Using New Technologies to Learn Programming Languages

Dan Alexandru  
Faculty of Computer Science, Alexandru Ioan Cuza University  
Iasi, Romania  
dan.alexandru@info.uaic.ro

Adrian Iftene  
Faculty of Computer Science, Alexandru Ioan Cuza University  
Iasi, Romania  
adifene@info.uaic.ro

Daniela Gifu  
Faculty of Computer Science, Alexandru Ioan Cuza University, Institute of Computer Science, Romanian Academy, Iasi Branch  
Iasi, Romania  
daniela.gifu@info.uaic.ro

Abstract

Current eLearning systems are increasingly used by both students and professors, considering the various facilities they offer. In the field of computer science, these eLearning platforms need to provide integrated program editors with facilities for compiling and running them. We propose a creative architecture of an eLearning system for Python which comes with new facilities related to the possibility to create content (lessons, exercises, content, and tests) inside of this platform. Thus, the professors can fully prepare their lessons and homework on our CSP (computer science platform) platform via the web interface. Similarly, the students can access this content via the platform and can solve their homework in this special space. Depending on the number of users the allocated resources dynamically change in order to ensure the proper functioning of the application, trying to keep lower operative costs.

Keywords: eLearning systems, computer science platform, Python lessons, dynamic distribution.

1. Introduction

In recent years, more and more eLearning platforms have been introduced to help professors and students at school or at a distance, through modern technologies [14]. Having a significant role in higher education [3], eLearning platforms are very diverse and can be classified into the following categories: (1) memory-based (Memrise, Duolingo) [16], [22], (2) exercise-based/project-based (Codecademy, Dataquest - similar to DataCamp) [9], [24], (3) programming-based (Infoarena, HackerRank) [20], (4) visualization-based (VisuAlgo, CodinGame) [7], [29], (5) course-based (edX, Coursera, Udacity, Pluralsight) [2], [4], [6], [13], [27], and (6) mixed platform (multiple features) (Khan Academy, DataCamp) [28], [30]. The main research questions of this paper intend to answer: (1) How efficient is an eLearning system in order to study programming language (here, Python) or a programming technique? (2) How could the resources be dynamically allocated according to the number of users in order to ensure the proper functioning of the application, minimizing costs?

Our research proposes the architecture of an eLearning system for Python which comes with new facilities related to the possibility to create content (lessons, exercises, movies, and tests) inside of the proposed platform to demonstrate the effectiveness of a virtual
assisted educational approach.

The paper is structured as follow: Section 2 presents a short overview of eLearning systems in order to clarify their importance and what we can do to add new functionalities, while Section 3 refers to the architecture and the design structure of our eLearning system, named CSP (Computer Science Platform). Section 4 briefly discusses the evaluation of this platform before drawing some conclusions in the last section.

2. An Overview of eLearning Platforms

In order to present the specificity of our system, this section addresses the following question: what is a good eLearning system? Thus, we recall some of the most popular methods of virtual learning platforms.

2.1. Memory-Based Platforms: Flashcard Method

A key exercise in memorizing different information is to memorize the question and its answer using the opposite sides of a card. Some platforms were launched for learning languages - you associate a word with a translation. Thus, learning a mnemonic will help remembering the information. This process might have been used for a long time (intuitively or learned), but applications such as Memrise or Duolingo helped solidify this process and make it more accessible and less strenuous compared to simple flashcard learning. Memrise, created by Ed Cook and Greg Detre [8] is structured as a crowdsourced course site, most of the courses being made for learning various languages [25]. As such, you have various levels in a course (grouped by context), each level consisting of learning words and their respective translation. However, this system has several inherent weaknesses [22]: the base for learning languages is not structured enough, the computerized voice is not clear; the voice recognition for the learners to practice speaking is missing, etc.

2.2. Exercise-Based Platforms

In the past 10 years, online learning (simple or mixed) has become very popular and still increasing. For instance, Khan Academy created by Salma Khan in 2008 [18] offers online lessons on math, science, and other topics that have made a lot of sensation in the virtual environment.

