**ISBN** 978-981-11-7861-0
**Proceedings of 2018 the 8th International Workshop on Computer Science and Engineering (WCSE 2018)**
*Bangkok, 28-30 June, 2018, pp. 678-684*
doi: 10.18178/wcse.2018.06.112

# CodeWiki: Toward Efficient Source Code Sharing on the Web

Hiroaki Fukuda[1+], Worachat Arunothaikrit[1] and Maylada Kanokchoteworakarn[1]

[1] Shibaura Institute of Technology, 3-7-5 Toyosu Koto, Tokyo, Japan

**Abstract.** Nowadays most of developers can find some useful references of source code on the website. Moreover, those source codes come with their explanations which usually located above or below the code. When the code is too long to fit within one page, the explanation will be inconsistently corresponding with the code because users have to scroll the page for comprehending both codes and explanations. As a consequence, users have to wasted extra time to conjugate the pieces of code with their explanations. In addition, most of wiki systems provide only single area to edit both code and explanation which also consuming the time for writing a good explanation for each section of the code. This paper offers a specific type of wiki system with provided parallel area for reading and editing source code, called CodeWiki. Besides, Codewiki allows readers to click on each section of the code for its in-line explanations. Consequently, readers and editors do not waste their time on scrolling the page and enable the correspondent between the source code and its explanation. We propose a prototype implementation of CodeWiki and the result of its usage.

**Keywords:** Wikis, corresponding reading and editing, source code sharing

## 1. Introduction

The web based services have significantly increased which influenced the usage of the web to find what people need. Especially for developers, finding the source code references is one of the common ways to understand the reason or technique. Most of those sharing websites also provide the explanations for the source codes to help readers to understand the context easier. Those websites usually created and edited using Wiki [1], [2] which allows users to express the ideas and share their information by using hypertext. The traditional wiki provides special tags, which enable editors to express their idea quicker than using (X)HTML tags. However, the usage of the traditional wiki system is limited because the obstacle of corresponding between the source code and its explanation. Most of wiki system do not have a tag for collaborate between the source code and explanation so the explanation cannot be seen parallel with the source code. When the source code is too long to fit in one page, the readers have to scroll the page which wasted their time and hard to comprehend with the source code. Moreover, the traditional wiki system usually has a single window for editing the contents. The editors also have to scroll the page and cannot efficiently use the tag to mark the specific area of the source code to add more information which leading to the mistakes and insufficient explanation for those source codes.

This paper proposes a special wiki system for writing and reading with more efficient way to corresponding between the source code and its explanation, called CodeWiki which provides multiple areas for editors to mark the specific area of the code with its explanation, so the editors can effectively see their source code parallel with the window to enter the explanation. Moreover, CodeWiki also allows readers to see the source code and its explanation as a parallel window or click the specific area of the source code that has been marked to see more detail of the explanation. Consequently, editors and readers do not need to scroll the page, reducing wasted time on scrolling and making the usage of understand and write the source

[+] Corresponding author. Tel.: + 81358598518; fax: +81358598500
*E-mail address*: hiroaki@shibaura-it.ac.jp.

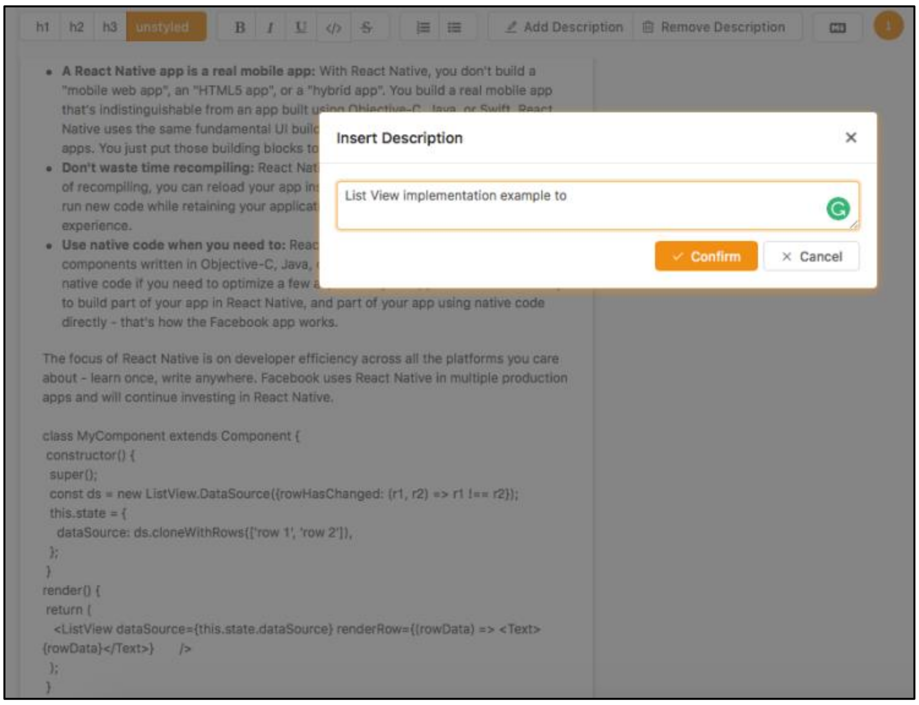
code corresponding with it explanation become easier. We propose a prototype of implementation of CodeWiki and the result of its usage.

