

**Product: S.S. Media**

**Client: Doctor Judith E. Rosenbaum**

**Code Inspection Report**



**Black Bear Analytics**

**Abdullah Karim | Colleen DeMaris | Griffin Fluet | James West | Ryan Handlon**

March 17<sup>th</sup>, 2020

## Revision History

Version Number	Release Date	Description
Version 1.0	03/17/2020	Original Release

*S. S. Media*  
Code Inspection Report

**Table of Contents**

	<b><u>Page</u></b>
Cover Page	<b>1</b>
Revision History	<b>2</b>
Table of Contents	<b>3</b>
Abstract	<b>4</b>
<b>1. Introduction</b>	<b>5</b>
1.1 Purpose of This Document	<b>5</b>
1.2 References	<b>5</b>
1.3 Coding and Commenting Conventions	<b>5</b>
1.4 Defect Checklist	<b>5</b>
<b>2. Code Inspection Process</b>	<b>7</b>
2.1 Description	<b>7</b>
2.2 Impressions of the Process	<b>7</b>
2.3 Inspection Meetings	<b>7</b>
<b>3. Modules Inspected</b>	<b>9</b>
<b>4. Defects</b>	<b>15</b>
<b>Appendix A – Coding and Commenting Conventions</b>	<b>18</b>
<b>Appendix B – Peer Review Sign-off</b>	<b>22</b>
<b>Appendix C – Document Contributions</b>	<b>23</b>

## **S. S. Media Abstract**

Social media has grown to be an extremely large part of society in recent years. Because of this, a need to gather data from it has also arisen. Black Bear Analytics is creating a tool that allows researchers to scrape public data from posts on Twitter and Instagram. This tool was requested by Doctor Judith Rosenbaum of UMaine's Department of Communication and Journalism. The S.S. Media will have an easy-to-navigate interface that allows users to switch between scraping public posts on Instagram and Twitter. The *New Search* page gives the user an option of either an advanced search or a basic search. The basic option searches by specified hashtags, locations, or phrases, and an acceptable start and end date to check with each post scraped, while an advanced search has the same general functions as the basic search, but adds the ability to search for more than one topic (hashtag, location, phrase), as well as run the search once every specified amount of time. The tool will store a number of scrape requests, so that users can either inspect a previous scrape request or request it again. Once the scraping is completed, the data will be downloaded to the user's computer in the form of a .csv file, and images will be stored in a corresponding folder. The outcome of this project will hopefully be that researchers like Doctor Rosenbaum can have a tool that will enable them to easily search for data and trends on social media sites.

# 1. Introduction

The purpose of this section is to introduce the reader to both this document and the Code Inspection Review, detailing introductory information that will be useful for the reader to know.

## 1.1 Purpose of This Document

The purpose of this document is to report the conclusion of an egoless code review conducted with defined coding conventions. Black Bear Analytics met several times to refine the coding conventions in this document to present to the reader in conjunction with defects identified during code review.

## 1.2 References

Black Bear Analytics SRS Document

Black Bear Analytics SDD Document

## 1.3 Coding and Commenting Conventions

The coding and commenting conventions defined in Appendix A seek to reduce readability errors within the code and enhance the future code review process. Black Bear Analytics took initiative in creating the coding conventions based on prior experience and models seen as the industry standard. Industry standard conventions can be found in the following references:

Admin. (2018, March 28). *Coding Standards and Best Practices*. Aversan.  
<http://www.aversan.com/coding-standards-and-best-practices-2/>.

*Coding standards and guidelines*. (2019, July 2).  
<http://www.geeksforgeeks.org/coding-standards-and-guidelines/>.

Black Bear Analytics did not modify any of the coding conventions listed in reference. The conventions are either created by Black Bear Analytics or adopted from the listed references. Please see Appendix A for coding and commenting conventions used in our project.

## 1.4 Defect Checklist

Below is an overview of the defects Black Bear Analytics uncovered during code review.

