**Abstract**

With computer science expanding rapidly as a primary career path for many students, it is imperative that students who choose to pursue this field learn good coding techniques. While some sources do exist for this enhancement in proper coding skills, the accessibility to these resources is very limited. Chrome extensions provide the user a way to enhance the functionality of the Chrome web browser. By implementing this software program, this project allows for the user to go to three specific websites and get useful feedback while coding. By combining the chrome extension with extensible message and presence protocol (XMPP) via Openfire, this project passes messages from the front-end extension to the back-end checkstyle and sends back information beneficial to the user.

The primary reasoning behind this project is to allow for easy access to good coding methods. A user study can be further conducted to test for the improvement in students’ abilities to code properly. Students can be assigned to work on some of their projects in the online environment, with this chrome extension implemented, and other projects in their Eclipse environment and the results of their coding can be tested to check for any growth/differences. Using both qualitative and quantitative analysis, the data can be evaluated and conclusions can be drawn as to the extent of the usefulness of this chrome extension.

**Theory and Proposal**

As the use of technology becomes increasingly prevalent in modern day, computer science has become a useful and promising field for students to pursue. However, outside of the classroom environment, it is rare for the budding computer scientist to obtain the necessary aid to understanding proper procedures and coding techniques. There has already been the creation of a style checker called CheckStyle (and other checkers such as Google checks), which checks Java code (and code in other languages) on the any integrated development environments (IDEs) such as Eclipse. By combining the usage of chrome extensions with the checkstlye implementation, which checks for correct coding techniques, this project demonstrates a way for fellow coders to enhance their coding skills in the online coding environment. This paper discusses the development process of this chrome extension and the reasoning behind its development.

**Chrome Extension**

*What is a Chrome Extension?*

Before going over the different parts of this project, it is necessary to have a thorough understanding of a chrome extension. The chrome extension is a small software program that can enhance the functionality of a Chrome browser. The idea behind a chrome extension is that it can work in the backdrop of multiple webpages and can alter the content or other aspects of the Chrome browser for the betterment of the user interactions with these pages.

There are many familiar chrome extensions that users use on a daily basis, each with an individual purpose: some chrome extensions, such as Ad Block are created for the purpose of preventing any harmful ads from affecting a computer, while other chrome extensions such as Grammarly are employed in order to enhance the user’s skill set. The idea behind this particular project, as stated above, is to enhance a student programmers’ coding style.

*Front-End of the Chrome Extension*

In order to create any type of chrome extension, the first file that must be made is called the manifest.json file. This file contains all JavaScript Object Notation (JSON) objects that provide information about the extension. The first object needed is the name of the project, called “Checkstyle WebCoder” because the checkstyle, which is the code style checker, will be used to check Java code on online environment. Next, a description of the project is provided, which is illustrated above along with the version and the manifest\_version, which are 1.0, since this is the first version of the project, and 2.0 respectively. After that, the objects detail other files that are included in this chrome extension. The browser action object is employed because the chrome extension can be used for a plethora of web pages (currently used for three pages but can be expanded for usage in other coding sites). This browser action contains two objects of its own: the default\_icon and the default\_popup.

The default\_icon is the picture of the chrome extension that pops up to the right of the address bar when the user downloads this project. The default\_popup is an HTML page appears when the user clicks on the icon. For this project, the default popup contains the login credentials, so that the user can connect to the backend via Openfire, a server that uses XMPP. Once the user enters his username a password, he can click on the connect button for the connection to be formed.

In order to interact with the web page itself, the manifest.json file must include a “content\_scripts” object. For this project, this object has six other objects (two objects repeated three times). Three of these objects are called “matches” which provide website URLs for webpages the extension can access. The three websites used for this project are compilejava, ideone, and jDoodle. The “\*” that is used in the website URLs specify that anything (http or https) is allowed before each URL name. The other three objects are called “js” and these provide the files which are used to interact with the website along with the jquery file, which allows for the other JavaScript files to use jquery. Since the HTML id’s needed for this project, are the same for the ideone and jDoodle this pair will have the same JavaScript file.

Inside the JavaScript files, the only difference between the two files is the name of the id of the textbox in which the user codes in each of the websites. The rest is the same in that once the document is loaded, the JavaScript file grabs the content inside of the textbox and stores it in a variable called message and sends this message to the background page. In addition, the file uses the chrome.storage option to store the information in the console.log. In order to use this chrome.storage method, the “storage” object must be present inside of the “permissions” object in the manifest file. Lastly, an image is created inside of the textbox, which, when clicked, opens up a new HTML document that has the code that the user wrote in the textbox along with the stylistic errors in separate textboxes.