## 2. Approach

As we have mentioned, CodeWiki provides an editor section with editors and reader section for readers. In this section, we explain each section in detail.

### 2.1. Editor Section

CodeWiki provides the functions to improve the traditional wiki system for editing the contents. The design of CodeWiki purposes to reduce extra time on scrolling the page and see the source code and its explanation corresponding together.



- **A React Native app is a real mobile app:** With React Native, you don't build a "mobile web app", an "HTML5 app", or a "hybrid app". You build a real mobile app that's indistinguishable from an app built using Objective-C, Java, or Swift. React Native uses the same fundamental UI buil apps. You just put those building blocks to
- **Don't waste time recompiling:** React Nat of recompiling, you can reload your app in run new code while retaining your applicat experience.
- **Use native code when you need to:** Reac components written in Objective-C, Java, native code if you need to optimize a few a to build part of your app in React Native, and part of your app using native code directly - that's how the Facebook app works.

The focus of React Native is on developer efficiency across all the platforms you care about - learn once, write anywhere. Facebook uses React Native in multiple production apps and will continue investing in React Native.

```
class MyComponent extends Component {
 constructor() {
  super();
  const ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
  this.state = {
   dataSource: ds.cloneWithRows(['row 1', 'row 2']),
  };
 }
render() {
 return (
  <ListView dataSource={this.state.dataSource} renderRow={(rowData) => <Text>
{rowData}</Text>}      />
  );
 }
```

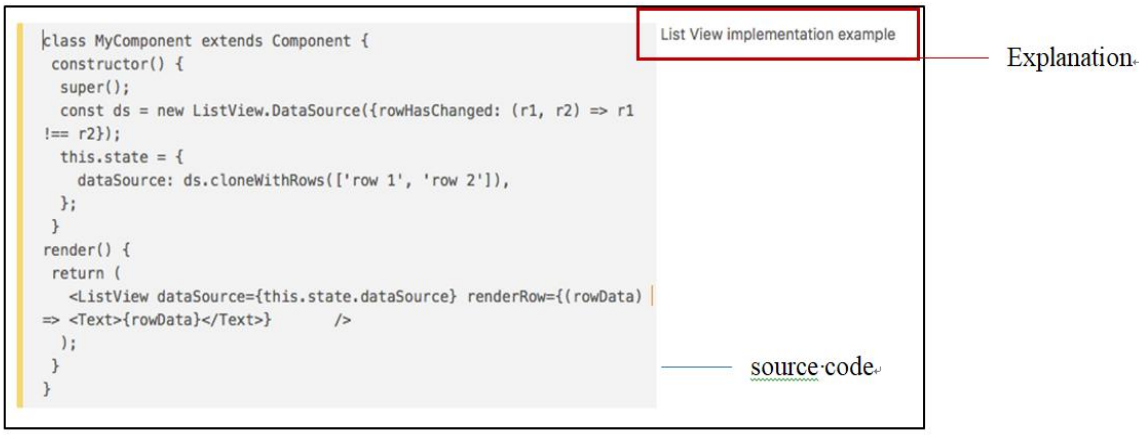
List View implementation example to

Fig. 1: A design for adding the explanation to the source code.

