automatically fix, according to *FileType*, its extension.
- Method *_saveImage(String FileName, String ViewElementName)*, to save a screenshot image of the wep app's user interface (determined by the View element name in EjsS).
- Method *_saveState(String FileName)*, to save into a .json file the complete list of variables (and their values) that define the state of a web app simulations.
- Method *_saveVariables(String FileName, List<String> vars)*, to save only the indicated variables (*vars* parameter) and their values in a .json file.

And there are three ways to recover the data stored in files from an EjsS web application:

- Method *_readText(String FileName, String FileType, String VariableName)*, to read the contents of a plain text file and write them into a variable called *VariableName*. When reading plain text files, *FileType* should be set to 'text'. **Important note: to use this from Moodle, *FileName* must be null, as the app will show a menu to let the user select which file he/she wants to read data from.**
- Method *_readState(null)*, to read and load the complete list of variables of a web app simulation from a .json file.
- Method *_readVariables(null, List<String> vars)*, to read and load a listo f selected variables of a web app simulation from a .json file.

All saving methods open a modal window like the one displayed in Figure 3.a, while all loading methods open one like that shown in Figure 3.b. You don't need to create these windows yourself, as they are automatically created by the saving and loading methods.

a) **b)**

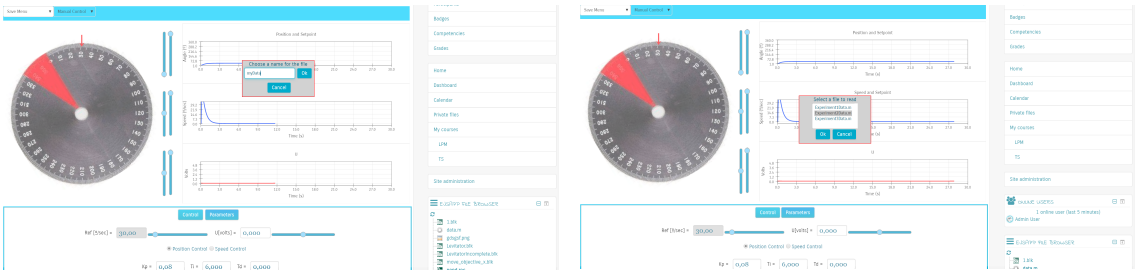

Figure 3. Modal windows created by the saving and loading methods to write/read files

2.3. Remote lab

If the EjsS app is meant to be a remote lab, then further work needs to be done to connect the HTML5 application with the software/hardware in the lab. If you want to use our Remote Interoperability Protocol (RIP) for this, read the **RIP Element User Guide** to learn how to configure and prepare your EjsS app and the **RIP Server User Guides** to learn how to prepare your lab device to accept connections from RIP clients. There are two implementations (and their corresponding guides) for the RIP server: LabVIEW and Python.

Finally, if you are using an ENLARGE system to enable communications with labs within your institutional LAN and to manage the remote lab connections, then you also need to use the ENLARGE Proxy EjsS Element, for which you should read the **ENLARGE Proxy Element User Guide**.

3. CONFIGURING AND USING EJSAPP ACTIVITIES IN MOODLE

3.1. Adding an EjsS app to MOODLE

Figure 4 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.

The field for the name of the activity can be filled using the HTML nomenclature that allows Moodle to filter the text depending on the language selected by the user. This way, you can set the name of the activity both, in English and Spanish, for example (see Table 2) and it will be displayed in the appropriate language for each user.

Table 2. Code to set a text in Moodle so that it supports the multi-language filter

```
<span class="multilang" lang="es">Laboratorio Virtual del Servo Motor</span><span class="multilang" lang="en">Virtual Laboratory of the Servo Motor</span>
```