Consolidation works through (spaced) repetition (different contexts, similar solutions) and through applied projects: (1) Codecademy has projects for various web technologies [24]; (2) Dataquest offers Guided Projects - having some instructions to follow, but that only hint what to analyze; (3) DataCamp employs a similar solution as Dataquest [9]; (4) Pluralsight or similar platforms for professionals who want to upskill [4]; MOOC-based platforms [19] as; (4) Udacity offers academic projects upload, without the context where you code that project like in the Dataquest case; (5) edX and (6) Coursera, both are almost exclusively academic courses, offering certifications in various domains where a student might need a portfolio (e.g. Android Development, Deep Learning).

The assets of technology-supported learning and education, using these digital systems and services are obvious: they are practical - exercises actually develop the proposed skills; consolidation - to avoid the danger of memorizing solutions (rote memorization, applying Stack Overflow solutions without understanding/reshaping/improving them); controlled failure - the course creators might put bigger challenges or intended mistakes throughout the course in order to make students comfortable with temporary setbacks; thorough repetition - persistent and consistent incremental steps outside the comfort zone yield mastery. These key drivers contribute to the cyber dialogue between learners, professors, researchers, practitioners and technologists to improve the evolution of Open Access to Formal and Informal Learning [23]. In time, the developers have extended the eLearning functionalities, implementing mixed platforms that combine a mixture of facilitation traits (e.g. Khan Academy and DataCamp): courses, code assignments, streaking and minimal
tracking of class assignments, etc. Even though the tendency in learning is towards eLearning, the great disadvantage is that we obtain knowledge only on a theoretical basis. When it comes to putting to use whatever you have learned, it may be a little difficult. This may come in conflict with the applicable part, especially for the online assessments.

2.3. Competitive Programming Platforms

Most platforms under this category have a large variety of problems to solve, and are not updated in terms of languages of use. For instance, in order to better understand the behavior of software developers, IDE (Integrated Development Environments) could be a good rewriting and visualization system [11]. In addition, explanations for algorithms or original solutions are scarce (due in part to small population and fear of memorization of solutions, which could set an uneven playfield for contests) or rarely encouraged through solution sharing (after some form of locking that problem solution, i.e. not being able to edit it further). In this context, most programming platforms are very similar in interface and functionality, slightly different in problem sets, with most of them focused on training students for ACM (or similar) competitions. In this context, examples such as HackerRank [5] (focused on computer science topics like math, algorithms, artificial intelligence, etc.) and CodinGame [7] (a way to improve the programming skills by playing) are becoming relevant, both with distinct feel and particular features. Though, there are some limiting frame factors as the number of students in a class or the available online resources.

2.4. Visualization Platforms

There are few examples of such platforms due to this fundamental problem: good interactive visualizations take time to be made and few people know the technologies to do so. Amongst these few, VisuAlgo stands out by having visualizations for the most frequent data structures and algorithms. Alternatives: Galles algorithms visualizations [12] or some resources from the enjalot/algovis repo [17].

2.5. eLearning Platforms for Computer Science

Usually, eLearning platforms provide professors and students access to a platform to manage students’ courses and themes (in the form of HTML or PDF or DOC, or TXT or as multimedia content). They can also create space where hours can be done by putting in contact professors with their students, allowing interaction between them, either by e-mail or chat, allowing evaluation based on grid tests, offering graphs with students’ evolution, etc. In computer science, when students learn a new programming technique or new programming language, they need to write programs they can evaluate either on their own or under the supervision of the professor. For this reason, platforms that put together the elements of an eLearning system, with the specific elements of a development and programming environment, are highly appreciated by professors and students from computer science faculties. Next, we will see some eLearning platforms designed for computer science students. In the article [10], e-learning technology opens up new possibilities for improving and enhancing the quality of creative work, accessibility to learning, communication and learning content by creating a virtual interactive learning tool of the Java programming language. Another platform for Java, the Mag learning system [26] has three student facilities: tutorial activities, feedback grid questions, and online programming. Mag also supports example-based learning and programming exercises. In [21], the authors present an eLearning platform designed specifically for learners of C language. The system has five basic modules: (1) for course content, (2) testing, (3) exercises, (4) questions and answers, and (5) help.

The platforms presented were successfully accepted by those who tested them, and the assessment of the students who used them demonstrated their usefulness. A drawback of these platforms comes from the fact that systems lessons are uploaded many times as Pdf or Doc, and they cannot edit the lesson content and update it when needed. Another aspect which was not presented in these papers is related to the scalability of the systems. What
is happening if at the same time hundreds or thousands of students will try to simultaneously use the platforms? Our solution comes to solve these issues, offering a solution for Python language which integrates an editor for lessons and a mechanism to scale into Dockers when this is needed in order to assure the scalability of the system.

