Collaborative coding interfaces on the Web

Jakub Fiala[1], Matthew Yee-King[1], and Mick Grierson[1]

[1] Goldsmiths, University of London
mu201jf@gold.ac.uk
m.yee-king@gold.ac.uk
m.grierson@gold.ac.uk

Abstract. The recent developments in Web technologies, including full-stack reactive application frameworks, peer-to-peer communication and client-side audiovisual APIs have introduced the possibility of creative collaboration in a number of contexts. Such technologies have the potential to transform the way Internet users interact with code. This paper introduces a theoretical and technical methodology for developing collaborative coding interfaces as web applications, tackling the issues of interactive rendering, user-platform interaction and collaboration. A number of existing interactive programming environments are reviewed, followed by a technical description and evaluation of *CodeCircle,* a collaborative coding web platform developed at Goldsmiths, University of London.

Keywords: Collaboration, Web Applications, Interactive interfaces

Introduction

In the recent years, the popular definition of the Internet has changed significantly – starting with the introduction of dynamic websites and Ajax (Dutta 2006), reactive web application frameworks such as AngularJS (Darwin 2013), and realtime communication technologies such as WebSockets (Hickson 2012), the Web has been widely accepted and treated as an application platform, not unlike traditional 'native' platforms such as Microsoft Windows and OS X. This notion has been further reinforced by the introduction of standardised Web APIs such as Web Audio (Adenot and Wilson 2015), WebGL and the File API (Ranganathan and Sicking 2015), which often match native platforms in terms of performance and potential. JavaScript and CSS have evolved from simple complements to HTML markup, to powerful and popular languages, the former being used on both client- and server-side, on mobile systems (Janczukowicz 2013) and even microcontrollers (Espruino.com 2016).

Unlike the static form of Internet content (Web 1.0), and the single user type of native application, the dynamic Web offers the opportunity of sharing and collaboratively creating documents, including directly executable code. This opportunity has been exploited in open-source software projects on platforms such as GitHub (GitHub 2016), 'pastebin' tools such as jsFiddle (Jsfiddle.net 2016), and cloud-based Web IDEs such as Cloud9 (Ciortea et al. 2010). Recently, a number of projects have also used Web Audio and Web GL to create audiovisual live coding environments, such as Gibber (Roberts and Kuchera-Morin 2012) and Livecodelab (Della Casa and John 2014). Applications like glslsandbox (Glslsandbox.com 2016) and Shadertoy (Jeremias and Quilez 2014) focus on developing small, self-contained visual pieces reminiscent of the Demoscene. In this paper, we introduce CodeCircle which is the result of research in Web-based collaborative live coding, combining the elements of Web IDEs and rapid prototyping pastebin apps with real-time collaboration and rendering.

Research objectives

CodeCircle is the result of our ongoing, in-depth research in digital interactivity and collaboration. Our principal aims in developing CodeCircle are as follows:

- To design and develop a web-based interface that enables real-time, collaborative, and social coding in a creative context.

- To implement an integrated code sharing and collaboration system that enables us to study the process of collaborative, creative coding.
- To devise the platform in such a way that makes it attractive and accessible to learners and professionals alike, and can be used in a variety of contexts including computing education.

This paper is mainly concerned with the first phase of this research. In the 'Existing platforms' section, we discuss the existing web-based coding environments, and compare their feature sets with the desired features of the CodeCircle platform. We then elaborate on the issues and requirements related to the main features of existing applications. In the 'Technical description' section, we describe the various design decisions and solutions implemented in CodeCircle. Finally, we discuss the potential use cases and the future development, as well as research, in the CodeCircle project.

