Learning JavaScript in a Local Playground

Ricardo Queirós
Department of Informatics – School of Media Arts and Design, Polytechnic of Porto, Portugal
CRACS INESC TEC, Porto, Portugal
http://www.ricardoqueiros.com
ricardoqueiros@esmad.ipp.pt

Abstract

JavaScript is currently one of the most popular languages worldwide. Its meteoric rise is mainly due to the fact that the language is no longer bound to the limits of the browser and can now be used on several platforms. This growth has led to its increasing use by companies and, consequently, to become part of the curriculum in schools. Meanwhile, in the teaching-learning process of computer programming, teachers continue to use automatic code evaluation systems to relieve their time-consuming and error prone evaluation work. However, these systems reveal a number of issues: they are very generic (one size fits all), they have scarce features to foster exercises authoring, they do not adhere to interoperability standards (e.g. LMS communication), they rely solely on remote evaluators being exposed to single point of failure problems and reducing application performance and user experience, which is a feature well appreciated by the mobile users. In this context, LearnJS is presented as a Web playground for practicing the JavaScript language. The system uses a local evaluator (the user's own browser) making response times small and thus benefiting the user experience. LearnJS also uses a sophisticated authoring system that allows the teacher to quickly create new exercises and aggregate them into gamified activities. Finally, LearnJS includes universal LMS connectors based on international specifications. In order to validate its use, an evaluation was made by a group of students of Porto Polytechnic aiming to validate the usability of its graphical user interface.

2012 ACM Subject Classification Software and its engineering → Development frameworks and environments

Keywords and phrases programming languages, gamification, e-learning, automatic evaluation, web development

Digital Object Identifier 10.4230/OASIcs.SLATE.2019.10

Funding This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project : UID/EEA/50014/2019.

1 Introduction

Nowadays, we are seeing a great evolution of the JavaScript language. The language commonly used on the client-side of the Web, has freed itself in recent years from browser boundaries and is nowadays also used on the server side (Node.js) and also on other platforms such as desktop (e.g. Electron) and mobile devices (NativeScript, React Native). With this evolution, companies began to adopt more and more the language for the multi-platform development. This growth in the companies made the schools begin to integrate the teaching of language in the courses curricula.

In the teaching-learning of computer programming, one of the most used teaching methodologies is the massive resolution of programming exercises. The methodology enhances the practice, but on the other hand creates an excessive workload on the teacher who has to evaluate all the exercises of all the students. To mitigate this task, automatic evaluation systems have been used in recent years, whose mission is to assess and give feedback on student performance. Despite their apparent success, these systems present several problems, namely issues related to performance, interoperability, feedback or even personalized gamification.

This article introduces LearnJS as a playground for JavaScript language training. The system provides a set of thematic activities composed of various types of educational resources (exercises, videos). After the resolution of the exercise, the system presents feedback on the resolution allowing the user to progress in the activity in a sustained way.

LearnJS distinguishes itself from other systems in various facets, namely in the exercises authoring, performance and interoperability.

In the following sections we will focus the attention in the computer learning environments, namely, the code playgrounds. Then, we present LearnJS as a tool to practice JavaScript. In this section we present its architecture, main components, data and integration models and the playground GUI. In the next section, we conduct an usability survey for the LearnJS user interface. Finally, we share the main contribution of this work and point out some future work.

2 Computer Programming Learning

The process of learning computer programming can be very complex and demotivating [1, 3, 6]. In order to help with those issues, several approaches appeared trying to simplify the process and make it accessible to everyone, even those with absolutely no coding experience or knowledge [7, 8]. These approaches come in several formats ranging from non-interactive approaches (e.g. YouTube channels, blogs, books) to integrated and interactive solutions (e.g. intelligent tutors, on-line coding providers, code playgrounds, pastebins, MOOCs). All these approaches can be grouped in four categories as shown in Figure 1.