Table 1.1 - Defect Checklist by Category

Num.	Name	Module No.	Convention	Logic Flaw	Security	Comments
1.	Login User/Pass Scrub	1				
2.	Chrome Data	7				

	Breach Popup					
3.	Backend User Credentials	4-21				
4.	Emoji Scrape Errors	1				
5.	Archived Write Data Loss	1				
6.	Category 2 Redundancy	2				
7.	Browser Waiting	2				
8.	Sleeps In Code	2,3				
9.	Comment Conventions	All				
10.	Code Conventions	All				
11.	Fetch requests deformed	7-21				
12.	Login Auth Fails	7				
13.	Database displays	5				
14.	Scrape Twitter not correctly parsing input	6				
15.	Scraping more tweets than requested	1				
16.	Checking for user already in database	6				
17.	Deployment Pipeline Breakage	23				

## 2. Code Inspection Process

The purpose of this section is to give an overview of the Code Inspection Process that Black Bear Analytics took in it's most recent Code Review, as well as their general impressions of it.

### 2.1 Description

We started the review with our backend system and then switched back and forth with the frontend discussing the coding conventions to follow versus what was done. This gave each of the code developers ideas as to what refactoring had to be done. We listed those coding conventions here in Appendix A and continued to look for logical errors in the Instagram/Twitter Scraping. The UI was tackled after and we uncovered any defects a user might see. We then looked at the restAPI for defects in communication with our database and front-end to back-end systems.

In each review module, we began with naming conventions and then commenting conventions. Then we moved on to walking through the sections of the code, discussing what each piece does and what needs to be done on the backend. This process was repeated during the whole review to uncover any defects. We also took a look at our deployment and identified its degradation in the review.

### 2.2 Impressions of the Process

The code inspection process was very useful in bringing the team together, getting to walk through each piece of the project was insightful. Each team member was able to add valuable input towards finding bugs, creating standards to follow, and what pieces required refactoring. Now that we know how a code inspection works, we would have created commenting and naming conventions from the start.

The worst parts of our program are the pieces we still have not worked on yet as there is still a lot of work to be finished. We also do not have the Login Page verification working, a critical component for accessing the application. Our main issues revolve around code not being implemented yet.

### 2.3 Inspection Meetings

Below is a list of all inspection meetings Black Bear Analytics committed to for the review process:

03.11.2021: Met for code inspection.

Location: virtual, on discord

Time Started: 5:30pm

Time Ended: 8:30pm

Participants: Abdul, Colleen, Griffin, James, Ryan.

Abdul: scribe for commenting conventions

Colleen: scribe for meeting details

Griffin: reader for his code, inspector for other code

James: inspector

Ryan: reader for his code, inspector for other code

03.15.2021: Met for code inspection. Location: virtual, on discord

Time started: 5pm

Time ended: 7pm

Participants: Abdul, Colleen, Griffin, James, Ryan

Abdul: reader for his code, inspector for other code

Colleen: scribe, inspector

Griffin: inspector

James: reader for his/colleen's code, inspector for other code

Ryan: inspector



### 3. Modules Inspected

The purpose of this section is to provide the reader with a brief understanding of all modules in the current software project that were inspected, and any relevant information that should accompany them.

The following modules appear in the SDD: URLPostExtractor, KeywordURLExtractor, User Interface, DOM Queue, and Page Processor. URLPostExtractor and KeywordURLExtractor differ from in the SDD, and are described in their modules. The User Interface is different from the SDD because we broke down the modules more. Instead of the User Interface, we have individual modules for each page that will be added into the S.S. Media.

#### Module 1: TweetExtractor.py

- Incomplete
- Projected Completion Date: 03.25.21
- Need to finish: Find a less resource intensive method of collecting tweets.
- Description: This module creates the query to send to Twitter's API based on input of hashtags, locations, and phrases. The returned tweets from the API call will have the data points of interest scraped and if a form of media exists in the tweet, it will be downloaded. Each tweet written to the CSV file receives an ID that will match the name of the media associated with it. In the case where there are more than one media in a tweet, there is a file for each one and have an incrementing value concatenated to the end of the file name. The result of this module will be zipped archive with the scraped data points and media.
- Difference from SDD: This module differs from the SDD since it is acting as both the KeywordURLExtractor and PostExtractor. When designing the SDD it was not determined if Twitter's API was going to be used, so the document was written in the context of scraping based off of HTML. This way of scraping Twitter makes it much easier to scrape more tweets than HTML based scraping. Since we are instead querying Twitter for the tweets that have a match based on hashtags, location, phrase, and even date ranges.