3. Methodology

The CSP system aims to provide a middle ground for professors and students, in order to find an integral and common approach in Computer Science education. We accomplish this by providing a modular microservice-oriented architecture that combines features and integrations from multiple projects such as Node.js, React, Python, NVidia Docker, and Jupyter, among others. Through this platform, we enable workflows as: presenting reveal.js or Jupyter RISE slides, edit pages using Markdown, create ephemeral Docker containers for labs, and enable persistent deep-learning ready Docker containers.

3.1. System Architecture

CSP system (Fig. 1) has been built with the following constraints in mind: (1) as fewer configurations on the host as possible (e.g. minimal drivers, for example, NVidia drivers for nvidia-docker), Docker config, monitoring); (2) deployment should be isolated (as fewer external dependencies as possible after initial setup, package managers, and Docker registry should be self-sufficient), for example, local Docker registry, caching of package managers; (3) possibility to allocate resources in a dynamic way using a predictive model in order to minimize the costs.

Fig. 1. CSP architecture.

Standard workflow is: user accesses the React client (deployed on a Node server), which calls the Node server for authorization or minimal services, which in turn will call the Python server (which should do the most heavy-lifting to avoid coupling above). The Python server will create ephemeral Dockers when needed.

Per the constraints above, every module in the platform is dockerized (i.e. module is built and run inside a container), a local Docker registry for that VM/machine providing the needed images after the initial setup. It is recommended to run the compute-heavy or ephemeral Docker containers on a distinct VM/machine, such that the main application and the correlated Dockers are not blocked. In our experiments (Fig. 2 left), the containers on that machine were orchestrated by k8s (Kubernetes) and Docker Swarm.
During the development of this platform, most features have been first implemented separately as modules (written in pure JavaScript) and had been afterward integrated into the React application. As such, this application favored the use of microservices and modularization from the start.

**Markdown Editor Tool**: For course pages, editing documentation needed a Markdown Editor. By combining the Ace Editor and Showdown parser we got the result from fig. 2 right.

**Reveal Importer Tool**: For presentations made on slides.com, importing them and especially versioning them is a hassle (injected styling and scripts in a page, themes used are not public, even though reveal.js is open-source, etc.). We have made an importer tool which parses the resulting files from slides.com and you can compare the original file to the simplified file and check for visual differences (both loaded in iframes, events in the tool window are captured and sent to the children iframes) (see Fig. 3).

**Modularization**: Every module of the application (the clients and servers, databases and tooling Dockers) has been dockerized, and Dockers are also available for separate modules (Fig. 4). As such, the solution consists of having an online platform, an easy on-premise deployment with Dockers and packaging of separate tools for offline work.

**Jupyter Notebooks on NVidia-Docker**: A persistent limitation of Docker was that NVidia cards could not be used inside a container. So anything related to HPC (High-Performance Computing) that would be enabled by CUDA (amongst other technologies) or deep learning would not be possible.

![Fig. 2. Orchestration of modules (left) and CSP editor (right).](image1)

![Fig. 3. CSP serializer/importer.](image2)
Through the use of CUDA and CuDNN-enabled Ubuntu images, we can accomplish this with a solution that (Fig. 5 left):

- Gracefully avoid vendor lock-in by using open-source technologies - Docker, Jupyter and configurations for it and various pip packages.
- It has most python dependencies needed for linear algebra (numpy, scipy), visualization (matplotlib, seaborn), machine learning (scikit-learn), deep learning (keras, tensorflow, pytorch)
- It is cheap to deploy - built Docker image has 3.6GB, approximately one of a standard VM (virtual machine).

As stated before, Jupyter is capable of integrating various workflows (Fig. 5 right) and can be used for various learning activities: presentations, visualizations, learning documents, analysis.

### 3.2. CSP Functionalities

We have described some common use cases for the platform and what workflows are enabled by it (including future workflows).

**Course pages**: Users can edit and display markdown pages on CS platform. Multiple alternatives are given: upload manual or statically generated HTML site, edit markdown real-time.

