# DeployGround: A Framework for Streamlined Programming from API Playgrounds to Application Deployment

Jun Kato, Masataka Goto
*National Institute of Advanced Industrial Science and Technology (AIST)*, Tsukuba, Japan, {jun.kato, m.goto}@aist.go.jp

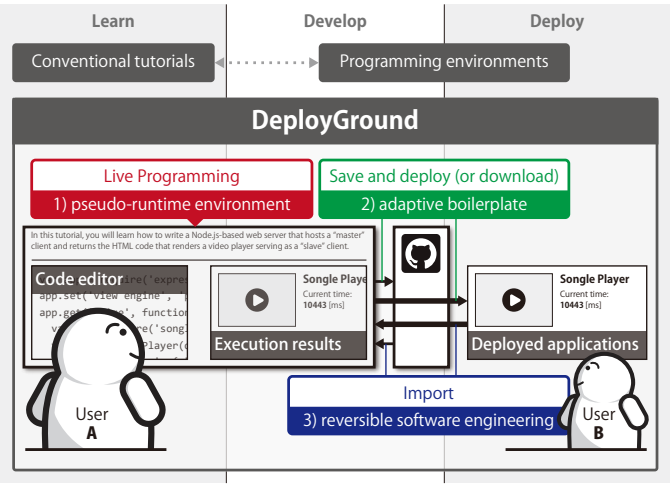


Fig. 1. The DeployGround framework features 1) a pseudo-runtime environment, 2) an adaptive boilerplate, and 3) a reversible software engineering feature for interactive coding tutorials, which altogether streamlines learning APIs on playgrounds and developing and deploying applications.

***Abstract*—Interactive web pages for learning programming languages and application programming interfaces (APIs), called *"playgrounds,"* allow programmers to run and edit example codes in place. Despite the benefits of this live programming experience, programmers need to leave the playground at some point and restart the development from scratch in their own programming environments. This paper proposes *"DeployGround,"* a framework for creating web-based tutorials that streamlines learning APIs on playgrounds and developing and deploying applications. As a case study, we created a web-based tutorial for browser-based and Node.js-based JavaScript APIs. A preliminary user study found appreciation of the streamlined and social workflow of the DeployGround framework.**

***Index Terms*—Coding tutorials; online learning; API playground; live programming; programming experience**

## I. Introduction

It is not easy for programmers to learn new programming languages and application programming interfaces (APIs). Prior research has extensively investigated how to design learnable languages [1] and APIs [2], but only recently has the research community started to discuss the effectiveness of the online learning resources for coding [3], such as interactive tutorials, web references, massive open online courses (MOOCs), educational games, and creative platforms. Web references and MOOCs courses such as API documentations and step-by-step introductions are often provided in *read-only* formats, in that they consist of text and optional multimedia content, such as images of input and output data and screen recordings of programming environments. To try out the tutorial content, learners need to switch back and forth between the tutorial and their programming environments.

Recent web-based tutorials avoid this frequent context switching by incorporating code editors into web pages, allowing the learners to practice *live programming* with the language or API without installing anything on their computers. A set of such features is often called a *"playground,"* because it constitutes a sandboxed environment in which the learners can play with the target language or libraries (e.g., Khan Academy [4], TypeScript [5], and Vimeo API [6]).

Although the *playground* approach has significant advantages over the conventional *read-only* tutorial, programmers developing applications need to leave the web-based playgrounds and restart the development in their own programming environments. This tedious transition is usually handled by the learners and is not supported by the tutorials. This paper proposes DeployGround, a framework for creating web-based tutorials that streamlines learning APIs on playgrounds and developing and deploying applications (**Figure 1**).

## II. Related Work

This section introduces prior work on web-based coding tutorials. More thorough reviews of the related work including research on executable documents [7], [8] and live programming [9]–[13] can be found on the web[1].

While there is much work on creating tutorials for various purposes [14]–[17], there is only a handful of work specialized in creating interactive coding tutorials. Harms et al. explored automatic generation of interactive step-by-step tutorials by transforming each sentence of example codes into a step [18]. Tutoron [19] allows one to write micro-explanations of code and allows learners to read them automatically inserted next to example code on the web. Codepourri [20] allows annotation of the history of program executions through which learners can navigate to learn the program behavior. Our work does not provide tools for creating a new kind of tutorials as these



[1]DeployGround website. https://junkato.jp/deployground

do but instead presents a framework that addresses limitations of existing web-based coding tutorials for learning APIs.