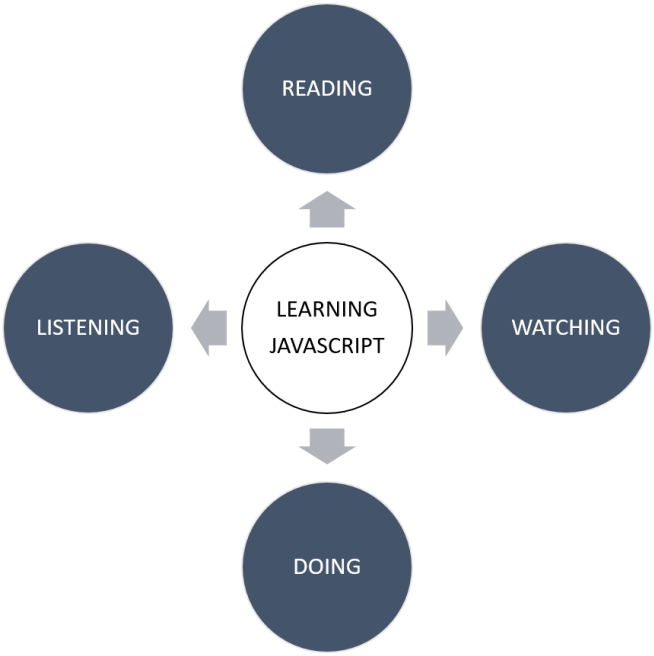

Figure 1 Learning approaches.

One of the best ways to learn is by reading books. Books are always a solid knowledge base because they synthesize best practices and guarantee a quality certified by a publisher. The disadvantage of books is that in the area of Information Technology, always in constant update, they become quickly obsolete. Other reading options are modern development blogs (e.g. Dev.to, Medium) and newsletters.

We can also observe or listen others to code. The former, can be useful if we want to dig deeper into certain aspects of coding. The later, is also a great way to learn to code without needing to be actively tied to a screen. In this realm, the podcasts community is evolving at a good pace[1].

The last, and arguably most important, part of learning JavaScript is the actual doing part: writing code, getting it to work, and repeating. This is the intrinsic concept of practice. However the practice will only be useful if there is immediate and expressive feedback on what we practice. The absence of feedback or the inclusion of false positives is detrimental to practice and will only make the learner more demotivated. Several online learning environments appeared to address theses needs. The goal of this section is not to detail all of them in depth, but rather to focus on those that have the following characteristics:

- Web-based
- practice oriented
- with automatic feedback
- based on gamified challenges
- JavaScript language support

Based on these features, **online code playgrounds** are increasingly dominating and a selected set of them will be detailed and compared in the next section.

2.1 Front-End Code Playgrounds

A variety of front-end code playgrounds have appeared over the years. These type of tools are extremely flexible and help students to practice a programming language. These tools have several common features. Among the main features of these tools are: collection of exercises organized by modules or categories, sophisticated JavaScript code editors with preview windows, developer consoles and code validation tools, code sharing facilities, gamification elements, such as level unblocking, badges and rankings.

There are a lot of playgrounds with these main features. A selected set of Code Playgrounds will be described:

- Sololearn (SL)
- CoderByte (CB)
- CodeWars (CW)
- CodinGame (CN)
- HackerRank (HR)

In the following section, we compare these code playgrounds based on set of predefined criteria such as: edition, evaluation, gamification and interoperability.

2.1.1 Edition

This type of Web-based tools use typically a sophisticated JavaScript editor (e.g. Ace) with some of the inherited functionality of traditional IDEs editors such as syntax highlighting, intelligent code completion, themes, snippets and syntax checker.

At the same time, there are several types of exercises that may be available in the editor: exercises where the student will have to code a solution for a particular problem, puzzles, fill in the blanks, questionnaires, etc.

Table 1 compares these features on the select editor.

[1] URL: `https://player.fm/podcasts/Js`

Table 1 Edition features.

Features	SL	CB	CW	CN	HR
Syntax Highlighting	X	X	X	X	X
Code Completion	X	X	X	X	X
Themes	X	X	-	X	-
Snippets	-	-	-	X	X
Syntax checkers	X	-	X	X	X
Quizzes	X	X	-	-	-
Puzzles	-	-	-	-	X
Fill the blanks	X	-	X	-	X

2.1.2 Evaluation

A programming exercise can be evaluated dynamically and/or statically.