**Course review and memorizing aids**: Due to the AutoCourse Builder feature, course pages will feature learning materials from professors and students, summaries, quick quizzes and mind maps.

**Lessons/Presentations**: Users can build lessons or presentations with multiple tools: create and edit on slides.com and import to CSP platform, upload or edit on CSP platform
(from slides.com or pure reveal.js), edit and present from a minimal Docker container with Jupyter and RISE

**Labs**: Labs can be created through either: pick Docker container that you want to run (new one or already-created one), VNC or openssh to a desired container (be mindful that it must be exposed to the internet).

**Teaching labs** can be created through either: the professor using and presenting from a demo container, local and then deployed through webhooks to katacoda, which will then be embedded in the platform, local and then deployed through webhooks directly to CSP platform, edit on CSP platform and serve lab from it.

**Projects** can be submitted by students on selected pages. The workflow is as follows: upload on drop zone page (project will be saved both as zip and unzipped form to minio), the platform will display the project for both student and professor, and the platform will check the project for any irregularities.

**Project statistics**: Concurrently with the above workflow, on the upload of project, the git microservices will analyze the git metadata of the project.

**Distance Learning** is enabled by the following platform features: AutoCourse Builder - additional resources for any subject by automatic classification and ranking, reveal.js presentations - access to both professor and student presentations for any subject (provided the crowdsourcing condition is encouraged at that specific course), Docker playgrounds, Docker labs VNC access to both AWS instances or on-campus computers or local machine (if away from home).

### 3.3. CSP Scalability

In the current cloud landscape, we identified in the CSP platform a need for a robust FaaS (function as a service) for Python that would aid both developers and researchers in their Python workflows. Thus, we leverage existing cloud infrastructure to allow flexible workflows both over existing FaaS and IaaS (infrastructure as a service) solutions. We propose a solution, leaning towards a Service-Oriented Architecture (SOA). In Fig. 6, all components are atomic, containerized, and easy to swap and to upgrade.

![Fig. 6. A solution leaning towards a SOA proposer.](image)

As it is, a client (professor or a student) would connect to the CSP platform that provides the service where they could access functions (add or edit or read or solve their lessons or homework), afterward running in a one or multiple hosts, during which logs will be displayed for every active client. The anonymized metadata from the ephemeral containers and the functions that are deployed is stored, queried and displayed in the ELK stack (Elastic-search, Logstash, Kibana - Logstash stores, Elasticsearch indexes and
exposes an API, Kibana displays metrics). Furthermore, Elasticsearch can be queried in analysis ephemeral instances (in AWS context - spot instances) which host a Python analysis container for easier visualization, and use of data. Eventually, that spot should offer the possibility to create metrics, alarms, automatically-updated notebooks or models for future improvements of the models used by the predictive microservice.

The user interface (UI) module is composed by ELB (elastic load balancing) routing to multiple machines, EC2 (Amazon Elastic Compute Cloud) hosting of our Kubeless UI, implementation of the Kubeless UI/custom frontend on top of it. The Kubernetes backend is composed of ELB routing to the k8s (kubernetes) backend, Kubeless deployment of k8s, kubeless controller, PyFaaS controller (our custom k8s controller for prediction). The metrics aggregation module is composed of Prometheus pulling of the metrics from k8s, metrics pushing to ELK (ELastic Stack) stack, exposing Prometheus/ELK to the analysis instances. The Analytics module is composed of on-demand open of the instance (Jenkins job/AWS interface), analysis spot instance, Docker analysis image (stack-deep from CSP platform), and pushing notebooks from the analysis spot to nbviewer. The predictive module is composed of predictive models from the analysis spot are backed up either Pickle or ONNX format on S3, trigger notifies the Predictive microservice that new models are available, predictive microservice is exposed to the PyFaaS k8s controller.

3.4. Used Technologies

**React** is an open-source MV* (model-view) library made by Facebook. Advantages for using React are: very small overhead (faster render and update), tree-based update, reusable components, and Chrome DevTools support. React has been used because of its very small overhead and very high extensibility.

**Jupyter** is a web notebook that enables making documents that combine code, markdown, and visualizations and makes prototyping and learning faster. Advantages: being a web notebook, you can easily dockerize it and deploy it on an AWS server and connect to it remotely, supports multiple “kernels”, native plugin support, support for parallel computing, easy to share.

