Can a Content Management System Provide a Good User Experience to Teachers?

Yannik Bauer
DCC – FCUP, Porto, Portugal
CRACS – INESC TEC, Porto, Portugal

José Paulo Leal
CRACS – INESC TEC, Porto, Portugal
DCC – FCUP, Porto, Portugal

Ricardo Queirós
CRACS – INESC TEC, Porto, Portugal
uniMAD – ESMAD, Polytechnic of Porto, Portugal

Abstract

The paper discusses an ongoing project that aims to enhance the UX of teachers while using e-learning systems. Specifically, the project focuses on developing the teacher's user interface (UI) for Agni, a web-based code playground for learning JavaScript. The goal is to design an intuitive UI with valuable features that will encourage more teachers to use the system. To achieve this goal, the paper explores the use of a headless Content Management System (CMS) called Strapi. The primary research question the paper seeks to answer is whether a headless CMS, specifically Strapi, can provide a good UX to teachers. A usability evaluation of the built-in Strapi UI for content creation and management reveals it to be generally consistent and user-friendly but challenging and unintuitive to create courses with programming exercises. As a result, the decision was made to develop a new teacher's UI based on the existing Agni UI for students in an editable version. Once the development is complete, a new usability evaluation of the fully developed teacher's UI will be conducted with the Strapi UI evaluation as a baseline for comparison.

2012 ACM Subject Classification Applied computing → Interactive learning environments

Keywords and phrases learning environment, programming exercises, programming learning, automatic assessment, headless CMS, CMS, user experience

Digital Object Identifier 10.4230/OASIcs.ICPEC.2023.4

Category Short Paper

Funding This research was conducted within the "FGPE Plus: Learning tools interoperability for gamified programming education" project supported by the European Union's Erasmus Plus programme (agreement no. 2020-1-PL01-KA226-HE-095786), and financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020.

1 Introduction

Learning programming can be challenging for beginners. Winslow [13] noted that many novice programmers might know the syntax and semantics of individual statements but struggle with combining these features into valid programs. His and other studies [11, 3] have emphasized the importance of practice through exercises, which is most effective with immediate feedback. However, it is impossible for teachers to manually provide immediate feedback for every exercise. Automated assessment systems for programming exercises emerged as a solution, freeing teachers to focus on students needing additional support.

While there are e-learning systems that offer the ability to create and manage courses with automated assessment, many of them were designed to focus on the UX of students and less on the teacher's UX, which can result in fewer teachers utilizing them.

This paper presents the design and development of a UI to manage and author course contents with automated assessment for Agni, a code playground for learning JavaScript. Previously, Agni consisted only of a UI for students. The course contents were saved in configuration files without the ability to alter or create them through an interface. The main objectives for the UI are 1) a good user experience for teachers; 2) an effective way of creating, adapting, and managing course contents with automated assessment; 3) a repository of exercises with the possibility of extending it with other repositories and 4) allowing for sequencing the course content.

The idea was to use a headless CMS to achieve the goal. This offers the flexibility to use a separate UI for displaying the content (in our case, the Agni Student UI) without being tied to a specific model, as with traditional CMSs. Also, headless CMSs provide a UI to manage and create the contents, which the teachers could use. Strapi was selected for this task. However, a usability evaluation showed that the UI for creating and managing content was generally consistent and user-friendly but challenging and unintuitive for creating course content with programming exercises. This, and the inability to customize the UI, led to the decision to develop a new UI for the teachers. The design approach was to leverage the Agni student's UI and make it editable. This means having input fields instead of text boxes, for example, for lesson names, and adding missing functionalities, such as the addition of new modules. The work is ongoing, and once development is complete, a usability evaluation of the fully developed teacher's UI will be conducted and assessed its effectiveness with the previous evaluation as a benchmark.

The remainder of this paper is organized as follows. Section 2 presents the state of the art, covering user experience and existing programming virtual learning systems (VLEs). The following section describes the main parts of the user interface, a presentation of the data model, and a subsection about the API. The evaluation method and evaluation of Strapi's UI are presented in Section 4. The final section summarizes the paper's contributions and highlights future work.

2 State of The Art

The field of programming VLEs is constantly evolving, with emerging trends and innovations shaping how students interact, and teachers create programming courses. This section starts with a general review of UX and explores teachers' UX in VLEs, analyzing their features and missing potentials.

2.1 User Experience

According to ISO 0241 – 210, the definition of UX is "A person's perceptions and responses that result from the use or anticipated use of a product, system or service.". However, the concept of UX encompasses various aspects, leading to multiple possible definitions, dynamic concepts, and theoretical models, including aesthetics, usability, effectiveness, pleasure-based, and emotional experience. [4, 1, 6] provide more detailed information about the different concepts and challenges related to UX.