The dynamic evaluation consists in the definition of a set of test cases composed by input data and corresponding output. In the evaluation process the input data is injected into the student's solution and the result is obtained. This same result is compared to the output data provided by the teacher. If they are identical it means that the test was successful and the procedure for the remaining tests is repeated. Typically several predefined tests are provided in the learning environment. The student can perform only one or all of the tests at any time. Some systems provide the functionality of adding new test cases on the part of the student in order to encourage the creation of tests. Even in the tests it is possible to have hidden test cases in order to prevent the student from solving the tests-oriented challenges and not the problem to solve in a generic way. Each test, in addition to the input and expected output, may have associated feedback that will be shown to the user if that test fails.

In the previous evaluation type, the feedback that is given to the student is whether the test has passed or not. In short, the student will have a test to say that his solution is correct or, alternatively, that it is not correct because it failed some tests, undercutting them. To enrich the feedback it is necessary to apply a static analysis that instead of executing the student code and inject input data, does an introspect to the code, without executing it, and checks a predefined set of metrics. In this context, the presence of a certain keyword or block of code, the style of code (convention of variables names), the presence of comments or even the application of certain algorithm can be verified. For this type of analysis linters or other static analysis tools are usually used.

Table 2 compares these features on the select editors.

Table 2 Evaluation features.

Features	SL	CB	CW	CG	HR
Dynamic Analysis	X	X	X	X	X
Hidden Tests	X	X	X	X	X
Feedback Tests	X	-	-	X	X
New tests	X	X	-	-	-
Static Analysis	-	-	X	-	X

2.1.3 Gamification

Regarding gamification and social features, most platforms adhere to the same components, such as forums (F), learning dashboards (D), user profiles (UP), comments (COM), recommendation (REC), levels and badges. For instance, CodePlayer offers a different approach to learning code by playing code like a video, helping people to learn front-end technologies quickly and interactively. The platform also includes a commenting tool and links to related walkthroughs. CodeAcademy includes a user progress dashboard informing of the current state of the learner regarding its progress in the courses. This platform enhances the participation in the courses by also including achievements (ACH) that are rewarded with badges and users are also able to share completed projects with the rest of the site community and potentially showcase their skills to employers. Except for Code.org, all the platforms have a strong presence in the mobile world, with app versions for Android and iOS.

Table 3 compares these features on the select editors.

Table 3 Gamification features.

Features	SL	CB	CW	CG	HR
Forums	X	X	X	X	X
Dashboards	X	X	-	X	X
User Profiles	-	X	X	X	X
Comments	X	-	X	X	X
Recommendation	X	-	-	X	X
Levels	-	X	-	X	X
Badges	X	X	X	X	X
Achievements	X	X	X	X	X
Likes and followers	-	X	-	X	X
Awards	X	X	X	X	-
Hints and code skeletons	-	X	-	X	X

2.1.4 Interoperability

The category of interoperability is perhaps one of the most important, but possibly one of the most overlooked by learning environments. Most systems live on their own server without any integration with existing systems and have proprietary forms of exercises and activities. Communication with other systems, such as evaluators or plagiarism systems, is done in a ad-hoc way using internal APIs.

Also the integration of these playgrounds with Learning Management Systems (LMS) would be an expected and desirable functionality. The student authenticates in the LMS and solves a programming activity in the playground. At the end of the activity, a report on student activity is automatically sent and its grade book updated in the LMS.

At the same time, the authorship of programming exercises remains complex. Not only by the absence of systems that facilitate his construction, but also by the absence of standard formats that allow his formal definition, dissemination and discovery by other systems. The lack of repositories with these characteristics makes reusing and adapting exercises difficult.

Table 4 compares these features on the select editors.

Table 4 Interoperability features.

Features	SL	CB	CW	CG	HR
Prog. exercise formats	X	-	X	-	X
Activities formats	X	-	-	-	X
LMS integration	X	-	-	X	-
Repository integration	X	X	-	X	X
Gamification standards	X	-	-	X	X

3 LearnJS

LearnJS is a Web playground which enables anyone to practice the JavaScript language [4]. The architecture of LearnJS is depicted in Figure 2.