#### Module 2: InstagramKeywordURLExtractor.py

- Incomplete
- Projected Completion Date: 03.25.21
- Need to finish: This module currently works but needs to be refactored. The current scraping process needs to be modified for better handling of when the bottom of Instagram's explore page is reached, all sleep function calls removed, browser.implicitly\_wait() needs to be rethought to account for slow internet, and the link extraction should be writing to the URLFrontier.txt file during scraping rather than after.
- Description: This module opens up a headless chrome browser navigated by Selenium. It logs into Instagram, and then visits the Instagram explore page for a specific hashtag or

location. From there it scrolls down the explore page collecting links to posts and populating them into a text document.

#### Module 3: PostExtractor.py

- Incomplete
- Projected Completion Date: 04.10.21
- Need to finish: Using multiple accounts for each user for when an account gets blocked from the explore page. Also need to implement error handling when an Instagram post is deleted and we try scraping it.
- Description: This module grabs the links from the URL\_Frontier and scrapes the wanted data points from an Instagram post and writes them to a CSV file. The image or video from that post is downloaded to a media directory. Each post written to the CSV file receives an ID that will match the name of the media associated with it. The result of this module is a zipped archive with the CSV file and media directory.

#### Module 4: ScrapeManager.py

- Incomplete
- Projected Completion Date: 04.10.2021
- Need to finish: Work on this module has not been started yet.
- Description: System to manage each scrape within a thread to allow for multiple scrapes to be going at the same time. Scrapes will start and end here and have an id based off of the user that started the scrape. This will allow us to pass back information like the number of tweets scraped during a running scrape.
- Difference from SDD: This is a new module that we found a need for since we overlooked the need for managing the scrapes when a user starts one.

#### Module 5: User Database Schema

- Incomplete
- Projected Completion Date: 04.20.2021
- Need to Finish: Adding columns, foreign keys, and tables to reflect recent scrape history.
- Description: This is the database used to store user information. It fits into the SDD at the login section where we are storing user information. Haven't finished adding necessary columns to store all user information. We need to include a table or another column storing the users recent searches. There may be other information that needs to be stored that we haven't come across yet and this is an unknown risk to storing live user information.

#### Module 6: Endpoints

- Incomplete
- Projected Completion Date: 4.15.2021

- Need to Finish: Adding endpoints that the front-end can use to get information and the backend can use
- Description: This is the connection between the frontend and the backend. This fits into our SDD where any processing makes a backend or frontend call. The API we've created is missing some endpoints that are required for accessing the database. We lack the functionality to edit a user's entry, which will be necessary when saving recent searches to the user database which also does not have an endpoint. There are also many endpoints that will need to be created as more of the project is implemented such as an instagram scrape endpoint and other endpoints that facilitate the communication between front-end and backend.

#### Module 7: Login Page

- Incomplete
- Projected Completion Date: 04.10.2021
- Need to Finish: The connection to the restAPI we have is configured but does not send the user to the HomePage based on verification or give a message barring them from entry because of invalid credentials.
- Description: This is the landing page for all users to start at. It fits into the start of our design from the SDD because users need a place to input credentials for a secure connection. All of the styling is done for this page along with the input sections that the user can enter information into.

#### Module 8: Search Criteria

- Incomplete
- Projected completion date: 04.10.2021
- Need to finish: Creating CSS and ReactJS files for both the advanced and the basic options. Need to connect to the database in order to store the search criteria. Need to connect to the scraper back end to initiate the search with the filled out information.
- Description: This page displays the options to fill out a scrape search. The user can choose between the different platforms, Twitter and Instagram, in a drop down menu. The user can also choose whether they would like a basic or advanced search. The advanced search allows the user to search using time intervals, from one date to the next every given amount of time. The advanced search also includes everything on the

#### Module 9: Edit Users Page

- Incomplete
- Projected completion date: 04.10.2021
- Need to finish: Creating the CSS and ReactJS files. Need to then connect to the database so that it can display the list of all accounts.

- Description: This page is the Edit Users page for admin users of the S.S. Media. It is navigated to from the Admin Settings page. This page displays a list of accounts with information on whether they are an admin or not. The admin looking at the account list can change who is an admin and who is not. The admin user can also ban accounts from this page.