Existing platforms

There are numerous publicly available web based coding interfaces. However, their intended use cases, which inform their design decisions, vary significantly. Through a survey of existing web based coding interfaces, we have identified the following classes:

- **Pastebin-style applications.** These interfaces enable rapid prototyping of simple web code, usually allowing HTML, JavaScript and CSS code, and render using the browser as the engine. They are designed to store code experiments and examples, and are often used as supplements to programming community forums such as Stack Overflow. Pastebin applications surveyed in our research include jsFiddle, JSBin, CodePen, Codr, Liveweave, dabblet, Mozilla Thimble and CSSDeck.
- **Live and creative coding and environments.** Another distinct class of web-based coding platforms are creative coding applications. These are designed specifically for live coding performances, education, or building artistic works. They usually allow the user to write in one language, and either use user code to control an internal audiovisual engine (Livecodelab, Gibber) or inject the code into a pre-built web page (GLSL Sandbox, Shadertoy).
- **Web IDEs.** These are targeted at software companies and developers, and usually include cloud container services, rich asset systems, and complex coding tools. They always require authentication to access documents. While these applications enable development of complex client-server architectures and present the user with a very large feature set, their document representations are not easily shareable, and they require high skill levels. In our research, we reviewed several Web IDEs, including Cloud9, Koding, and Codeanywhere.

	Platform type			Features					
Platform	Live coding	Pastebin	IDE	Live mode	Code Validation	Forking	Collaboration	Assets	Shareable
Gibber	x				Yes		Partial		
jsFiddle		x				Yes	Partial		Yes
Livecodelab	x			Yes	Yes				
CodePen		x		Yes		Yes	Partial		Yes
GLSL sandbox	x	x		Yes	Yes[1]				Yes
Codr		x		Yes		Yes	Yes		Yes
Cloud9			x		Yes		Yes	Yes	
CodeCircle	x	x	x	Yes	Yes	Yes	Yes	Yes	Yes

Table 1. Comparison of feature sets of selected existing platforms

One of the principal aims of the CodeCircle platform is to combine certain features of all web coding application classes to build an environment that supports multiple approaches to writing code on the Web. While it is important for such a platform to provide the comfortable user experience of a Web IDE, combined with the ease of use and shareability of

[1] No error description

pastebin apps, it is also crucial to leverage the creative and educational potential of live coding interfaces. Finally, by emphasizing the aspect of collaboration and sharing on the platform, CodeCircle is an attempt at creating an environment for *real time social coding* – a practice which may radically change the way we interact with code and computing in general.

Feature requirements

Building on the survey of existing platforms, we can specify a set of requirements for the CodeCircle platform which differentiate it from pre-existing platform and which will imbue it with the novel functionality required to enable us to begin addressing our research objectives. First and foremost, an integrated collaboration system is required which enables semantically synchronized real-time collaboration, applied directly to document content. This is in contrast to systems such as *TogetherJS,* which is used in a number of pastebin-style applications. Second, it is necessary to maintain a high level of interactivity, i.e. maximize the intensity of system feedback and minimize the feedback delay. Many existing platforms, such as CodePen and JSBin include an 'auto-update' function, which renders user code automatically as it changes. Others, including jsFiddle and Gibber, require user interaction to perform rendering, which increases feedback delay. Furthermore, manipulating binary assets is a core requirement for a number of audiovisual programming techniques including sampling, processing and sequencing. It is often impractical and limiting to work in an environment without the possibility of using pre-made content. Functional asset systems are currently only implemented in professional Web IDEs (Ciortea et al. 2010)(Koding.com 2016). Finally, pastebin tools such as jsFiddle have been widely adopted as the standard way of sharing code snippets on the Web. According to Goméz et al. (2013), Stack Overflow contains over 300 000 references to jsFiddle URLs. A simple shareable URL scheme and a forking function could improve the social aspect of the platform.

CodeCircle – technical description

In this section we provide a technical description of the CodeCircle system and explain how it meets the requirements listed above.