As discussed in the introduction, many online tutorials present *read-only* content that programmers can read, watch, and sometimes discuss with other learners but cannot interactively run and edit. However, there is an increasing number of interactive coding tutorials that provide code editors with which programmers can run and edit example code. In terms of the implementation, they can be roughly divided into three categories (**Figure 2**).

The first category (**Figure 2** (1)) is for learning client-side web technologies such as HTML/JavaScript/CSS and utilizes the JavaScript `eval()` function and/or HTML5 sandboxed inline frames (`Iframe`). For instance, W3Schools [21] provide a JavaScript code editor next to the preview pane, in which the code gets executed in the `Iframe`. The `eval()` and `Iframe` implementations are very simple and provide quick response to the user edits, but both are vulnerable to malicious code that can crash the browser (such as infinite loops). Furthermore, this category cannot handle programming languages the browser cannot interpret.

The second category (**Figure 2** (2)) is for learning how to use the character-based user interface (CUI) and how to build CUI programs. It provides each user access to a lightweight virtual machine (VM) on the server, such as a Docker container. For instance, the C programming course in Tutorials Point [22] provides access to the GCC compiler and allows the user to compile and run the program. Codecademy [23] provides access to a console of a Linux-based VM and allows programmers to test CUI commands. npm [24], the package repository for Node.js libraries, allows the user to test libraries within the browser. Although this approach is flexible and can safely run any code, it is usually slow because of its high computational cost and the latency between the server and client. To make matters worse, all visitors to the web pages need to share the computing resources, which are usually limited owing to the running cost, resulting in even slower responses.

Our work and many live programming environments on the web fall in the third category (**Figure 2** (3)), in which the user code gets executed in an interpreter. The interpreter is implemented in JavaScript and runs on a web browser. This approach is slightly slower than the `Iframe` method because of the interpreter overhead but significantly faster than the VM-based method because everything runs on the client computer without network or VM overhead. Because the user code is always executed under the supervision of the host interpreter, malicious code can be detected in a practical manner, and it is much safer than the `eval()` and `Iframe` methods. The execution is often more controllable than that in the VM method because there is no black box in the code execution process. Our work implements a *pseudo-runtime environment* that emulates the behavior of a server machine or a microcontroller with a thin interpreter layer and wrapped APIs of a fixed set of libraries.

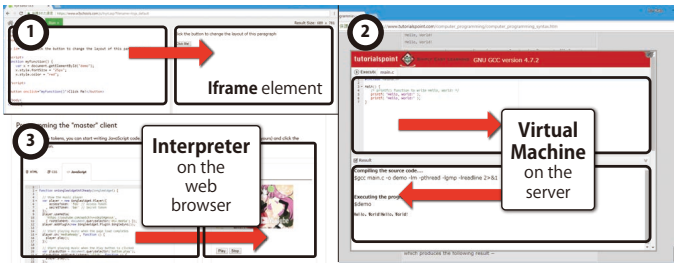


Fig. 2. Three implementation-based categories of interactive coding tutorials.

## III. Brief Review of Existing Tutorials

In this section, we use a JavaScript API called "Songle Sync API [25]" as a representative example of modern APIs and briefly introduce the *previous* version of its web-based tutorial. Then, based on the review and additional analysis of other popular tutorials, we identify four limitations of the existing interactive coding tutorials, each of which contributed to the design of the DeployGround framework.

### A. Representative Example API: Songle Sync API

The Songle Sync API allows hundreds of devices to play visual and physical computing performance synchronized with music playback (**Figure 3**). We chose it as a representative example of modern APIs for the following reasons.

- It is provided for JavaScript, which according to the annual report from the social coding platform GitHub was the most popular programming language in 2017 [26].
- Its has been actively developed since its initial release in August 2017.
- It supports both web browsers and physical devices such as the Raspberry Pi [27], reflecting the diverse application domain of modern APIs.
- It involves multiple (sometimes >100) clients over the Internet with real-time communication and handles complex data, making its behavior a non-trivial example of API behavior.
- It provides a large number of methods and properties (73 as of April 2018).