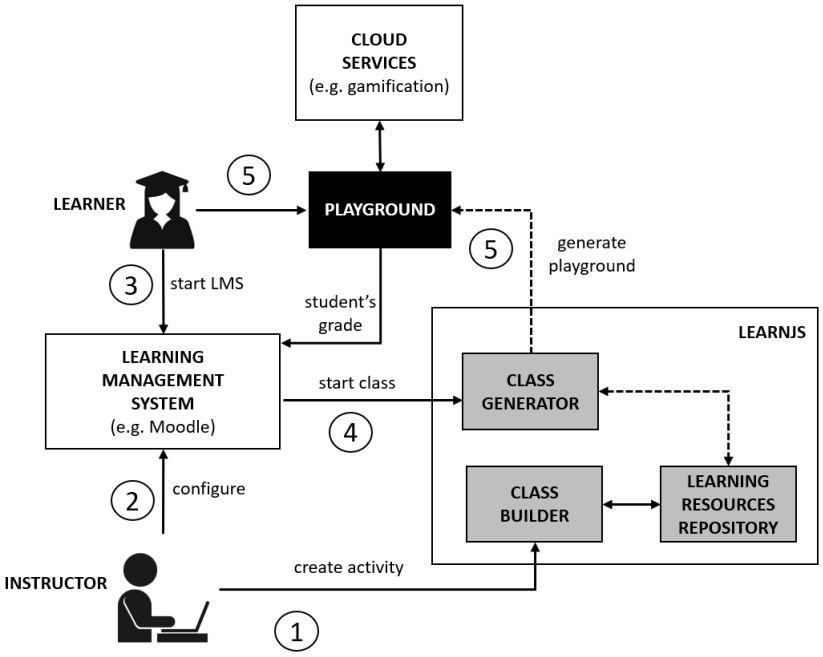

Figure 2 LearnJS architecture.

At its core, LearnJS is composed by two components used by the two system user profiles:

- Instructors: use the LearnJS Class Builder to create and select resources from the Learning Resources Repository in order to compose a learning activity. Next, they configure the activity in a Learning Management System.
- Learners: they run the LMS in a browser and launch the activity in the LMS. The launch process is controlled by the class generator that receives a LearnJS manifest and generates the corresponding activity in a playground which is returned to the user. Beyond the internal gamification features, the playground can benefit from other Gamification Services to foster student's competitiveness and engagement.

3.1 Data Model

The LearnJS data model is composed by two entities: the activity and the resource.

A resource is an educational object, either expository or evaluative. An expository resource is typically a video or a PDF file showing how to master a specific topic. An evaluative resource is a JavaScript challenge to be solved by coding. Instructors can use the LearnJS class builder to contribute with new resources. The supported resources in LearnJS follow Sweller and Cooper [6] paradigm based on a learner centred approach to define a constructivist learning model. This model foster the learning by viewing and learning by doing approaches where educational resources, either expository or evaluative, play a pivotal role. A resource is defined as a JSON file which should comply with the LearnJS official resource schema formalized by a public JSON Schema [2]. It contains basic properties for identification and metadata purposes.

Instructors can also perform other operations in the class builder, such as the creation of activities. An activity combines a set of resources of several types (evaluative, expositive) with gamification attributes. An activity JSON file is composed by several properties. We highlight two:

- levels: can be considered as sub-activities composed by a set of resources identified in the resources sub-property. Students should see and solved the respective resources of the level. The completion of the level and the respective unlock of the next level is granted after the student solved a specific percentage of evaluative resources defined in the perc property of the level.
- gamify: a set of attributes that can be assigned to resources. After a success completion of an evaluative resource, students can be awarded in multiple forms. Hence, the award property can have one of the following values:
  - HintExtra: gives an extra point to the learner. The learner can spend the hint points on any exercise by unhiding the hint associated;
  - ShowNextSkeleton|ShowAllSkeleton: gives the learner the ability to unhide the code skeleton associated to the next (or all) gamified resources;
  - UnlockLevel|UnlockAllLevels: gives the learner the ability to unlock the next (or all) level.