When designing a UI, authors agree that the focus must be on the user's needs rather than solely relying on the designer's perspective. Therefore, evaluation is crucial to gain insights into the user's experience. There are several evaluation methods, including questionnaire-based evaluation, such as the User Experience Questionnaire (UEQ) by Martin Schrepp,

Bettina Laugwith, and Theo Held [5], or a questionnaire based on the 10 usability heuristics for User Interface Design described by Jakob Nielsen in [7]. These heuristics include visibility of system status, match between system and real world, user control and freedom, consistency and standards, error prevention, recognition rather than recall, flexibility and efficiency of use, aesthetic and minimalist design, help users recognize/diagnose and recover from errors, and help and documentation. Other evaluation methods are explained in more detail in [2, 8]

2.2 System Review

Various systems are available to assist teachers in creating and managing programming courses, including Udemy, Mooshak 2, and Moodle. Typically, the course content is structured within one or two levels, consisting of modules and lessons where teachers can create materials and exercises.

The most common exercises with automated assessment are multiple-choice quizzes and programming exercises, supported by many systems such as Udemy, Mooshak 2, and Coderbyte. The correctness of the student's code for programming exercises is being evaluated dynamically. This means that predefined test cases run on the code to verify its correctness and efficiency and give immediate feedback to the student. The tests can be divided into dynamic and static testing [12]. Static testing refers to tests for which the code does not have to run, for example, checking for expression usage or code design. On the other hand, dynamic testing is performed on code execution. Unit tests are an example of dynamic tests. Most e-learning systems with automated assessment only support the creation of dynamic tests that can either be introduced with an input and wanted output field or a file for the teacher to write the whole code for unit tests. Other helpful features, such as an exercise repository, are often missing [10].

Sequencing the course content is another interesting feature. It can be based on time conditions, where the teacher can define a date or number of weeks/days after which the students can work on specific content. Alternatively, content can be sequenced based on the progress of exercises. Mooshak 2, for example, supports the sequencing of all content based on time and exercises completed.

Although these functionalities are important, they lose their impact with an unintuitive UI. The majority of systems involve a strategy of "form filling", where the interface consists of pairs of field names and inputs to declare the data of the contents. Some systems, such as Udemy, support the visualization of the created content in a student's view to see how it will appear. Moodle's teacher UI is more similar to an editable version of the Student UI with additional functionalities, such as adding lessons and declaring metadata.

3 System Design

Agni is a web-based code playground designed for learning JavaScript and providing students with a user-friendly interface for programming courses with exercises that are being evaluated automatically. However, a crucial feature was missing - a dedicated interface for teachers to create, manage, and customize the course content, along with a backend system to independently store them. Previously, the content was saved in configuration files with no interface for editing or creation.

To address this issue, Strapi, a headless CMS, was chosen as the solution for content management. Strapi offers both a user-friendly interface and an API for quickly creating a data model and managing/creating content. The API supports CRUD (Create, Read, Update, Delete) operations and can be customized using hooks. Strapi provides three types

of data structures: single types, collection types, and components. Single types and collection types have an API endpoint and can be created and edited independently. Components are reusable structures that can be used in different collections and single types.

While Strapi's UI for content management was initially considered a hypothesis for teachers to create and manage content, a usability evaluation (explained more in Section 4) revealed that it was generally consistent and user-friendly but challenging and unintuitive for creating course content with programming exercises. Additionally, the lack of customization options in the UI led to the decision to develop a new UI to ensure a better UX for the teachers. The design of this UI, the data model to support it, and the API for the communication between the UI and the server will be explained in the following subsections.

3.1 User Interface

The teacher's UI consists of two main parts: class management and content management. The former is to import, create, and manage students and their work, which is not fully developed yet, and the latter is to create and modify courses with their materials. To ensure a positive UX for teachers and provide an intuitive interface for creating and editing courses, the Agni student's UI was transformed into an editable version (see Figure 1). This approach enables teachers to see exactly what they are modifying and how it will look to students. Currently, the system only supports course creation.

Figure 1 displays a screenshot of the UI in the state of editing or creating a course. It is divided into three panels. The left panel is a navigation menu. The right panel shows buttons for actions on the middle panel, such as save the course or exit. The middle part is the main panel where teachers are able to see the different courses, classes, etc., and edit them. In the main panel of Figure 1, the editable Agni student interface for creating or editing courses can be seen. In order to make the student UI editable, text fields, such as module name and lesson description, were transformed into input fields. Additionally, buttons add functionalities, such as creating or deleting lessons, exercises, and other elements. Furthermore, the edit icons in the menu open dialog boxes to introduce information, such as conditions for when a student can work on a lesson that the student cannot see directly in the UI.