### B. Songle Sync API Tutorial

The previous version of the Songle Sync API tutorial links to the API documentation and provides step-by-step explanations of the concepts used in the API. In the later steps, programmers can not only read but also modify the

Fig. 3. Example applications made with Songle Sync API [25]—a web browser-based one (left) and Node.js-based ones (right; actuator modules controlled by Raspberry Pi devices).

example code and try calling the API within the web page that executes code in an `Iframe` element. This is a typical tutorial implementation in the second category (**Figure 2** (1)), which can be used to prototype a single HTML page containing HTML/JavaScript/CSS code.

### C. Limitations of Existing Tutorials

*1) Ephemeral Code:* Example code can be modified, but the modified code is ephemeral. Once the programmer leaves the tutorial, it is gone. The transience of the code prevents learners from continuously growing their codebases and gaining ownership of the code they edit.

Existing tutorials, such as W3Schools, allow one to download the code, but the downloaded code cannot be imported again. DS.js [10] allows the code to be stored in the query parameter of the URL but limits the size of the stored code. Codecademy tutorials and other tutorials that provide a VM instance to each user can keep the session between tutorial steps, but the session cannot be exported to nor imported from a local machine.

*2) Toy Sandbox or Expensive Sandbox:* Existing web-based tutorials tend to suffer from the issues related to the sandbox on which the user code runs. For instance, consider providing a tutorial for building Node.js-based applications. Tutorials simply utilizing `IFrame` or `eval()` do not allow the programmer to edit and test the JavaScript code for the Node.js environment.

Tutorials that use virtual machines, in contrast, can theoretically host the Node.js-based applications. But, running VM instances is computationally (and thus financially) expensive, so many tutorial creators would be unable to provide sufficient computational resources for fluid programming experience. In addition, it is difficult to gain meaningful debugging information when using a virtual machine. Furthermore, there is no way to emulate physical computing devices such as a Raspberry Pi device with a blinking LED.

*3) No Support for Deployment nor Social Interaction:* With many existing web-based tutorials, the learner can download the edited code as a single HTML file. Although the downloaded file can be loaded into a web browser, recent web browsers prohibit executing JavaScript in local files to prevent security risks. There are usually no instructions on how to deploy the code to the HTTP server. Deploying server-side code such as a Node.js-based project is more complex, but typical API tutorials only show text-based instructions or point to external resources that explain how to set up servers.

In addition, the learner needs to learn the content alone. The authors of the tutorial provide example code and nothing more. There is no platform support to collect all of the variations created by previous learners, which could potentially serve as new tutorial content for new learners. Nor is there any way to connect with other learners, who could help the learner with respect to the tutorial content. Social interactions in online learning have been extensively studied in the context of MOOCs as in [28] and [29], but there is not much prior research on how to augment API tutorials with social features.

## IV. DeployGround Framework

We propose the DeployGround framework (**Figure 1**), which addresses the limitations discussed above by revising the interaction design of the existing tutorials. This section provides an overview of the revised Songle Sync API tutorial and explains the key features of the tutorial's framework.

The revised tutorial website (**Figure 4**) allows the learners to play with APIs, those for prototyping HTML/JavaScript/CSS applications as well as Node.js applications, save project files in GitHub, and deploy the files to public web servers—all without leaving the tutorial website. Additionally, its social features help the user learn from concrete examples.

### A. Framework that Covers All Tutorial Steps

The framework provides a *unified workspace* throughout all of the tutorial steps—each code editor in the steps corresponds to a different file in the workspace, and each file can load other files with the `require` function, whose implementation is provided by the *pseudo-runtime environment*. We borrow the concept of the workspace from integrated development environments (IDEs), and in terms of implementation, the tutorials in the DeployGround framework are built on top of the web-based IDE.

With this support for a continuous session throughout the tutorial, we expect the ephemeral code to become permanent, written incrementally by programmers confident of their progress. Unlike the previous version that provided each step almost independently, the revised version makes all steps relevant to each other. For instance, the previous version forced the learner to input string tokens for the API calls in each step, but the revised version makes it possible to create a JavaScript file that is shared among all steps.