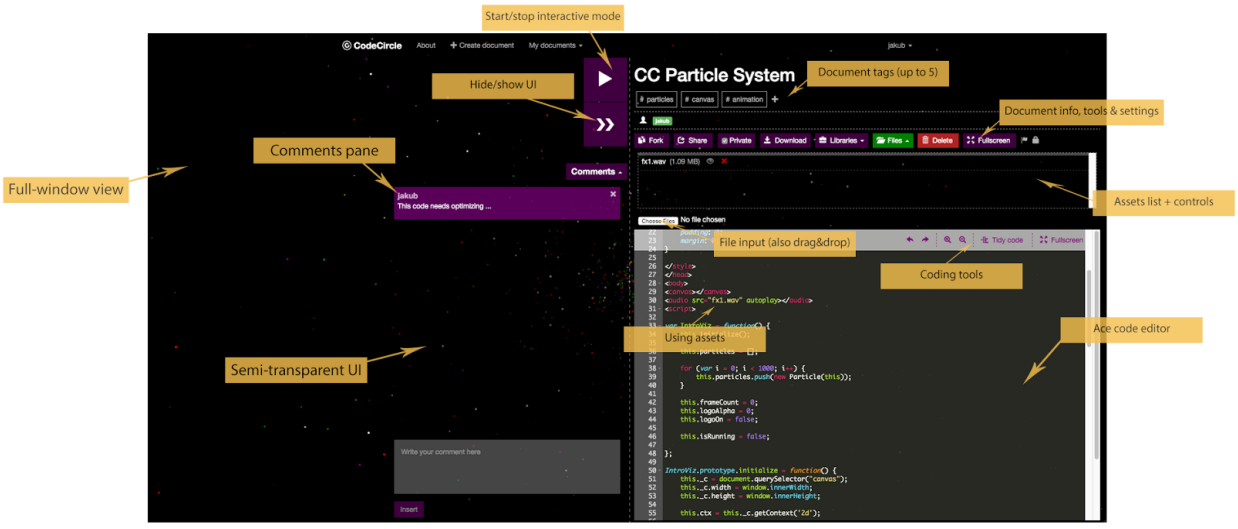

Figure 2. Screenshot of the CodeCircle UI

The user interface (UI) of CodeCircle aims to mirror the underlying rendering system structure by splitting the window into two parts – first, a dismissable semi-transparent editor panel containing document settings, controls, and the Ace code editor. Second, the viewer frame, which is positioned behind the editor panel and spans the entire window. This layout paradigm is prevalent in live coding environments that are based on a custom audiovisual engine, such as Gibber and Livecodelab. It is also extremely useful in a direct rendering environment, because it increases the intensity and immediacy

of visual feedback from the application. Documents can be viewed in a viewer-only embeddable mode by appending */view* to the document URL. CodeCircle also includes a number of coding tools and features to improve the user experience. We used the beautify.js library to provide a 'Tidy code' function which automatically adjusts line indentation. By embedding the Ace editor, the UI also benefits from code highlighting, multiple cursors, and code completion. Similarly to a number of pastebin-style platforms, each document can be associated with pre-built JavaScript and CSS libraries that are automatically added to the code before rendering.

Application structure

Real-time collaboration requires reactive content, which stays up to date without the need for refreshing the page. Solutions to this problem are part of many web frameworks, such as Angular, React.js, etc. For CodeCircle, we decided to use Meteor (Meteor.com 2016), a full-stack framework that combines a number of JavaScript libraries, and a server- and client-side engine. Meteor has a number of features that help build scalable collaborative applications rapidly. Since its server-side engine is based on node.js and MongoDB, and client engine on Blaze (Meteor.com 2016), the entire app logic is written in JavaScript. Meteor's subscribe/publish model and reactive templates are ideal for managing complex real-time interactions between collaborating users. The Meteor package library contains useful packages wrapping systems such as ShareJS, UI libraries, etc.