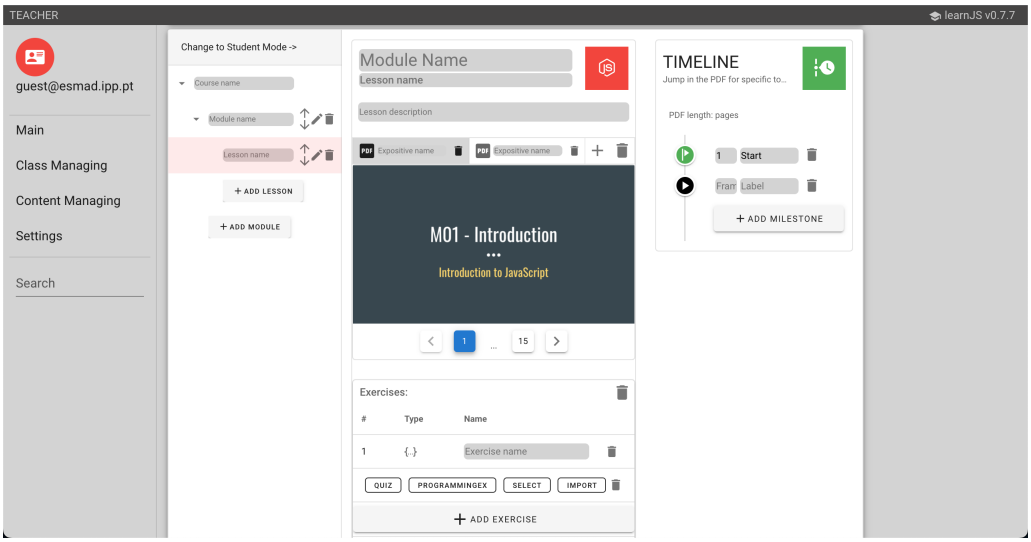

Figure 1 UI while editing or creating a course.

3.2 Data Model

The data model for the content created by teachers and viewed by students is illustrated in Figure 2. It has been implemented in Strapi and can be divided into two substructures: one for managing the classes, which teachers can access in the class management navigation, and the other for managing courses and their contents, which teachers can access in the content management navigation.

A *Course* is composed of *Modules*, which contain *Lessons*, that can include multiple *Expositive* and *Evaluative* contents. *Expositives* are used to present information to students and can be in the form of PDF files or video files. *Evaluatives*, on the other hand, are exercises that can be in the form of multiple-choice quizzes or programming exercises. To automatically evaluate these programming exercises, teachers can create *Tests* with input, expected output, type (log, expression, metric, function), and subtype (error for expression, occurrences, and lines for metric) parameters. This provides teachers with a quick way to create different types of tests without having to write complex code for unit tests. In order to sequence *Modules* and *Lessons*, they contain conditions with fields named "afterWeek" to specify after which week the student can work on a *Module* or *Lesson*, and "afterPercDone" to define a percentage of completed exercises after which the student can progress.

Occurrences, *Classes*, *Students*, and *Statuses*, form the structure for managing classes. *Occurrences* are linked to a *Course* and have a start date, determining when students can start accessing the *Course*. The "afterWeek" condition for *Modules* and *Lessons* is based on this start date. *Classes* and *Students* have fields for declaring delays that may occur during the year. The grade and solution to a programming exercise done by the student are saved in the *Status*.

Course, *Expositive*, *Evaluative*, and *Question* were chosen as collection types for easy reuse and individual editing. Similarly, *Occurrence*, *Class*, *Student*, and *Status* were chosen as collection types for quick and convenient access and creation.