CodeWiki allows editors to add two types of explanations: a block explanation attached to a code block that contains multiple lines of source code, and an inline explanation attached to certain texts that are parts of source code. We assume that a block explanation is used to give a brief explanation for a code block such as an algorithm, and an inline explanation is also used to give a more detailed explanation such as explanations of arguments in a method invocation if needed. For both explanations, CodeWiki prepares a separated area to add the explanations with a pop-up window which allows editors to see both source code on the background and explanation section as shown in Fig. 1. Once editors click the "confirm" button, the explanation written the pop-up window will connect to the source code chosen by the editor. In addition, when the editor click "sub-description" button, the description written by the editor will connect the chosen source code as an inline explanation. Finally, once editors save their contents (e.g., source code and explanations), the CodeWiki generates a content which combine the source code and the corresponding explanation as a Web content.

### 2.2. Reader Section

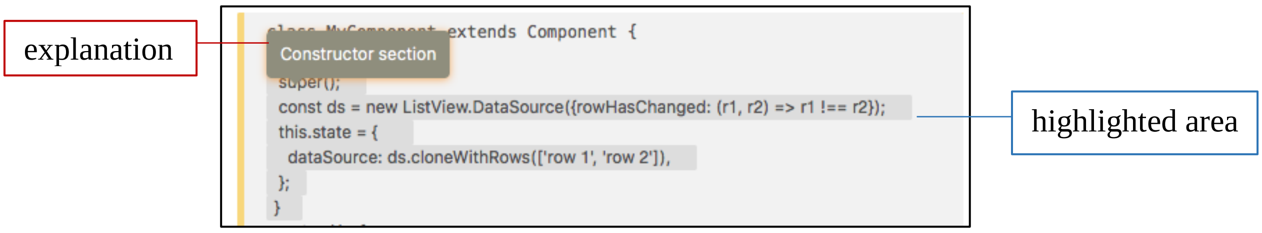
Readers want to read the source code along with explanation without scrolling the page. However, some readers do not need to see every sub-description of the specific source code. Therefore, it is better for readers to choose whether which section of the explanation that they want to see and prevent the readers from misunderstanding each section of the source code.



```
class MyComponent extends Component {
 constructor() {
  super();
  const ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1
!== r2});
  this.state = {
    dataSource: ds.cloneWithRows(['row 1', 'row 2']),
  };
 }
render() {
 return (
   <ListView dataSource={this.state.dataSource} renderRow={(rowData)
=> <Text>{rowData}</Text>}       />
  );
 }
}
```

Fig. 2: A design for displaying explanation of the source code.

The description of the source code is normally presented on the right side of the screen to help readers to coordinate between the source code and its explanation as shown in Fig. 2. Moreover, as described section 2.1, readers can see the inline explanation with pop-up when they hover the mouse over the specific area as shown in Fig. 3.



```
class MyComponent extends Component {
 super();
 const ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
 this.state = {
  dataSource: ds.cloneWithRows(['row 1', 'row 2']),
 };
}
```

Fig. 3: A design for displaying sub-description.

Readers can easily find the sub-description by the highlighted area of the source code. The highlighted area allows readers to hover on to see the specific explanation with pop-up, enabling readers to understand the pieces of source code without scrolling and help readers to emphasized their interested area of the source code.

In this way, the CodeWiki gives two ways for readers to see the explanations without scrolling the screen, resulting in the ease and efficiency for understanding specific areas of the code.

## 3. Implementation

This section explains the implementation details of CodeWiki. We briefly explain the architecture of CodeWiki and components that consist of CodeWiki. Then we mention how to connect source code to the corresponding explanation.

### 3.1. System Architecture

Fig. 4 shows the entire architecture of CodeWiki. CodeWiki mainly consists of two parts: FRONT-END and BACK-END.

In FRONT-END, we combine React [3] and Redux [4]. React is a library which provides view components on the Web browser. Using React enables to building interactive user interfaces, meaning that the data will be changed over time without reloading the web browser. Redux is a library that can control states of the application. The combination of React and Redux allows developers to provide interactive user interfaces with well-managed state of the application.

In BACK-END, we use Node.js [5] as a runtime engine of the server, and Hapi.js that is a framework running on Node.js and make it easy for developers to implement server-side applications. This is because it seems to be efficient for developers to use only one programming language (e.g., JavaScript) to develop an application. The data transmitted between FTONT-END and BACK END is formalized by JSON Web

Token (JWT) for secure information between parties as a JSON object with digital signature called JSON Web Signature (JWS).