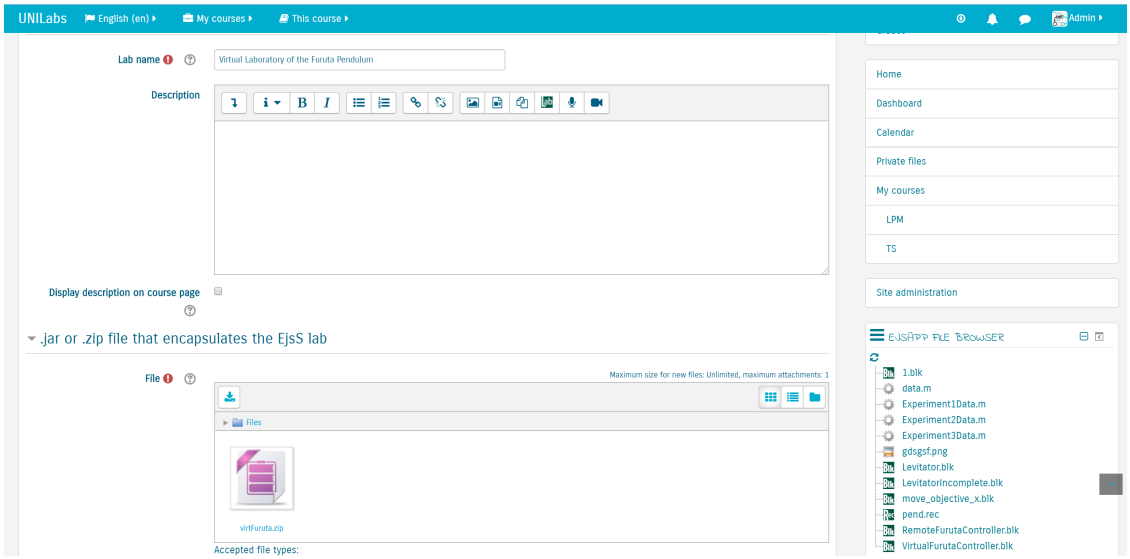

Figure 4. Adding an EjsS app activity in MOODLE

Once the name of the activity has been established and the EjsS .zip file has been uploaded, several options can be configured in the activity. Next sections go through these options.

3.2. Loading an initial state

When an EJSApp activity is created in MOODLE, it is possible to add a .json file that contains a particular state of the lab. When this is done, the system will automatically load the app in the state contained in the .json file. To do this, just add the .json file to the "*.json file with the state to be read when this EjsS lab loads*" section of the add/edit form (Figure 5) of the EJSApp activity.

This option allows loading a different state to the one that is set by default in the EjsS application. This, in turns, can be used to propose different scenarios or predefined use cases (with a special didactical value) with the same lab system.

To generate these states and store them in .json files, use the `_saveState()` method introduced in Table 1.

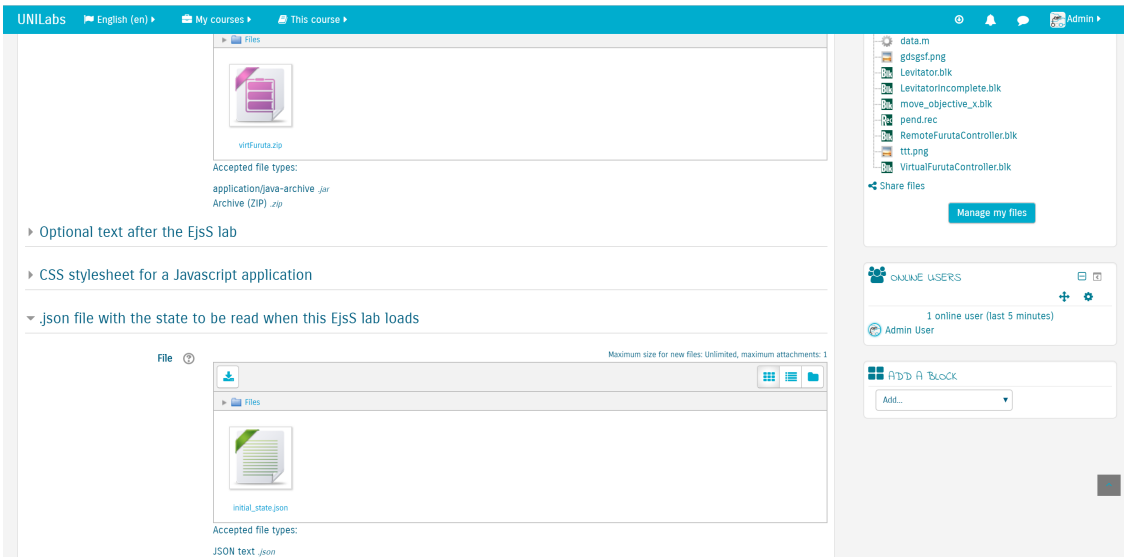

Figure 5. Loading an initial state in and EJSApp activity

3.3. Loading a recorded interaction

It is also possible to add a text file with .rec extension that contains a script with the instructions to automatically reproduce the interactions performed by a user with the EjsS app. This can be done in the "*.rec file with the recording to be run when this EjsS lab loads*" section of the add/edit form, similar to the two previous ones.

This option allows to automatically load and reproduce the interactions performed by a user with the EjsS app, similar to a video but with the simulation model or the remote lab communications running in real time. This can be used as demonstrations, tutorials and so on.

To record these interactions and store them in .rec files, use the EJSApp File Browser Block.

3.4. Visual programming interaction with the web app

A visual programming language base don Blockly can be used to let users interact with the EjsS app through block coding. This opens many possibilities, bringing the experimentation closer to a sandbox experience. Figure 7 shows an example of configuration, where the different programming blocks categories can be added or not to the library of blocks the users would have available to work with the EjsS app.