Next, the manifest file includes the “background” object and inside of this object contains the “scripts” object, which gives the background JavaScript file names. The background object is used for the purposes of short-term storage as well as for socket connections with the server. There are two JavaScript files implemented in the project: strophe and background.

The strophe file is a JavaScript library for writing XMPP clients. It provides a persistent, stateful, two-way connection to an XMPP server. The background JavaScript file provides a lot of different methods to interact with the server. It first uses the strophe.connection method to the Web Socket. Next, it contains the chrome.runtime.onMessage method to listen for messages from the content scripts and packages these messages into two different JSON objects: the from object, which gets the username of the user who logged in using the popup.js, and the editorContents object, which gets the code that the user typed into the textboxes. After, creating this messageBody JSON object, it sends the information to the server using the Strophe.Builder method. In addition, the background JavaScript file includes an onConnect function, which sends a message to the background console with the information about the login connection status. Finally, the background includes a handleMessage, which waits for the server to send information back as a JSON object, parses this message, and calls the sendToContent method. This method sends the message back to the content scripts.

The last two things that the manifest file includes are the “permissions” and “web\_accessible\_resources” objects. The permissions object includes the storage object, as state above, and the activeTab object, which is used for accessing the active chrome tab. The “web\_accessible\_resources” object includes two different files, a picture that pops up in the textboxes of the four websites used in this project. The second file is the HTML document, which pops up when the picture is clicked on. This HTML document contains two textboxes, one that includes the original code that the user typed and a second textbox that includes the checkstyle message that the server sends to the background, which sends it to the content scripts.

*Back-End of the Chrome Extension*

The main part of the back-end of this chrome extension is the MessageBus jar file. When this file is executed, it initializes the XMPP, sets up the servers, checks to see if the server is connected and listens for any messages sent from the background file. Once a message is received by the MessageBus, it parses the message and checks to see if any program (client) is interested in the JSON. The way it identifies any clients is that if any client if looking for the tag “EditorContents”, the MessageBus sends this message to that client and thus it sends this message to the CheckstyleWebCoder jar file, which contains two different java files.

The AFileMessage file contains a “publicClassChecker” method, which takes in a string, and checks for the “public class” or “class” if it doesn’t have “public class” text from the editorContents object. Once it finds this, it stores the word that comes after this. The next method is called the “packageChecker” and it works similarly to the “publicClassChecker”, except that it looks for the word “package” and next word. If there is more than one word (separated by dots), it stores all of these words. For both of the methods, it checks to see if any line is commented out and ignores these words. The “receiveMessage” method, which takes two string arguments as well, calls both of these methods in order to create folders and files with correct names. This method first creates a folder called “Styles”, if it doesn’t already exist, in the “Users/name\_of\_user/” location. Next, it creates a folder with the name of the username of the user logged in inside of the Styles folder. After that it creates x folders with x being the number of Strings in the array returned by the “packageChecker” method. It creates these folders inside of each other. So if there was a package named “hello.main.code”, then it would create a folder called “hello” inside of “src” and a folder called “main” inside of “hello”, and finally a folder called “code” inside of “main”. If the “packageChecker” returned an array of length 0, then it would create one folder inside of “src” called “default”. The file would be created inside the last package name and it takes the name of whatever String the “publicClassChecker” method returns. Finally, it creates a “ACheckStyleRunner” object which is the other class the CheckstyleWebCoder jar file includes.

This “ACheckStyleRunner” contains one method, which takes in the folder in which the text stored in the “receiveMessage” method and takes a configuration file that the Professor Dewan created. This configuration file is downloaded from Professor Dewan’s website and stored in the Styles folder and is run along with the text file, in order to check for stylistic errors and this method returns all of these errors (other style checkers such as Google checks can be run by downloading those files and running those configurations instead). The “receiveMethod” takes these messages and turns them into Strings.

All of these previous messages are called from another “receiveMessage” method in the “AFileMessage” java file. This method first gets the JSON objects from the MessageBus and stores them in strings. It also, takes the information that the “ACheckStyleRunner” returns, packages them into JSON objects and sends it back to the MessageBus jar file. This jar file receives the messages and passes it along to the background JavaScript file, which sends it to the content scripts where the user can view it.

*Chrome Extension Reasoning*

The primary question that arises as a result of this proposed study is regarding the accessibility and general usability of the chrome extension. After all, all this chrome extension does is pass all the messages to the checkstyle, which runs the text to find any stylistic errors passes the information back. By simply adding the checkstyle library to any program written in the Eclipse (or any other IDEs), the user can obtain the same information. Although, this is the case, this chrome extension is written for the average user who wants to code on an online environment, making it increasingly accessible and easy to use for students. In addition, previous checkstyle programs that have been created are not accessible to the general public. Thus, this chrome extension could allow the every day user to quickly grasp proper coding habits in an environment accessible to all users.