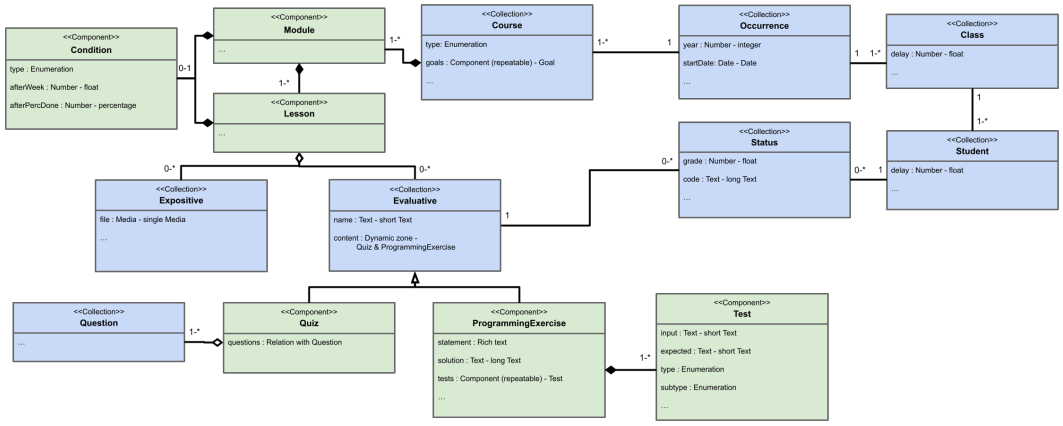

Figure 2 Main parts of the Data Model.

3.3 API

Strapi offers a RESTful API as the primary means for communication between UI and server. The API supports the full range of CRUD operations, including creating, reading, updating, and deleting content within collection types. These operations can be customized to the

specific needs of individual collection types through hooks. Additionally, the API supports a wide range of query parameters, which can be utilized to filter, sort, and paginate data within requests.

To optimize performance and reduce communication between the UI and server, the possibility of creating multiple instances of collection types with one request was implemented. Furthermore, the creation of a complete course structure encompassing different collection types, such as *Course*, *Evaluatives*, *Expositives*, and *Questions*, with one request was developed.

The API is secured using Strapis' built-in user-permission plugin, in which user roles were created for students and teachers. This plugin allows the customization of permissions for each collection and request type, depending on the user's role. For instance, only teachers are authorized to create content. When teachers create content, the author's identity is also saved in the data structure. This enables the declaration of permissions for modifying or getting content created by other teachers. For example, can they only access the data of their students. As for the students, the API enables them to access only their course, with the modules and lessons viewable contingent on the conditions established by the teacher.

4 Validation

The evaluation is an essential part of creating a UI with a good UX. Strapi's UI, which was the first idea for the teachers, was evaluated using a satisfaction questionnaire. Since the result was not satisfactory, the decision was made to develop a new UI for the teacher. The work is ongoing, and after finishing it, a satisfaction evaluation will be done with the previous evaluation used as a baseline for comparison and evaluating progress. The evaluation methodology, results of the Strapi UI evaluation, and its conclusion will be presented next.

4.1 Evaluation Methodology

The evaluation was done using a satisfaction questionnaire based on the ten usability heuristics for User Interface Design from Jakob Nielsen [7]. These are visibility of system status, match between the system and the real world, user control and freedom, consistency and standards, error prevention, recognition rather than recall, flexibility and efficiency of use, aesthetic and minimalist design, help users recognize/diagnose and recover from errors, and help and documentation. These ten heuristics and additional the easiness of learning, speed, and reliability of functions and tasks that the system wants to solve were evaluated with multiple questions on a five-point Likert scale (1 - Never, 2 - Almost Never, 3 - Regular, 4 - Almost always, 5 - Always). The questionnaire also included an overall classification of the system on a five-point scale (1 - Bad, 2 - Insufficient, 3 - Sufficient, 4 - Good, 5 - Very Good) and text fields to describe strong points, weak points, and improvement suggestions. Before completing the questionnaire, the respondents were required to carry out typical tasks performed by a professor, such as creating courses, reusing course materials, and associating them with students. Following Jakob Nielsen's proposal [9], five computer science master students from the University of Porto were selected to evaluate Strapi's UI, as this number is sufficient to identify the majority of usability issues.

4.2 Results Strapi UI

Figure 3 shows the evaluation heuristics with its average mean score and standard deviation of their questions. Many of them leading to a positive spectrum, especially speed, consistency, and emphasis, with a mean score of about 4, were evaluated positively. On the other hand,

flexibility, easiness, reliability, and overall classification, with a medium score of 2, 2.2, 2.9, and 2.8, respectively, were evaluated negatively. As strong points were mentioned, easy creation of a *Course* with *Modules* and *Lessons*, excluding *Expositives* and *Evaluatives*, and also the generally easy-to-use and fast UI. Weak points described were the need to create collection types like *Course*, *Expositive*, *Evaluative*, and *Question* individually and only after that being able to associate them with each other, which is unintuitive and time-consuming. The lack of a help system, a landing page without helpful information, and no explanation of some fields were also critiqued. A tutorial was added as an improvement suggestion.

The overall more positive evaluation of the 10 usability heuristics for Strapi's UI was expected due to it being one of the positive aspects of why people use Strapi. However, the Strapi UI is too generic for the specific necessities of managing programming courses. Different collection types can only be created separately, which makes the creation of a whole course time-consuming and not effective. Also, the inability to create a hierarchy of collection types or define different sizes for fields makes it difficult to organize and use the space in an efficient way. This led to the conclusion that the Strapi UI is not sufficient to achieve the objective of a good UX for the teacher. Nevertheless, the API and database can be used due to the ability to customize using hooks.