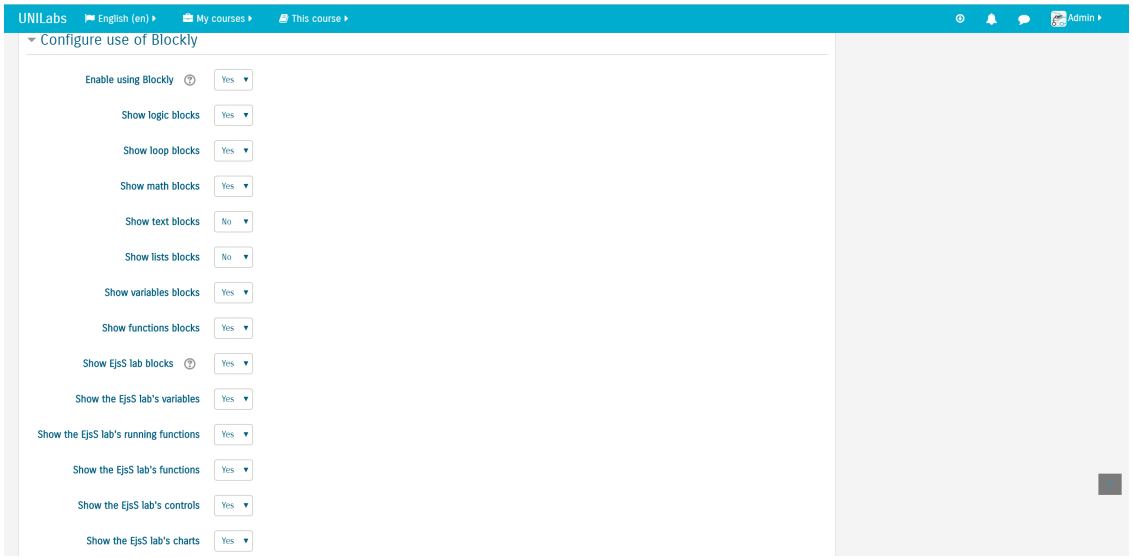

Figure 6. Enabling visual programming with Blockly in EjsS activities

In addition, it is possible to add a .blk file if we want an initial blockly program to be loaded when a user enters in the EJSApp activity, as shown in figure 8.

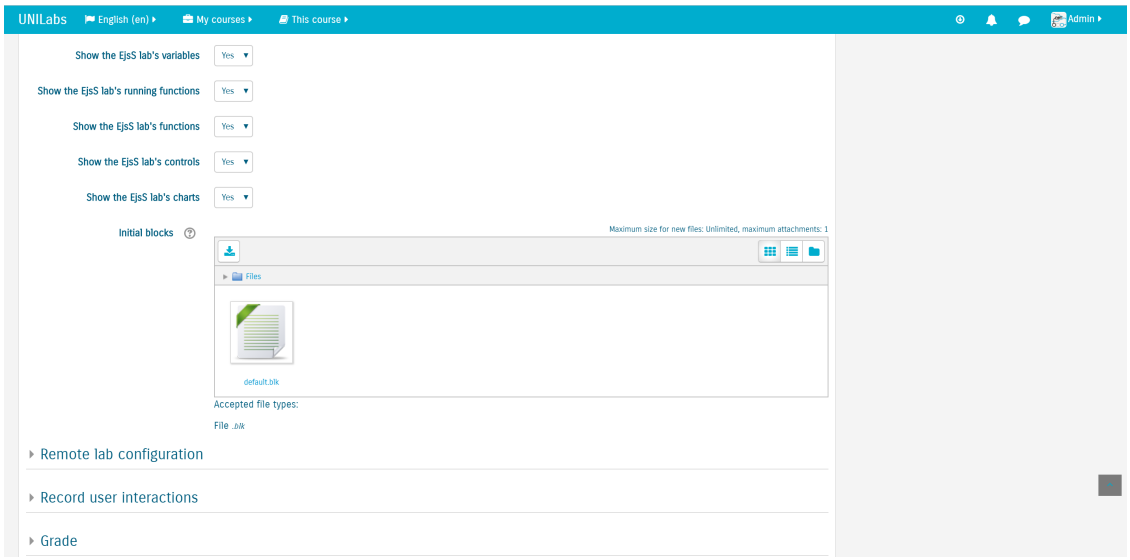

Figure 7. Enabling visual programming with Blockly in EjsS activities

Figure 9 shows and example of an EjsS app whose EJSApp activity was configured to enable the visual programming interaction and load an initial block program.