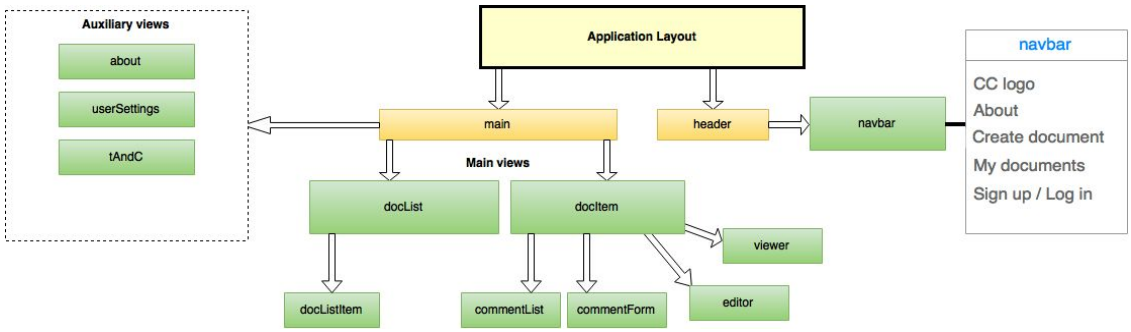

Figure 3. CodeCircle client-side template structure

The CodeCircle application layout consists of two main views, and a set of auxiliary views. Each view is represented as a Meteor template. Meteor's dynamic templates system enables a highly structured approach to web application development, where individual templates can be reused in a modular fashion. Each of the main views is a template, which is composed of one or more sub-templates. In Meteor, each template is given its own set of helper functions and event handlers, which allows for even greater flexibility, and ensures that any interactivity is directly bound to its corresponding HTML structure in the application.

Rendering system

In its simplest incarnation, a real-time web-based web development environment consists of two elements: a code editor and a viewer frame, to which the code is rendered. These modules are easy to implement in HTML using the `<iframe>` tag for the viewer, and the `<textarea>` element for the editor. With these elements in place, a JavaScript mechanism is needed to render the contents of the `<textarea>` as the DOM (Document Object Model) of the `<iframe>`. This is the code renderer, the core element of any browser-based coding environment.

```
var editor = document.querySelector('textarea');
var viewer = document.querySelector('iframe');

editor.onkeydown = function() {
      var code             = editor.value;
      viewer.innerHTML     = code;
```

```
}
```

A simple HTML code renderer.

The above code executes an event handler whenever a key is pressed while the editor element is in focus. This solution seems perfectly good for writing basic HTML and CSS code – if the entire "webpage" is written as a single HTML file with inline CSS. On a modern browser, changes appear in the iframe element immediately. However, several problems arise when this simple system is implemented: the foremost being that when a script tag is included in the code, the browser does not execute the JavaScript inside. Furthermore, with enough code to render and a high typing frequency, the rendering system becomes overloaded, and the JavaScript thread freezes or slows down significantly. However, we can use a slightly more complex technique to render script content, too:

```
viewer.onload = function () {
      //write the new DOM
      viewer.contentDocument.open();
      viewer.contentDocument.write(code);
      viewer.contentDocument.close();
}

viewer.src = "";
```

Writing to an iframe.contentDocument stream.

With this technique, we have solved the script issue, as well as another problem arising from rendering a DOM repeatedly – when using computationally heavy APIs such as the Web Audio API, there is often a limited number of "contexts" one can create in a single window instance. However, a context needs to be created every time a new DOM is rendered, as the old contexts fall out of the new JavaScript scope. After a certain number of rendering routines[2], new contexts cannot be created anymore. The `<iframe>` element needs to simulate a page reload in order to release the existing context(s) and enable the creation of a new one. The one remaining issue is related to execution frequency. Since CodeCircle is primarily envisioned as a creative programming platform, the renderer response should be as close as possible to real time. Rendering on each keystroke results in a great performance decrease above a certain typing speed and DOM size. It is necessary to find a compromise between rapid renderer response and efficient execution. The most useful API for achieving this is requestAnimationFrame (Robinson and McCormack 2015), which aligns JavaScript execution with the browser's animation loop. The code is executed only if the browser is ready to render another frame, avoiding the 'bottleneck' which would increasingly prolong the time needed to continue rendering.