The example on Listing 1 shows an activity JSON instance for learning JavaScript arrays:

Listing 1 Learning activity JSON instance.

```
{
 "id": "http://learnJS/activities/129387",
 "title": "Learn the basics of Arrays",
 "metadata": {
   "author":"Ricardo Queiros",
   "date":"19-04-1975",
   "level":"basic",
   "tags":["arrays"]
 },
 "levels": [
  {"id":"1", "name":"Basic operations", "perc":"75",
          "resources": ["...resources/125412", "..."]},
  {"id":"2", "name":"Sort", "perc":"50", "resources":["..."]}
 ],
 "gamify": [
```

[2] Public GitHub link: `https://github.com/.../learnJS`

```
    {"resource": ".../resources/125412", "award":"HintExtra"},
    {"resource": ".../resources/225232", "award":"ShowNextSkeleton"}
  ]
}
```

3.2 Integration Model

The purpose of LearnJS is also to integrate an e-learning ecosystem based on an LMS (e.g. Moodle, Sakai, BlackBoard). For this, it benefits from the interoperability mechanisms to provide authentication directly from the LMS and to submit exercises grades back to the LMS, using the Learning Tools Interoperability (LTI) specification.

3.3 Playground

The LearnJS Playground is a Web-based component which will be used by learners to browse learning activities and interact with the compound resources. Here students can see videos of specific topics and solve exercises related with those topics with automatic feedback on their resolutions (Figure 3).

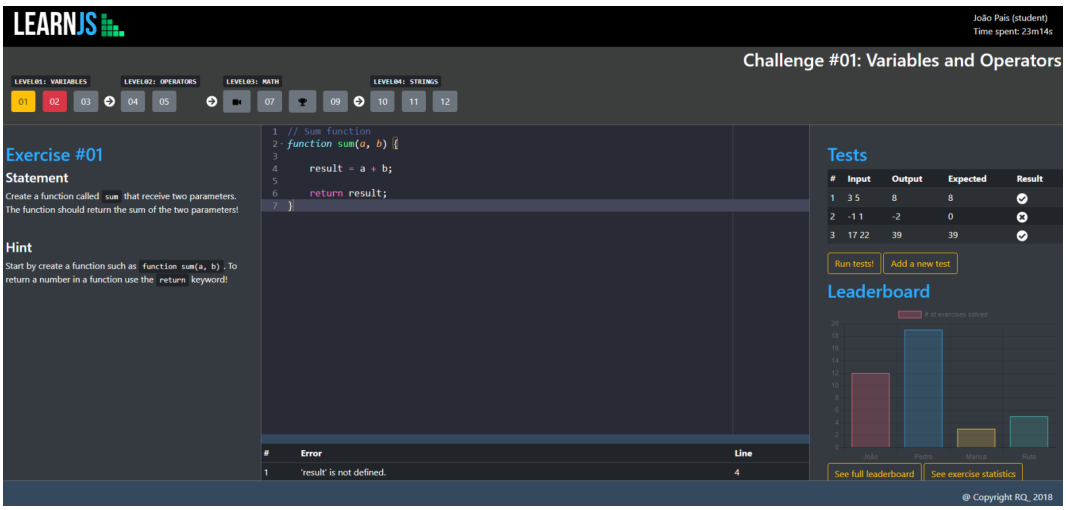

Figure 3 LearnJS playground GUI.

The playground is composed by three main components:

1. Editor: allows students to code their solutions in an interactive environment;
2. Evaluator: assess the student's solution based on static and dynamic analyzers;
3. Gamification Engine: gamifies the learning activity with the management of levels and several awards.

For the Editor component, the playground uses Ace (maintained as the primary editor for Cloud9 IDE) which can be easily embedded in any web page and JavaScript application. The editor is properly configured for the JavaScript language and supports the Emmet toolkit for the inclusion of dynamic JavaScript snippets. Ace editor can display errors on the editor itself but does not handle language dependencies. A parser needs to be used to detect errors and determine their positions on the source file. There are several tools that can improve code quality. One of such cases is code linters. Linters (e.g JSLint, JSHint) can detect potential

bugs, as well as code that is difficult to maintain. These static code analysis tools come into play and help developers spot several issues such as a syntax error, an unused variable, a bug due to an implicit type conversion, or even (if properly configured) coding style issues. LearnJS uses JSHint to accomplish this behavior. While static code analysis tools can spot many different kinds of mistakes, they can not detect if your program is correct, fast or has memory leaks. For that particularly reason, LearnJS combines JSHint with functional tests (based on test cases). For this kind of tests, and since the code is written in JS and the context is the browser, we use a simple approach by iterating all the case tests and applying the eval function for tests injection.