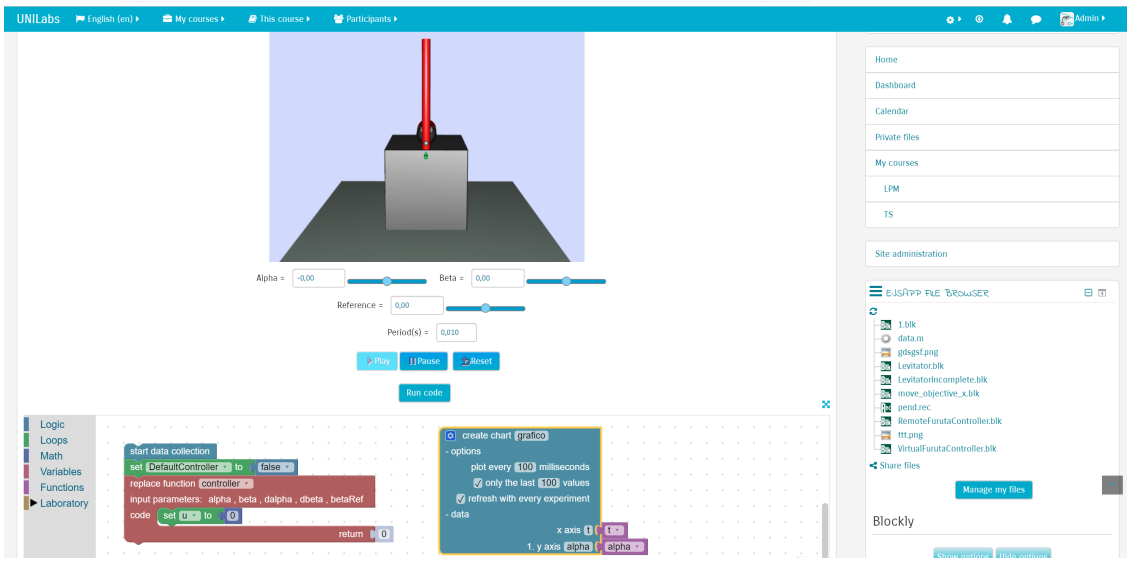

Figure 8. An EJSApp activity with the visual programming language enabled and a default block program

3.5. Remote labs

When adding a remote lab in an EJSApp activity, it is necessary to specify this in the "*Remote lab configuration*" section within the add/edit form. This part is heavily linked to the Remlab Manager block, but when such block has been installed, configured and used to define a remote lab, identifiers for these already defined labs will appear here. Then, it is as simple as changing the "*Remote experimental system?*" dropdown menu in the add/edit form to "*Yes*" and selecting the right experience identifier in the "*Practice identifier*" dropdown menu (see Figure 10).

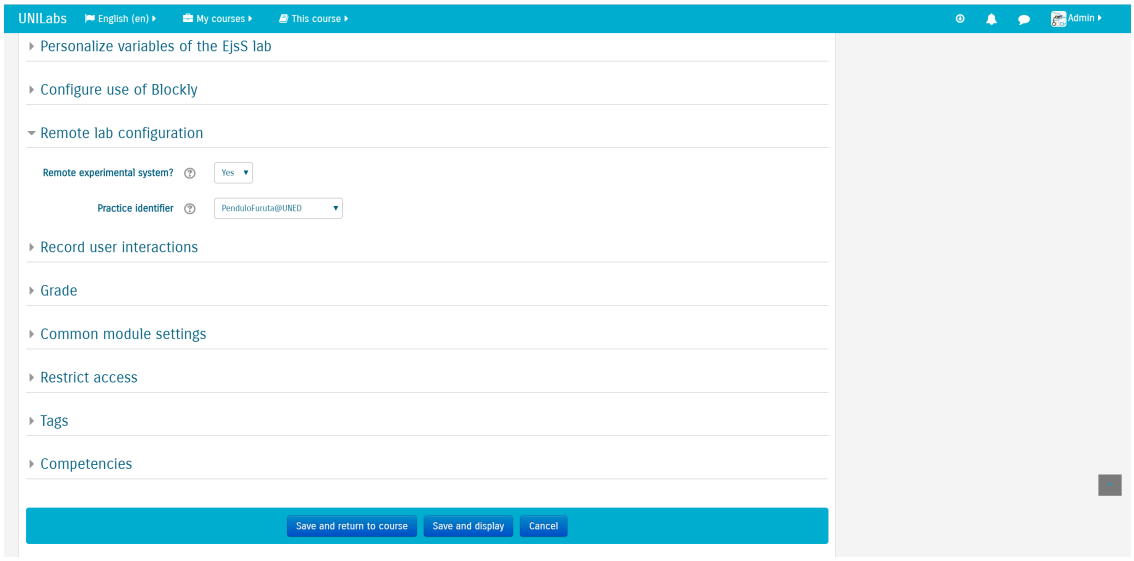

Figure 9. Configuring and EJSApp activity as a remote laboratory