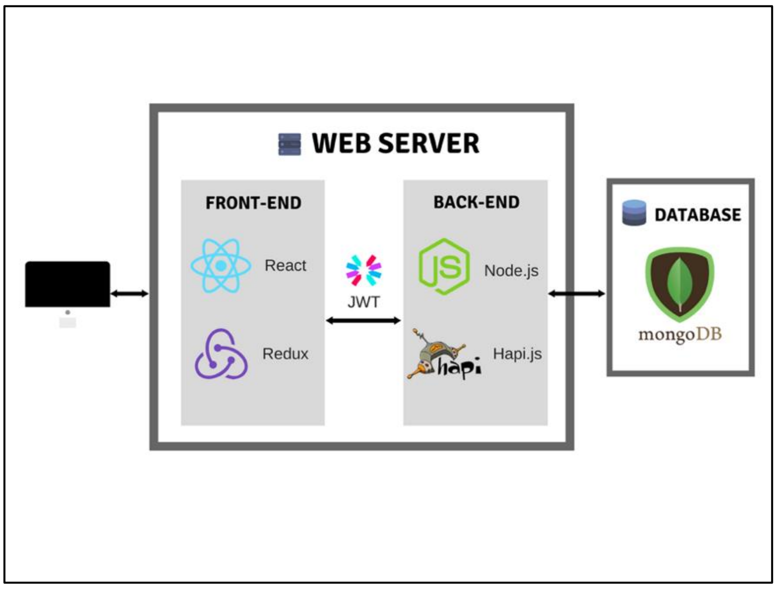


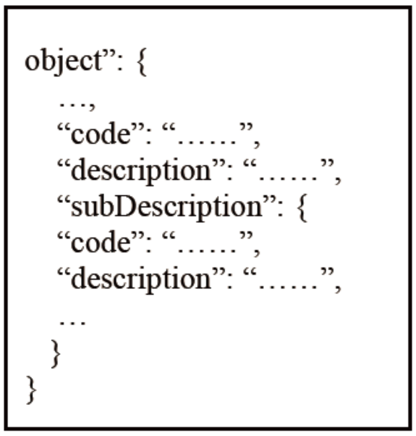
Fig. 4: The system architecture of CodeWiki.

The last component is Database to save data. We use MogoDB because it is one of the NoSQL database and uses JSON-like documents with schemas. The data transmitted from FTON-END to BACK-END is stored in this MongoDB in order to be used as persistent data.

### 3.2. Connection between Source Code and Its Explanation.

CodeWiki need to provide a rich text editor as traditional wiki systems. We choose Draft.js for this purpose. Draft.js is a framework for building complex and rich text editor in React. Using Draft.js enables developers to customize their own rich text editor by adding features such as image, text size, text width. We add the two types of explanations using Draft.js in CodeWiki.

The methodology of connecting between source code and explanation is using Draft.js libraries as a connection. After editors finish editing their source code, they have to decide the feature of the editor whether to add a description or sub-description. The source code and explanation will be transform into JSON format, therefore the source code will hold as "code" key which is an identification of the data and description will hold as "description" in JSON format. Moreover, if writers add the sub-description to the source code, the system will generate as another feature and store the sub-description as another key within the same object of the source code that has been tagged as a source code in the first place in JSON format as shown in Fig. 5:



```
object”: {
    …,
    “code”: “……”,
    “description”: “……”,
    “subDescription”: {
    “code”: “……”,
    “description”: “……”,
    …
   }
}
```

Fig 5: JSON format.

The methodology of adding description and sub-description is the almost same but the majorly difference is the key that hold the data of each component. After the editors saved the content, Draft.js uses the function to convert JSON to JSON raw. The JSON raw has been send as a state through the system with Redux libraries handling all the state in the system, then JSON raw will be store in the MongoDB to be called when readers want to read the content. The content will sequentially transform from JSON raw to JSON and texts to display on the screen as a reverse process from the way of creating the content. As shown in Fig. 6:

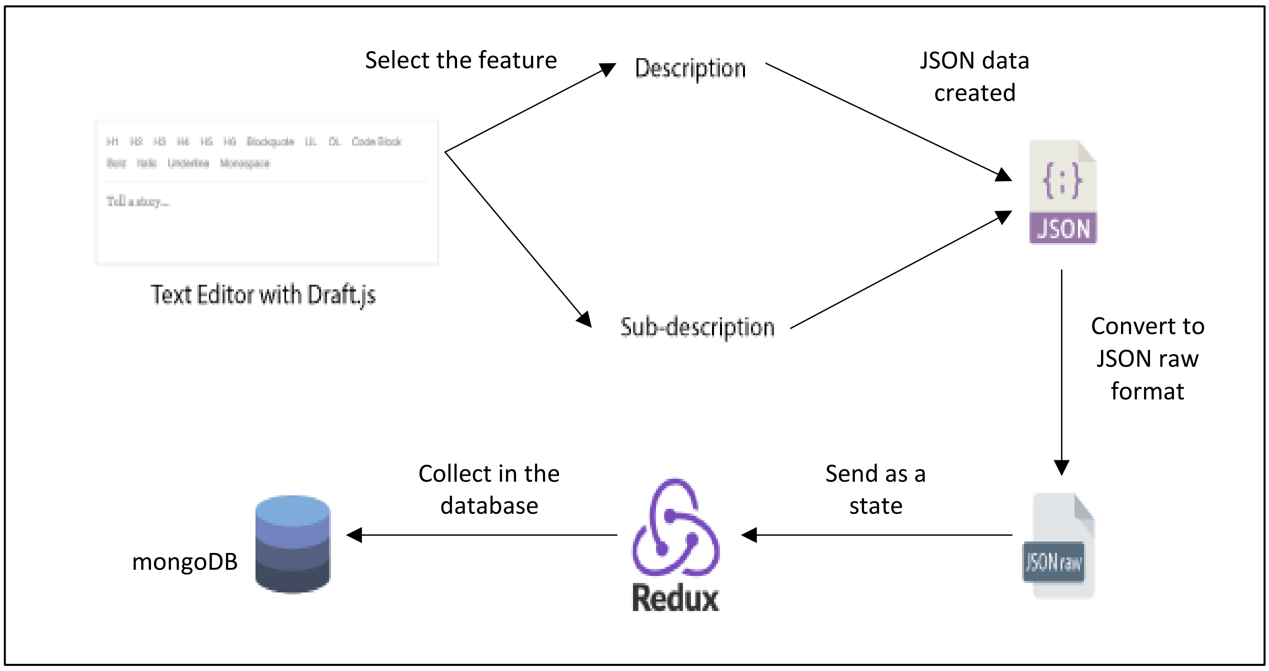
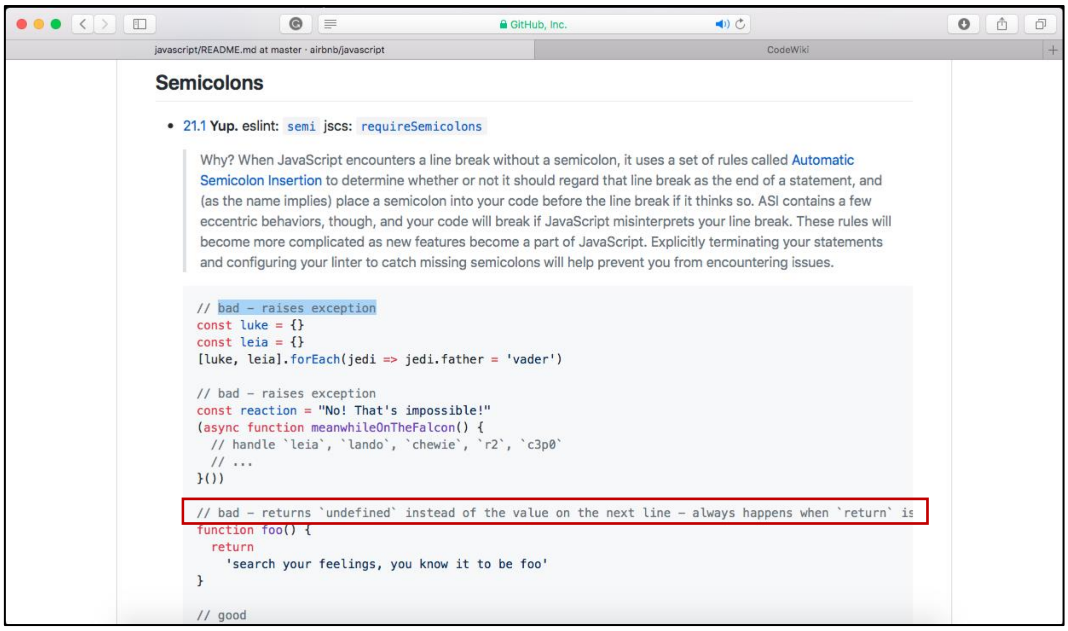


Fig. 6: The work flow of adding description and sub-description to the source code.