#### Module 10: Home Page

- Incomplete
- Projected Completion Date: 03.18.2021
- Need to Finish: Adding buttons and navigation
- Description: This is the page that the user can go to for accessing their scrape history, conducting a new search, or looking at current searches. This is the hub where users can access nearly all other pages in our website and currently only has one button to start a new search on it.

#### Module 11: Searching Page

- Incomplete
- Projected completion date: 04.10.2021
- Need to finish: Creating CSS and ReactJS files. Need to connect the page to the back end extractor, to display the number of posts found. Also need to be able to display the scraping criteria above the number of posts found.
- Description: This page is displayed after a scrape is initiated. It displays the scrape criteria that the user put in, as well as a loading circle with the number of posts found and scraped. The user has the options to either end the search, start a new search, go home, or enter the settings.

#### Module 12: Admin Settings Page

- Incomplete
- Projected completion date: 04.10.2021
- Need to finish: The CSS and ReactJS have been created, need to connect the page with the database to display the user accounts that are pending and must be approved. Must also create functionality for each button. Need to hook up the page to a .config file in order to store saved settings. Need to be able to navigate to the page.
- Description: This page is the settings page for admin users of the S.S. Media. The page gives users the ability to view any contact requests that were sent in, edit the users that are allowed access and what type of access they are allowed, deactivate the account, or change the scrape history, advanced search, and email notification toggles. This page displays accounts that must be approved so that the admin user can approve them or delete their requests. Users can also logout from this page.

### Module 13: Settings Page

- Incomplete
- Projected completion date: 04.10.2021
- Need to finish: The CSS and ReactJS have been created, need to connect the page to a .config file in order to store saved settings. Need to be able to navigate to the page.
- Description: This page is the settings page for non-admin users of the S.S. Media. The page gives users the ability to toggle their scrape history visuals on/off, the advanced search on/off as default, and email notifications on/off. The user can deactivate their account from this page, or log out.

### Module 14: Account Requested Page

- Incomplete
- Projected completion date: 04.10.2021
- Need to finish: Creating CSS and ReactJS files. Need to be able to navigate to the page.
- Description: This page is the page that is displayed when someone first makes a new account. Once the account information is filled out, this page is displayed and the user can only go back to the login page until their account is approved by an admin.

### Module 15: Register Account Page

- Incomplete
- Projected completion date: 04.10.2021
- Need to finish: Creating the CSS and ReactJS files. Need to be able to navigate to the page. Need to connect it to the database in order to store the account information.
- Description: This is the page that displays when a user clicks the Create button on the login page. It takes in an email, name and password, and sends it to the database.

### Module 16: Current Searches Page

- Incomplete
- Projected completion date: 04.10.2021
- Need to finish: CSS styling and ReactJS files, Linking using React-Router.
- Description: This is the page that displays the searches that are currently being run. The user can choose filters from a drop down menu as to what previous searches are displayed.

### Module 18: Reset Password Notification Page

- Incomplete
- Projected completion date: 04.20.2021
- Need to finish: CSS styling and ReactJS files, link to home login page.
- Description: This page displays a message that an email has been sent to reset the user's password. It has one button, Home, that brings the user to the login page.

#### Module 19: Reset Password Message Page

- Incomplete
- Projected completion date: 04.20.2021
- Need to finish: CSS styling and ReactJS files, link to function that sends email to the user to reset the password. Link to Reset Password Notification Page to display message.
- Description: this page has one input that takes the user's email address. It has a Go Back button that directs back to the login page. It has a submit button that will send the email address to the function that will send an email to that address. The submit button will also redirect the user to the notification page.

#### Module 20: Register Account Success Page

- Incomplete
- Projected completion date: 04.20.2021
- Need to finish: CSS styling and ReactJS files.
- Description: This page is displayed to inform the user of their new account when the user has entered valid input on the Register Account Page, and clicks the Create Account button.

#### Module 21: View Contact Requests Page

- Incomplete
- Projected completion date: 04.20.2021
- Need to finish: CSS styling and ReactJS files.
- Description: This page is accessed from an Admin user account and displays the contact requests made by non-users that filled out the Contact Us Page's form.

#### Module 23: Full-Stack Deployment Pipeline

- Incomplete
- Projected Completion Date: 03.20.2021
- Need to Finish: Adding deployment details for Back-end product
- Description: This is the pipeline we are using to deploy our software to AWS as seen in the SDD.