### B. Pseudo-Runtime Environment

The framework implements a *pseudo-runtime environment* that enables quick execution and debugging of the code written for the deployment target—in the case of the Songle Sync API, a Node.js environment. It is written in a client-side native language (JavaScript for web browsers) and interprets the target language (JavaScript for the Node.js runtime) with

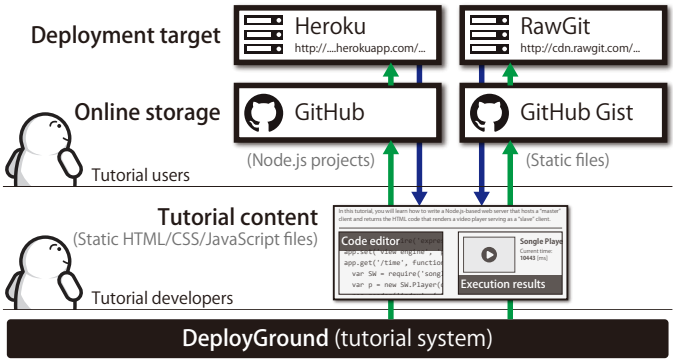


Fig. 4. Overview of the tutorial system implementation, on which tutorial content such as the Songle Sync API tutorial [25] is built.

partial support for the APIs of the default libraries (Node.js libraries such as `fs` for loading local files and `require` for loading `npm` packages).

In the revised tutorial, an emulated web browser or a figure of the Raspberry Pi device is shown next to the editor, both of which render the responses produced by the user code. When the programmer interacts with the emulated browser, the pseudo-runtime environment handles requests to the browser by emulating the execution of the Node.js code. Although the emulation is not perfect, it returns responses almost instantly because there is no network latency and the emulation layer is drastically thinner than that of a VM-based method. We expect the emulation to satisfy the needs of learners quickly experimenting with example code. When unsupported APIs are called, the tutorial shows error messages and a link to the supported APIs. It also shows typical errors such as execution timeout without freezing the browser.

### C. Adaptive Boilerplate

When the programmer wants to leave the tutorial and continue the application development, the user code in the tutorial cannot be naively executed in the programmer's environment. Because the framework is in charge of emulating the deployment target, it is aware of the transformation that wraps the user code with some boilerplate. For instance, `package.json` is needed for a Node.js project.

With this support for the adaptive boilerplate, the revised tutorial provides next to every code editor a `download` button that allows the user to download an archive file containing the transformed user code, boilerplate files, and a text file with instructions on how to install an IDE and the Node.js binary and how to run a command (`npm install`) that installs dependent Node.js libraries.

Furthermore, next to the download button is a `deploy` button that deploys the relevant files to the target server. Currently, static files such as HTML/JavaScript/CSS files are saved on a GitHub Gist [30] server and served through its unofficial content delivery service called RawGit [31], and Node.js project files are saved as a GitHub repository and deployed to a PaaS provider called Heroku [32]. After the deployment, the programmer can use a web-based IDE such as Cloud9 [33] to continue the application development.

### D. Reversible Software Engineering

The framework facilitates social interactions between learners who visit the tutorials. It utilizes a social coding platform, GitHub, to store the user code. Although it is usually difficult to reverse-engineer deployed web applications, applications developed within the framework can be made *reversible* by design. We call this *reversible software engineering*.

The *adaptive boilerplate* feature in the revised tutorial not only adds the ordinary boilerplate code but also hyperlinks to the tutorial page. By following the hyperlinks, the programmer can start the tutorial from scratch. Additionally, the programmer can optionally import the corresponding GitHub Gist or GitHub repository data into the tutorial. During the loading process, the project importer strips the boilerplate added by the exporter. The deployed applications thus become a new set of examples from which future learners can benefit.

## V. Preliminary User Feedback

As a preliminary study to gain initial qualitative user feedback, we asked three professional software engineers, two computer science researchers, and twenty-four university students to use the revised Songle Sync API tutorial. We asked the professional engineers and researchers to compare their experience in this use with their prior experience using web-based tutorials, and we asked the students to spend two days using the tutorial and developing applications.