Both analyzers (linter and Test Case runner) are subcomponents of the LearnJS evaluator component that runs exclusively on the client side. This approach avoids successive round-trips to the server which affects negatively the user experience. Lastly, the Gamification Engine component is responsible for loading/parsing the LearnJS manifest and fetching resources from the learning resources store. If levels are defined, the engine sequences and organizes the resources properly. Upon completion of evaluative resources from students, the engine deals with all the logic associated with the respective awards by unhiding/unlocking features of next challenges. Finally, the component send the results back to the server. At this moment, we have a simple running prototype. The source code is available at a GitHub repository. Figure 3 shows the front-end GUI of the playground.

4 Evaluation

This section evaluates the usability of the graphical user interface of LearnJS based on the Nielsen's model [2].

4.1 Nielsen heuristics

According to Nielsen the practical acceptability of a system includes factors such as usefulness, cost, reliability and interoperability with existing systems. The usefulness factor relates the utility and usability offered by the system. Utility is the capacity of the system to achieve a desired goal. As the system perform more tasks, more utility he has. Usability is defined by Nielsen as a qualitative attribute that estimates how easy is to use an user interface. He mentions five characteristics involved in the concept of usability:

- ease of learning - the system should be easy to learn so that the user can start doing some work with the system;
- efficiency - the system should be efficient to use, so after the user learns the system, a high level of productivity is possible;
- memorability - the system should be easy to remember so that the casual user is able to return to the system after a period without using it, without requiring to learn it all over again;
- errors - the system should prevent the user from committing errors as should deal with them gracefully and minimizing the chance of occurring catastrophic errors;
- satisfaction - the system should be pleasant to use so that users become subjectively satisfied when using.

4.2 Usability evaluation

LearnJS was evaluated according to the Nielsen's model using a heuristic evaluation methodology. A heuristic evaluation is an inspection method for computer software that helps to identify usability problems in the user interface (UI) design. Jakob Nielsen's heuristics

are arguably the most used usability heuristics for user interface design. Based on these heuristics a questionnaire with 41 questions was designed in Google Forms. The aim of the questionnaire is to identify usability problems in the UI design of LearnJS.

The experiment took place at the Media Arts and Design School (ESMAD) - a school of the Polytechnic Institute of Porto - during the month of March of 2019. The participants were students from the first-year of the course Object-Oriented Programming. This course is offered to the Web Technologies degree and aims to introduce students to programming concepts using the JavaScript language.

Students used LearnJS during one week. In the end, they were invited to fill a survey using Google Forms. The questionnaire includes questions on LearnJS usability. It had an average of 45 responses (the equivalent to 91% of the total of students).

Figure 4 shows the results obtained grouped by the Nielsen's heuristics. The data collected are shown in a chart graphs where the heuristics are sorted in ascending order of user satisfaction.

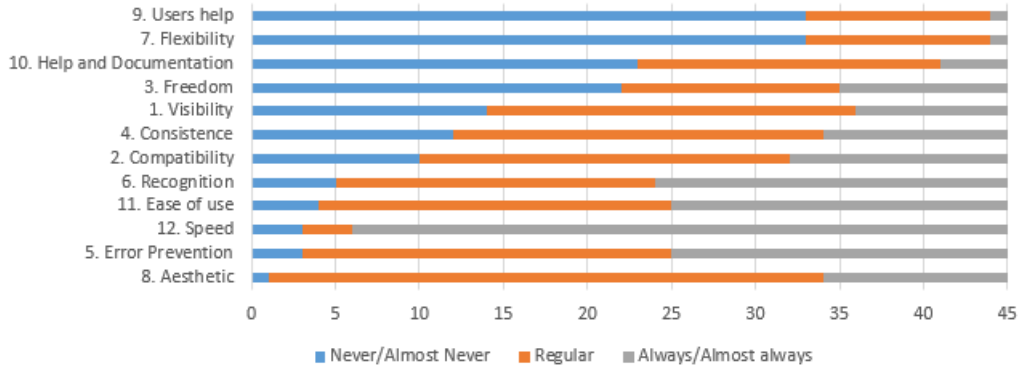

Figure 4 LearnJS usability evaluation.