## 4. Evaluation

Codewiki is a web-based service so both editors and readers can access this useful service via the Internet as a free service. Codewiki is a new system to express the code and explanation in the different ways from the traditional wiki system. CodeWiki allows users to correspond between source code and its explanation within one screen to prevent scrolling the pages to read the corresponding information by using multiple area for showing both code and its explanation and also provides multiple area of text editor space to easily write the code and explanation when sharing with other people. As the experiment with same content on both traditional system and Codewiki, as shown in Fig. 7, the traditional wiki system holds a long and over-length content as users have to scroll both up-and-down. Besides, using the comment notation (e.g. "//") in the source code section makes the source code is longer than the area provided so readers have to scroll left-and-right to read all the comments of this specific line of code which caused inconsistency to the readers. Moreover, using comment notation also causes a duplicate explanation.

On the other hand, as shown in Fig. 8, Codewiki helps users to avoid using comment notation to describe the source code in the line with the description and sub-description feature which help using to describe the all-over detail of the source code and also enhance users to give a detail about the specific line of code. Also, Codewiki provides the explanation next to the source code within a page so user do not bother scrolling left-and-right to read all the comment about particular line of code. Moreover, Codewiki also using an area in the page worthwhile for display the content so readers can see consistency content within a page. Comparing to traditional wiki system, Codewiki shows more lines of source code with different method of displaying the source code, description and sub-description instead of using normal text to be an explanation or comment notation to describe the specific line code which also prevent using duplicate explanation for the source code.



Semicolons

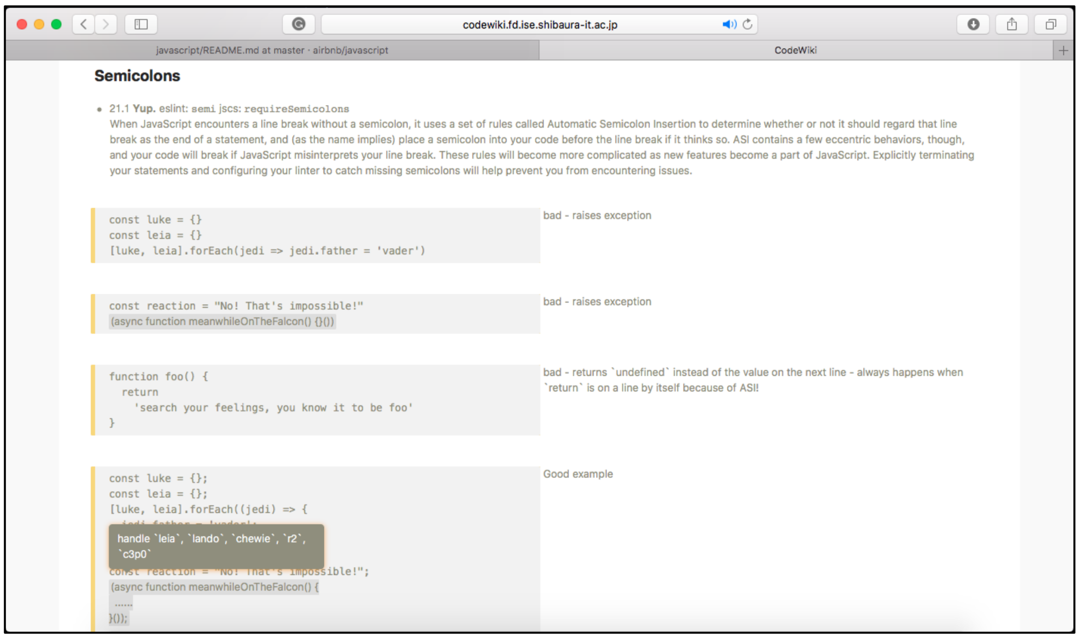
- 21.1 Yup. eslint: `semi` jscs: `requireSemicolons`