All the participants appreciated the streamlined experience from the playground to deployment. While ordinary web-based tutorials are targeted to a single developer, we observed the university students instantly sharing and boasting about their developed applications, supporting the social aspect of our framework. Other representative insights relevant to future work are discussed below.

*1) Potential Applications:* While the DeployGround framework has been tested for sandboxing only a web server and Internet of Things devices, there were enthusiastic expectations regarding its potential applications. For instance, the participants requested interactive tutorials for development frameworks for iOS and Android devices and for APIs for machine learning applications.

These expectations stemmed from the high initial cost of trying out the frameworks and APIs. In particular, installing and uninstalling a framework, preparing not only a server but also datasets for testing APIs, and looking for a variety of example codes are tedious.

*2) Demands for Architectural Visualizations:* A recurring request from the participants was for more explicit visualization of the workflow supported by the DeployGround framework. While the automated project export and import processes were considered extremely helpful, the participants wanted to know more about what is happening behind the scene. In particular, those who did not know the concept of PaaS wanted to see a figure like **Figure 4**, which shows the relationships between the tutorial, GitHub, and Heroku.

*3) Limitation and Potential Extension:* Given that our approach emulates the target rather than hosting it, there is an inherent limitation that was observed during the user study. For instance, there were complaints about convenient but unsupported APIs. We are aware of such APIs and clearly state in the tutorial that further developments should be done on a web-based integrated development environment, the transition to which should be very smooth thanks to the project exporter feature. Future work should also be done on instantly notifying the users of unsupported APIs—e.g., showing errors when unsupported APIs are typed in the code editor.

## Acknowledgment

This work was supported in part by JST ACCEL Grant Number JPMJAC1602, Japan.

## REFERENCES


[1] A. Stefik, S. Hanenberg, M. McKenney, A. Andrews, S. K. Yellanki, and S. Siebert, "What is the Foundation of Evidence of Human Factors Decisions in Language Design? An Empirical Study on Programming Language Workshops," in *Proceedings of the 22nd International Conference on Program Comprehension*, ser. ICPC '14. New York, NY, USA: ACM, 2014, pp. 223–231. [Online]. Available: http://doi.acm.org/10.1145/2597008.2597154

[2] M. P. Robillard, "What Makes APIs Hard to Learn? Answers from Developers," *IEEE Softw.*, vol. 26, no. 6, pp. 27–34, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1109/MS.2009.193

[3] A. S. Kim and A. J. Ko, "A Pedagogical Analysis of Online Coding Tutorials," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: ACM, 2017, pp. 321–326. [Online]. Available: http://doi.acm.org/10.1145/3017680.3017728

[4] "Computer Programming — Computing — Khan Academy," accessed April 1, 2018. [Online]. Available: https://www.khanacademy.org/computing/computer-programming

[5] "TypeScript Playground," accessed April 1, 2018. [Online]. Available: https://www.typescriptlang.org/play

[6] "Vimeo API Playground," accessed April 1, 2018. [Online]. Available: https://developer.vimeo.com/api/playground

[7] F. Perez and B. E. Granger, "IPython: A System for Interactive Scientific Computing," *Computing in Science and Engg.*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: http://dx.doi.org/10.1109/MCSE.2007.53

[8] C. N. Klokmose, J. R. Eagan, S. Baader, W. Mackay, and M. Beaudouin-Lafon, "Webstrates: Shareable Dynamic Media," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '15. New York, NY, USA: ACM, 2015, pp. 280–290. [Online]. Available: http://doi.acm.org/10.1145/2807442.2807446

[9] J. Kato, T. Igarashi, and M. Goto, "Programming with Examples to Develop Data-Intensive User Interfaces," *Computer*, vol. 49, no. 7, pp. 34–42, July 2016.

[10] X. Zhang and P. J. Guo, "DS.js: Turn Any Webpage into an Example-Centric Live Programming Environment for Learning Data Science," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '17. New York, NY, USA: ACM, 2017.