Code validation

In a secure web application, it is important to ensure that only valid, secure code is being executed on the client. Apart from the obvious security and privacy rationale, code validation is also important for performance reasons – code that causes errors at runtime decreases JavaScript performance significantly, resulting in suboptimal user experience. There are a number of tools for HTML/CSS/JavaScript validation, the most popular being JSHint (Kovalyov 2010), CSSLint (Csslint.net 2016) and HTMLHint (Htmlhint.com 2016). The latter depends on the former two as plugins. All of these are written in JavaScript, and can be easily used in the browser.

In CodeCircle, we used a custom version of HTMLHint, which normally collects JSHint and CSSLint with the require() function, combining the three scripts into a single validator object. The validator script is executed in a Web Worker to improve performance. While increasing execution time, this approach also relieves the JavaScript thread, which results in a smoother typing response. Additionally, validation parameters must be set to accept many coding styles, but ensure that as little code with fatal errors passes the test as possible. An example of this is requiring semicolons – the code does not render until the user has finished typing the line.

[2] In WebKit-based browsers, the maximum number of AudioContexts seems to be around 6

Assets

To solve the problems of secure asset loading, we have developed a system based on the CollectionFS Meteor package (GitHub 2016), which uses a GridFS file system in the application's database. When the user inserts a file on the client, the file is uploaded and associated with the currently opened document. Image, audio and video assets can be previewed in a modal window. Once the renderer receives new code to render, it replaces each occurrence of the asset filename with a data URI representation of the asset, and the asset can be used in HTML `src` attributes. The server-side URL of the asset is not exposed, and the user can include assets in their code by simply specifying their filenames. Another problem arises when loading assets via an XMLHttpRequest (XHR), which is a common pattern particularly in Web Audio applications. At the time of writing of this paper, the native XHR object only supported data URIs in the Chrome browser. To enable asset usage in all browsers, we integrated an XHR shim which wraps the native object, detects data URIs, and decodes them into arraybuffers without calling the native XHR methods.

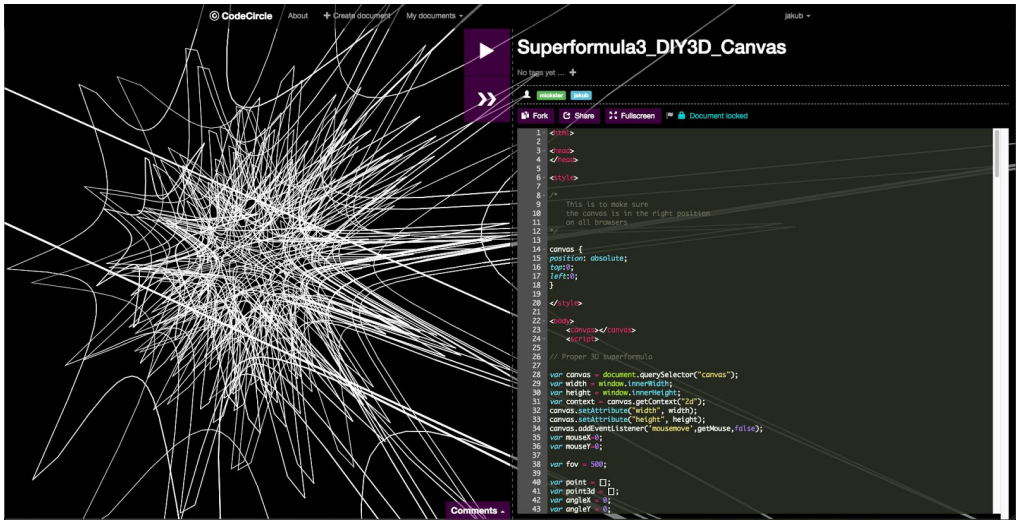

Figure 4. Public, locked CodeCircle document – content is visible, but only the owner can edit it.