The results highlight deficiencies in three areas: users help, flexibility, help and documentation and freedom. In regard to the flexibility of the system, respondents considered that the system do not allow the personalization of the interface, more precisely, the activation/deactivation of certain functions and the configuration of the screens. The possibility of use of accelerator keys to speed up the interaction with the editor and evaluator is also a handicap of LearnJS at this moment.

The Help and documentation is another heuristic with negative values. The respondents state that is difficult to find help and documentation in LearnJS. This a fact since we did not yet integrate any kind of tutorial mode in the system.

The freedom heuristic is the fourth worst facet. Most of the complaints focused on the inability to cancel or roll back mistakes made to a previous and safe state.

The respondents also reveal that the error messages are sometimes unclear and inadequate in LearnJS. The respondents also state that the documentation is scarce and is hard to find it.

In an overall comment, one can conclude that the majority of students classified LearnJS as a good tool according to the parameters evaluated.

5 Conclusions

In this paper we present LearnJS as a flexible playground for JavaScript learning. The paper stresses the design of the platform divided in two main components: the management tool and the playground. In the former, instructors can contribute with new exercises and

bundle related exercises in learning activities. All these entities were formalized using JSON schemata. The later, allows students through a sophisticated and interactive UI, to see and solve educational resources (mostly, videos and exercises). In order to engage students, the platform can be configured to gamify resources through the subgrouping of activities in levels, the assignment of awards and the exhibition of a global leaderboard. The main contributions of this work is the design of a platform with interoperability concerns in mind and the respective schemata for the simple concepts of educational resources and activities.

LearnJS is already being used in a Polytechnic School with promising results. In this paper, we present an usability survey that shows several issues which are already being tackled.

As future work, in mid 2019 we will start to work on several components, namely:

- **Class Builder** - to foster activities creation based on educational resources. We will also create a GUI for assisting instructors in the creation of exercises.
- **Importer** - to import exercises from other repositories. The importer will use an existing converter service for exercises with different formats called BabeLO [5].
- **Gamified sequencer** - to dynamically sequence exercises based on student's pace and progress. The sequencer should be enhanced by the gamified elements included in the activity.

References

1 Kirsti M Ala-Mutka. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*, 15(2):83–102, 2005. doi:10.1080/08993400500150747.

2 J. Nielsen. *Usability engineering*. Academic Press, 1993. URL: https://books.google.pt/books?id=fnvJ9PnbzJEC.

3 Jackie O'Kelly and J. Paul Gibson. RoboCode & Problem-based Learning: A Non-prescriptive Approach to Teaching Programming. *SIGCSE Bull.*, 38(3):217–221, June 2006. doi:10.1145/1140123.1140182.

4 Ricardo Queirós. LearnJS - A JavaScript Learning Playground (Short Paper). In *7th Symposium on Languages, Applications and Technologies, SLATE 2018, June 21-22, 2018, Guimaraes, Portugal*, pages 2:1–2:9, 2018. doi:10.4230/OASIcs.SLATE.2018.2.

5 Ricardo Queirós and José Paulo Leal. BabeLO - An Extensible Converter of Programming Exercises Formats. *TLT*, 6(1):38–45, 2013. doi:10.1109/TLT.2012.21.

6 Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2):137–172, 2003. doi:10.1076/csed.13.2.137.14200.

7 Elena Verdú, Luisa M. Regueras, María J. Verdú, José P. Leal, Juan P. de Castro, and Ricardo Queirós. A Distributed System for Learning Programming On-line. *Comput. Educ.*, 58(1):1–10, January 2012. doi:10.1016/j.compedu.2011.08.015.

8 J. Xavier and A. Coelho. Computer-based assessment system for e-learning applied to programming education. In *ICERI2011 Proceedings*, 4th International Conference of Education, Research and Innovation, pages 3738–3747. IATED, 2011.