**Node.js** is a JavaScript runtime built on Chrome V8 JavaScript Engine. Advantages for using Node are: a lot of scaffolding available, fast prototyping, almost all web-related technologies are easily integrable with Node.js or have a compatible bridge/client for Node.js, async by design.

**Python** is an interpreted high-level programming language that is designed for scripting and fast application development, for diverse use cases. Some of the advantages of Python: simple syntax, the reference implementation is CPython (written in C) and very good integration with C/C++ allows Python to use for a lot of its libraries, low-level implementations, usually C and FORTRAN, libraries for various workflows. In this platform, Python has been used as much as possible for extensible, easy to maintain microservices.

**Docker** is a container platform written in Go, that allows for extensible, maintainable, distributable containers. Advantages: easy deployment, lightweight, ephemeral use and easily disposable, isolation of OS-level dependencies. Docker has been used extensively for every tool, server or client.

**Amazon Web Services (AWS)** is one of the most popular cloud providers. Advantages: small starting costs, no on premise servers costs and maintenance, no upfront charge, fundamental cloud tenets such as “pay as you go” are respected, one of the smallest prices for compute especially with spot instances, various type of instances for any use, few quota locks that are disabled with use/time.

4. Server Load Prediction

Load balancing was a constant concern ever since the web started expanding. Too little compute power, and a site become slow and unresponsive, losing users and thus revenue; too much, negatively impacting profit. This created the need for load balancing. However,
adding more compute power after the users’ performance has already been impacted, especially since adding more compute power is often not very fast, only leads to limited gains. In the current market, where employing machine learning techniques becomes more and more practical, it comes as a natural extension to load balancing: replacing reaction with proaction.

Due to the volatile nature of web traffic and lack of massive datasets, research in the field has been inconsistent. However, this exact advent has been growing more and more in the field of electrical load balancing [15], for energy suppliers. Research into using convolutional recurrent deep neural networks has yielded positive results in forecasting load on a per week basis, forecasting loads for each day of the week. For example, higher loads in the middle of the day, lower loads during the nights when people usually sleep, and the same pattern during the weekends, but with slightly elevated levels.

Less prevalent, but still present, is an area of study more related to the present project: server load prediction. This field also has employed neural networks to achieve load prediction [1]. Although this application is closer to the current goal, it is still something different as server load is a different matter from a theoretically infinite cluster of computing spots that can simplify scale up power and costs until the stipulated demands are met. Nonetheless, the core of this project is achieving prediction on a time series. While the applications of such an endeavor vary wildly, the core ideas of the field are completely portable.

The problem of load prediction is essentially the prediction of a time series. A time series has different intervals of periodicity, as depending on the service being offered, traffic can vary from day to day - spikes in the morning for a news site, from week to week - more traffic on the weekend for an online game, from month to month - more traffic at the start of the month for a bank, when salaries come in, from year to year - for occasions such as Black Friday or the holiday season - or as a general lifetime trend such as the presence of the aforementioned patterns, but the general traffic being increased because of the growth in popularity of the service. The proposed solution took inspiration from coherence-based neural networks used for music generation that use multiple levels of “history” to ensure coherence in the generated material both in the short and long term.

The present solution uses extreme gradient boosting (xgb) as the primary model, and five such models are employed. The training data is then preprocessed such that the first xgb model only receives the hour and minute (time of day) of a data point, aiming to identify patterns in the scope of one day, the second receives, in addition to the time of day, the day of the week, trying to identify patterns in the scope of one week the third receives the time of day and day of month, identifying patterns during the course of a month, the fourth receives the same input data as the “month” model with the addition of the current month, widening its scope to one year, the final model receives the full time and date. The models are then combined by evolving a combiner function as an expression tree using genetic programming. Using R2 as the statistical measure for the accuracy, using the genetic programming combination we can improve its value by anywhere from 0.3% to 10%.

The setup refreshes every 5 minutes to a new state; this is the minimum reaction time for the predicted load. We do not interfere with traditional scaling model (based on thresholds or scheduled scaling) and our method offers better cost optimization. For the current solution, it takes on average 2 minutes for the predicted number of containers to be created, but this can be drastically being improved if the underlying hardware is improved. The setup can respawn a container in less than 10 seconds in case of a crush and can migrate in less than 1 minute if a node fails.