While students provided valuable insights into the UX, selecting teachers as respondents for the evaluation would have been more appropriate. Teachers possess a deeper understanding of the specific needs and requirements associated with their role within the system.

Category	Mean	SD
1. Visibility	3.25	0.94
2. Compatibility	3.17	0.82
3. Freedom	3.14	0.73
4. Consistency	3.94	0.87
5. Prevention	3.11	0.48
6. Emphasis	3.93	0.59
7. Flexibility	2	0.80
8. Aesthetics	3.25	0.43
9. Help to Users	3.2	0.53
10. Help with documentation	3.25	1.04
11. Easiness	2.25	1.12
12. Speed	4.2	0.49
13. Reliability	2.91	1.19
14. Classification	2.8	0.61

Figure 3 Strapi's UI evaluation results.

5 Conclusion and Future Work

This paper highlights the advancements made in the development of a UI with a good UX and useful functionalities for teachers to manage and create courses. One of the main contributions of this paper is the examination of whether a headless CMS, specifically Strapi, can adequately serve this purpose. The usability evaluation indicated it to be too generic and not flexible for effectively creating and managing programming courses.

Furthermore, the paper presents a promising approach to leverage the Agni student's UI and make it editable by having input fields instead of text boxes and adding missing functionalities, such as the addition of lessons for the teacher. The UI communicates with the customized API of Strapi to create and manage the contents of the data model.

In future work, features such as the reuse of exercises will be implemented. The possibility to import external contents, especially exercises, as well as the design and implementation of the class managing part, including the import and creation of students, will be finished. After that, a final evaluation of the UI will be conducted using the Strapi UI evaluation as a benchmark for comparison and evaluation of the progress.

References

1 Allam Hassan Allam, Ab Razak Che Hussin, and Halina Mohamed Dahlan. User experience: challenges and opportunities. In *Journal of Information Systems Research and Innovation 2013*, 2013.

2 D. Benyon. *Designing User Experience*. Pearson Educación, 2019. URL: https://books.google.pt/books?id=MXqFDwAAQBAJ.

3 John Dunlosky, Katherine A. Rawson, Elizabeth J. Marsh, Mitchell J. Nathan, and Daniel T. Willingham. Improving students' learning with effective learning techniques: Promising directions from cognitive and educational psychology. *Psychological Science in the Public Interest*, 14(1):4–58, 2013. URL: http://www.jstor.org/stable/23484712.

4 Marc Hassenzahl and Noam Tractinsky. User experience – A research agenda. *Behaviour & Information Technology*, 25(2):91–97, 2006. doi:10.1080/01449290500330331.

5 Bettina Laugwitz, Theo Held, and Martin Schrepp. Construction and evaluation of a user experience questionnaire. In *HCI and Usability for Education and Work*, volume 5298, pages 63–76, November 2008. doi:10.1007/978-3-540-89350-9_6.

6 Effie Lai-Chong Law, Virpi Roto, Marc Hassenzahl, Arnold P.O.S. Vermeeren, and Joke Kort. Understanding, scoping and defining user experience: A survey approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 719–728, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1518701.1518813.

7 Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pages 152–158, New York, NY, USA, 1994. Association for Computing Machinery. doi:10.1145/191666.191729.

8 Jakob Nielsen. Usability inspection methods. In *Conference Companion on Human Factors in Computing Systems*, CHI '94, pages 413–414, New York, NY, USA, 1994. Association for Computing Machinery. doi:10.1145/259963.260531.

9 Jakob Nielsen. Why you only need to test with 5 users, March 2000. URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

10 Ricardo Queiros and José Leal. Programming exercises evaluation systems – An interoperability survey. In *International Conference on Computer Supported Education*, volume 1, pages 83–90, January 2012.

11 Roshni Sabarinath and Choon Lang Gwendoline Quek. A case study investigating programming students' peer review of codes and their perceptions of the online learning environment. *Education and Information Technologies*, 25(5):3553–3575, September 2020. doi:10.1007/s10639-020-10111-9.

12 Zarina Shukur, Edmund Burke, and Eric Foxley. The automatic assessment of formal specification coursework. *Journal of Computing in Higher Education*, 11(1):86, 1999.

13 Leon E. Winslow. Programming pedagogy – A psychological overview. *SIGCSE Bull.*, 28(3):17–22, September 1996. doi:10.1145/234867.234872.