Collaboration and sharing

The collaboration functionality in CodeCircle draws inspiration from web applications such as Google Docs (Blau and Caspi 2009) and Overleaf (Overleaf.com 2016). Some popular pastebin platforms have implemented collaborative features with TogetherJS (Mozilla Labs 2016), a client-side library that enables real-time chat and view sharing. This approach does not integrate into the platform structures, and is not extensible or adjustable based on the particularities of the platform. Furthermore, TogetherJS shares the entire page view, which may result in decreased performance if the page DOM is complex. In CodeCircle, we have implemented a real-time collaboration system based on the ShareJS library, wrapped in the *mizzao:sharejs* package. ShareJS is designed for collaboration on text and JSON data, rather than entire web pages. The library is based on the concept of *ops,* i.e. highly specified representations of changes to a shared document. Ops enable efficient synchronisation of document views on a number of clients controlled by a server. A record of all ops applied to a document is stored in the database, containing metadata such as user ID. In CodeCircle, ShareJS is integrated in the code editor for each document, provided that the user is logged in.

CodeCircle documents can be shared either using a URL scheme inspired by community platforms such as Reddit, or by embedding generated `<iframe>` code. The document URL begins with the domain name, followed by the *'d/'* prefix and the document ID. All public documents are visible and searchable, although the owner may chose to lock a document, which results in the code editor being created as read-only. Any public document can be forked, which creates a copy of its code, and associates the original assets and the new document.

Planned use cases and future work

The main goal of CodeCircle is to provide a *social coding* platform, combining the features of existing pastebin tools, Web IDEs and live coding applications. There are many potential use cases for the platform: the highly interactive UI style and document shareability of CodeCircle implies the possibility of using the platform as a 'demo showcase tool' similar to GLSL Sandbox. The social and collaborative aspects of the application also offer themselves to educational use – for instance, CodeCircle may be used by a lecturer to prepare examples for their class, and by students to submit and share collaboratively developed code, with a straightforward way of receiving feedback in the comments. The platform is already in use in an online creative coding course on Kadenze. In addition to showcasing and educational sharing, CodeCircle can support 'code crowdsourcing'. Because of the openness of the system, where any registered user can contribute to other users' code, users are invited to propose coding 'problems' and enable the community to contribute to the solution. The ownership of a document becomes less important as the importance of collaboration increases, and the concept of authorship is dismissed. The next step in the development of CodeCircle is to test the effect of an open collaborative system on the way users interact with code and with each other. Testing shall be performed by using CodeCircle as a teaching platform for both small and large groups of students, as well as specific user experience studies with groups of users from a variety of backgrounds. The integration of ShareJS enables detailed monitoring of the user-document interaction, and the obtained data may produce interesting insights into the dynamics of collaboration in creative coding.

Conclusion

Technologies for collaborative programming and code sharing on the Web are a relatively novel, but very powerful addition to the Internet. In this paper, we have provided a theoretical framework for analyzing web-based coding environments. Next, we discussed functional requirements arising from their architecture, including collaborative and social aspects, code rendering, UI design and asset systems. By analyzing existing platforms and applications, we have identified solutions to some of these issues. Finally, we have provided an extensive technical description of the CodeCircle platform, which attempts to provide an environment for a fundamentally different, collaboration-driven way of developing audiovisual web programs. While the potential of the *social coding* concept can only be assessed by rigorous studies and testing on the CodeCircle platform, it is already possible to state that by combining a modern web application framework such as Meteor with an integrated collaboration system, an open sharing environment, a secure asset system and a highly interactive user experience, technical and design-related issues that arise in web-based coding platforms may be addressed effectively. The resultant web environment has a unique feature set that may provide a unique creative coding experience, as well as new modes of interaction between the community and the documents on the platform.