> Why? When JavaScript encounters a line break without a semicolon, it uses a set of rules called Automatic Semicolon Insertion to determine whether or not it should regard that line break as the end of a statement, and (as the name implies) place a semicolon into your code before the line break if it thinks so. ASI contains a few eccentric behaviors, though, and your code will break if JavaScript misinterprets your line break. These rules will become more complicated as new features become a part of JavaScript. Explicitly terminating your statements and configuring your linter to catch missing semicolons will help prevent you from encountering issues.

```
// bad - raises exception
const luke = {}
const leia = {}
[luke, leia].forEach(jedi => jedi.father = 'vader')

// bad - raises exception
const reaction = "No! That's impossible!"
(async function meanwhileOnTheFalcon() {
  // handle `leia`, `lando`, `chewie`, `r2`, `c3p0`
  // ...
}())

// bad - returns `undefined` instead of the value on the next line - always happens when `return` is
function foo() {
  return
    'search your feelings, you know it to be foo'
}

// good
```

Fig. 7: The display of traditional wiki system.

Codewiki helps users to avoid using comment notation to describe the source code in the line with the description and sub-description feature which help using to describe the all-over detail of the source code and also enhance users to give a detail about the specific line of code. Also, Codewiki provides the explanation next to the source code within a page so user do not bother scrolling left-and-right to read all the comment about particular line of code. Moreover, Codewiki also using an area in the page worthwhile for display the content so readers can see consistency content within a page. Comparing to traditional wiki system, Codewiki shows more lines of source code with different method of displaying the source code, description and sub-description instead of using normal text to be an explanation or comment notation to describe the specific line code which also prevent using duplicate explanation for the source code. As shown in Fig. 6:



codewiki.fd.ise.shibaura-it.ac.jp
javascript/README.md at master · airbnb/javascript
CodeWiki

Semicolons

- 21.1 Yup. eslint: `semi` jscs: `requireSemicolons`
When JavaScript encounters a line break without a semicolon, it uses a set of rules called Automatic Semicolon Insertion to determine whether or not it should regard that line break as the end of a statement, and (as the name implies) place a semicolon into your code before the line break if it thinks so. ASI contains a few eccentric behaviors, though, and your code will break if JavaScript misinterprets your line break. These rules will become more complicated as new features become a part of JavaScript. Explicitly terminating your statements and configuring your linter to catch missing semicolons will help prevent you from encountering issues.

```
const luke = {}
const leia = {}
[luke, leia].forEach(jedi => jedi.father = 'vader')
```

bad - raises exception

```
const reaction = "No! That's impossible!"
(async function meanwhileOnTheFalcon() {}())
```

bad - raises exception

```
function foo() {
  return
    'search your feelings, you know it to be foo'
}
```

bad - returns `undefined` instead of the value on the next line - always happens when `return` is on a line by itself because of ASI!

```
const luke = {};
const leia = {};
[luke, leia].forEach((jedi) => {
const reaction = "No! That's impossible!";
(async function meanwhileOnTheFalcon() {
  ......
}());
```

handle `leia`, `lando`, `chewie`, `r2`, `c3p0`

Good example

Fig. 8: The display of Codewiki system.

## 5. Conclusion

The web-based service has been use as references for developers to find the source code and the explanation. CodeWiki will help the editors and readers to enhance the usage of wiki system more efficiency by enabled the corresponsive between both source code and explanation within the same window. The editing section also provides multiple levels of the style of explanation which editors can use to add more detail of the explanation to the source code and also provide more than one area for entering the source code and the explanation altogether. Moreover, the reading section also provides the source code parallel with the explanation which help the readers to understand the source code easier and save the extra time for scrolling. The CodeWiki system also allows readers to click on the sub-explanation to see more detail on a specific area of code which marked by the editors. As a result, CodeWiki enables editors and readers to reduce scrolling motions and corresponding between the source code and the explanation with the detail on the specific area of source code which marked by the editors.

## 6. References

[1] J. M. Haake, A. Haake, T. Schu¨mmer, M. Bourimi, and B. Landgraf. End-user controlled group formation and access rights management in a shared workspace system. In Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, CSCW ’04, pages 554–563, New York, NY, USA, 2004. ACM.

[2] A. Haake, S. Lukosch, and T. Schu¨mmer. Wiki-templates: Adding structure support to wikis on demand. In Proceedings of the 2005 International Symposium on Wikis, WikiSym ’05, pages 41–51, New York, NY, USA, 2005. ACM.

[3] React, https://reactjs.org/ (accessed 2018-3-15).

[4] Redux, https://redux.js.org/ (accessed 2018-3-15).

[5] Node.js, https://nodejs.org/ (accessed 2018-3-15).

[6] Draft.js https://draftjs.org/ (accessed 2018-3-15).