[11] J. Kato and M. Goto, "f3.js: A Parametric Design Tool for Physical Computing Devices for Both Interaction Designers and End-users," in *Proceedings of the 2017 Conference on Designing Interactive Systems*, ser. DIS '17. New York, NY, USA: ACM, 2017, pp. 1099–1110. [Online]. Available: http://doi.acm.org/10.1145/3064663.3064681

[12] J. Kato, T. Nakano, and M. Goto, "TextAlive: Integrated Design Environment for Kinetic Typography," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 3403–3412. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702140

[13] C. Roberts, M. Wright, J. Kuchera-Morin, and T. H"ollerer, "Gibber: Abstractions for Creative Multimedia Programming," in *Proceedings of the 22nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 67–76. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654949

[16] J. Kim, P. T. Nguyen, S. Weir, P. J. Guo, R. C. Miller, and K. Z. Gajos, "Crowdsourcing Step-by-step Information Extraction to Enhance Existing How-to Videos," in *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 4017–4026. [Online]. Available: http://doi.acm.org/10.1145/2556288.2556986

[14] P.-Y. Chi, S. Ahn, A. Ren, M. Dontcheva, W. Li, and B. Hartmann, "MixT: Automatic Generation of Step-by-step Mixed Media Tutorials," in *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '12. New York, NY, USA: ACM, 2012, pp. 93–102. [Online]. Available: http://doi.acm.org/10.1145/2380116.2380130

[15] P.-Y. Chi, J. Liu, J. Linder, M. Dontcheva, W. Li, and B. Hartmann, "DemoCut: Generating Concise Instructional Videos for Physical Demonstrations," in *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '13. New York, NY, USA: ACM, 2013, pp. 141–150. [Online]. Available: http://doi.acm.org/10.1145/2501988.2502052

[17] B. Lafreniere, T. Grossman, and G. Fitzmaurice, "Community Enhanced Tutorials: Improving Tutorials with Multiple Demonstrations," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '13. New York, NY, USA: ACM, 2013, pp. 1779–1788. [Online]. Available: http://doi.acm.org/10.1145/2470654.2466235

[18] K. J. Harms, D. Cosgrove, S. Gray, and C. Kelleher, "Automatically Generating Tutorials to Enable Middle School Children to Learn Programming Independently," in *Proceedings of the 12th International Conference on Interaction Design and Children*, ser. IDC '13. New York, NY, USA: ACM, 2013, pp. 11–19. [Online]. Available: http://doi.acm.org/10.1145/2485760.2485764

[19] A. Head, C. Appachu, M. A. Hearst, and B. Hartmann, "Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct 2015, pp. 3–12.

[20] M. Gordon and P. J. Guo, "Codepourri: Creating Visual Coding Tutorials Using a Volunteer Crowd of Learners," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct 2015, pp. 13–21.

[21] "W3Schools Online Web Tutorials," accessed April 1, 2018. [Online]. Available: https://www.w3schools.com

[22] "Tutorials Point," accessed April 1, 2018. [Online]. Available: https://www.tutorialspoint.com

[23] "Codecademy," accessed April 1, 2018. [Online]. Available: https://www.codecademy.com

[24] "npm," accessed April 1, 2018. [Online]. Available: https://www.npmjs.com/

[25] "Songle Sync Tutorial," accessed April 1, 2018. [Online]. Available: http://tutorial.songle.jp/sync

[26] "GitHub Octoverse 2017," accessed April 1, 2018. [Online]. Available: https://octoverse.github.com/

[27] "Raspberry Pi," accessed April 1, 2018. [Online]. Available: https://www.raspberrypi.org/

[28] J. Kay, P. Reimann, E. Diebold, and B. Kummerfeld, "MOOCs: So Many Learners, So Much Potential ..." *IEEE Intelligent Systems*, vol. 28, no. 3, pp. 70–77, May 2013.

[29] C. G. Brinton, M. Chiang, S. Jain, H. Lam, Z. Liu, and F. M. F. Wong, "Learning about Social Learning in MOOCs: From Statistical Analysis to Generative Model," *IEEE Transactions on Learning Technologies*, vol. 7, no. 4, pp. 346–359, Oct 2014.

[30] "GitHub Gist," accessed April 1, 2018. [Online]. Available: https://gist.github.com

[31] "RawGit," accessed April 1, 2018. [Online]. Available: https://rawgit.com

[32] "Heroku," accessed April 1, 2018. [Online]. Available: https://www.heroku.com

[33] "Cloud9," accessed April 1, 2018. [Online]. Available: https://ide.c9.io