Bibliography

Adenot, Paul, and Chris Wilson. 2015. "Web Audio API". W3.Org. https://www.w3.org/TR/webaudio/.

Blau, Ina, and Avner Caspi. "What type of collaboration helps? Psychological ownership, perceived learning and outcome quality of collaboration using Google Docs." In *Proceedings of the Chais conference on instructional technologies research*, vol. 12. 2009.

Ciortea, Liviu, Cristian Zamfir, Stefan Bucur, Vitaly Chipounov, and George Candea. 2010. "Cloud9". ACM SIGOPS Operating Systems Review 43 (4): 5. doi:10.1145/1713254.1713257.

Codepen.io,. 2016. "Codepen". Accessed February 20. http://codepen.io.

Codr.io,. 2016. "Codr". Accessed February 20. http://codr.io.

Csslint.net,. 2016. "CSS Lint". Accessed February 20. http://csslint.net.

GitHub,. 2016. "Build Software Better, Together". Accessed February 21. http://github.com.

Darwin, Peter Bacon, and Kozlowski, Pawel. *AngularJS web application development*. Packt Publ., 2013.

Della Casa, Davide, and Guy John. 2014. "Livecodelab 2.0 And Its Language Livecodelang". Proceedings Of The 2Nd ACM SIGPLAN International Workshop On Functional Art, Music, Modeling & Design - FARM '14. doi:10.1145/2633638.2633650.

Dutta, Sunava. 2006. "Native XMLHttpRequest Object | IEblog". Microsoft.com. https://blogs.msdn.microsoft.com/ie/2006/01/23/native-xmlhttprequest-object/.

Espruino.com,. 2016. "Espruino - Javascript For Microcontrollers". http://www.espruino.com.

GitHub,. 2016. "Collectionfs". Accessed February 20. https://github.com/CollectionFS/Meteor-CollectionFS.

Glslsandbox.com,. 2016. "GLSL Sandbox Gallery". Accessed February 20. http://glslsandbox.com.

Gómez, Carlos, Brendan Cleary, and Leif Singer. "A study of innovation diffusion through link sharing on stack overflow." In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 81-84. IEEE Press, 2013.

Hickson, Ian. 2012. "The Websocket API". W3.Org. http://www.w3.org/TR/websockets/.

Hickson, Ian. 2015. "Web Workers". W3.Org. https://www.w3.org/TR/workers/.

Htmlhint.com,. 2016. "Htmlhint". Accessed February 20. http://htmlhint.com.

Janczukowicz, Ewa. *Firefox OS Overview*. Telecom Bretagne Research Report RR-2013-04-RSM (November 2013), 2013.

Jeremias, Pol, and Inigo Quilez. 2014. "Shadertoy". SIGGRAPH Asia 2014 Courses On - SA '14. doi:10.1145/2659467.2659474.

Jsfiddle.net,. 2016. "Jsfiddle". https://jsfiddle.net.

Koding.com,. 2016. "Koding For Teams | Configure Any Development Environment In One Click". Accessed February 20. https://koding.com.

Kovalyov, Anton, W. Kluge, and J. Perez. "JSHint, a JavaScript Code Quality Tool." (2010).

Meteor.com,. 2016. "Meteor". http://meteor.com.

Mozilla Labs,. 2016. "Mozilla Labs : Togetherjs". Accessed February 20. https://togetherjs.com.

Overleaf.com,. 2016. "Overleaf: Real-Time Collaborative Writing And Publishing Tools With Integrated PDF Preview". Accessed February 20. https://www.overleaf.com.

Ranganathan, Arun, and Jonas Sicking. 2015. "File API". W3.Org. https://www.w3.org/TR/FileAPI/.

Roberts, Charlie, and J. Kuchera-Morin. *Gibber: Live coding audio in the browser*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2012.

Robinson, James, and Cameron McCormack. 2015. "Timing Control For Script-Based Animations". W3.Org. https://www.w3.org/TR/animation-timing/.