### 5. Comparisons between 2018 and 2019

In our experiments, for Python course, we compare the results from the previous year (2018) with results from this year (2019), when the students and professors used the CSP platform. If in 2018, they use a platform only for final examination, in 2019, they use the CSP platform during the labs, the homework assessments, the preparation for the final
exam and the examination. In 2018, 488 students participated at Python exam, while in 2019 475 students participated.

**Positive effects:**
- While in 2018 the percentage of students which promote the exam was 86.50, in 2019 it was 92.74. This thing is related to the fact that the majority of the students (close to 100%) used the platform before the final exam, and they were familiar with it.
- In comparison to 2018, when a maximum of 50 students was evaluated at the same time in order not to block the server (evaluation lasting over 10 hours), in 2019, even more than 100 students could attend at the same time (evaluation lasting less than 5 hours). In 2019, the limitation was due primarily to the number of computers in the faculty’s laboratories.
- On the evening before the final exam, nearly 400 students used the CSP system at the same time, and it worked smoothly. Thus, we could test the robustness of the system, and the results were encouraging.

**Negative effects:**
- Because the students receive feedback in real-time and know when they are promoted, many of them stopped without going to the heaviest final tests that would allow them to get grades 9 and 10. In the end, 60% of them receive grades between 5 and 7 and 21.68 receive grades over 8. In comparison, in 2018, when they did not know in real time what they got to the current test, around 40% of them receive grades over 8.
- The professors and their collaborators involved in the Python course, have created content for the CSP platform besides their usual activities to. The advantage will come in the coming years when they can reuse many of the resources now created on the CSP platform.

At the end of the semester, students evaluate the professors’ activities, using an online platform where they complete a questionnaire with two parts. First part contains 20 grid questions with answers predefined (from insufficient to excellent). The second part allows including optional comments area (positive and negative opinions). At Python course was one professor at course and 6 PhD students at labs. 144 students have evaluated the course activity (from which 62 write comments) and 102 have evaluated the lab activity (43 write comments). Below, we present the students’ and professor’ opinions about the CSP platform.

**Students’ opinions:**
- The system is very useful (especially as the part of exercises can be called from a distance and helps to prepare the final evaluation).
- Those who dropped the exam last year noted that this year things went much better and that the evaluation lasted less.
- The students reported that more training problems would help them even more.
- The issues raised by them were related to the moments when new resources were allocated at the level of Dockers, problems encountered especially during the exam preparation when there was a delay (about 2 minutes) needed to request and negotiate new resources in the cloud. During the exam, these issues were not met because we set up 120 Dockers in the context in which those who connected to the evaluation component were not more than 100.

**Professors’ opinions:**
- It is very useful that all materials are on the platform and can be reused very easily.
- The fact that they have seen what issues are not solved by students, during the assessment. Furthermore, this helped them to prepare weekly courses and laboratories.
- They were delighted that the platform is dealing with the allocation of resources
when it needs to ensure scalability.

• The final evaluation that took less time made them want to use this platform in the following years.

6. Conclusions

E-learning has a great potential in higher education, involving the training, delivery of knowledge, and motivates both learners and professors to interact with each other, anytime and anywhere. The work we have done with CSP platform tries to enable workflows and teaching methods that have been in past either: heterogeneous - across multiple platforms/solutions, impossible - especially the case provided there was no Docker, Jupyter, NVidia-Docker, expensive - using the AWS Deep Learning AMI requires 80+ GB of EBS (file storage) compared to our solution which requires 4 GB and is highly configurable and cheap. We consider that through these reusable components and their composition and orchestration, we can create a full-fledged solution. Also, there is no open source project accepted in Kubernetes/Cloud Native Computing Incubator that handles predictions (not even on pods). There are commercial solutions offered by AWS/Google Cloud. With more refactoring and better integration with Kubernetes, we can propose the project to the Cloud Native Interactive Landscape Incubator.

What is important is that our proposed solution has been positively appreciated by both academics and students. Moreover, after a year of use, the CSP platform has helped to provide a space where students can work on laboratory topics and can prepare for session evaluations, helping to familiarize them with a new programming language (in this survey, Python).

Acknowledgments

This work is partially supported by POC-A1-A1.2.3-G-2015 program, as part of the PrivateSky project (P_40_371/13/01.09.2016).

References