## 4. Defects

The purpose of this section is to list all defects found during formal code review and coding convention definition and provide information about them. They are numbered in accordance with the defect checklist (Table 1.1):

1. Login User/Pass Scrub
  - Module: 1 - TweetExtractor.py
  - Description: Not scrubbing username and password from the Login page once entered.
  - Category: Security
2. Chrome Data Breach Popup
  - Module: Unknown, possibly 7- Login Page
  - Description: On entering a username/password combination and clicking submit, Chrome displays a popup that states there is a data breach.
  - Category: Security
3. Backend User Credentials
  - Modules: 4 through 21
  - Description: We need user credentials to be passed from the front-end process to the backend so that multiple users can access our application at the same time.
  - Category: Logic Error
4. Emoji Scrape Errors
  - Module: 1 - TweetExtractor.py
  - Description: If a tweet being scraped contained an emoji, its content was not able to be written to the CVS file. This could potentially be caused by not having an encoding type specified.
  - Category: Correctness
5. CSV Archive Data Loss
  - Module 1 - TweetExtractor.py
  - Description: The last few rows of data in the CSV file are cut off when extracted. This defect only takes place after the extraction of the CSV file.
  - Category: Logic Flaw
6. Category 2 Redundancy
  - Module 2 - InstagramURLKeywordExtractor.py
  - Description: Extra functionality was implemented allowing for scraping parameters that were not supposed to be implemented, specifically the scraping of user profiles. This was not supposed to be implemented and extra complexity that is not needed. This should be removed.
  - Category: Logic Flaw
7. Browser Waiting
  - Module 2 - InstagramURLKeywordExtractor.py

- Description: The InstagramURLKeywordExtractor uses the setting browser.implicitly\_wait(5) when using selenium to navigate the headless browser. This setting makes the browser wait 5 seconds for a response when fetching something from a page. This should be changed to allow for slow internet so errors do not get thrown when they shouldn't be.
  - Category: Convention
8. Sleeps in Code
- Module 3 - PostExtractor.py and Module 2 - InstagramURLKeywordExtractor.py
  - Description: Sleep function calls were used in both of these files in spots where the code needed to do something, such as try catch blocks. These should be replaced with pass function calls.
  - Category: Logic Flaw
9. Comment Conventions
- All Modules
  - Description: Comment conventions had not been created until the beginning of code inspection. This meant all code written was not following a standardized commenting convention and needs to be refactored.
  - Category: Comments
10. Code Conventions
- All Modules
  - Description: Coding conventions had not been created until the beginning of code inspection. This meant all code written was not following a standardized coding convention and needs to be refactored.
  - Category: Convention
11. Fetch Requests deformed
- Modules: 7 through 21
  - Description: Our fetch requests currently don't produce enough information for our restAPI to consume, limiting us from passing vital information such as user credentials. We need to modify the fetch requests to support all incoming data and information that the backend processes require.
  - Category: Logic Flaw
12. Login Auth Fails
- Module 7 - LoginPage
  - Description: Our login page currently bypasses user authentication and gives access to the site without credentials. We need to render new pages conditionally to bypass this issue.
  - Category: Logic Flaw, Security
13. Database Displays
- Module 5 - Endpoints



- Description: Our database is currently displayed to everyone if a certain endpoint for the restAPI is hit regardless of credentials. This functionality needs to be masked or removed prior to release.
  - Category: Security
14. Scrape Twitter Not Correctly Parsing
- Module 6 - Endpoints
  - Description: When the front end passes information to the scrape twitter endpoint, it doesn't correctly parse it into the format specified when passing it into the Twitter query builder. This needs to be fixed to correctly parse the information.
  - Category: Logic Flaw
15. Scraping More Tweets Than Requested
- Module: 1 - TweetExtractor.py
  - Description: Call to Twitter's API not returning the requested amount of tweets. In the case of this defect that was observed double the number of tweets were returned than requested.
  - Category: Logic Flaw
16. Checking for User Already in Database
- Module 5 - Database Schema
  - Description: Our check to see if a user is already in the user database prior to account creation doesn't happen. This needs to happen or we will have duplicate user information within the database which leads to faulty coding and ambiguity.
  - Category: Logic Flaw
17. Deployment Pipeline Breakage
- Module 23 - Pipeline
  - Description: Our deployment pipeline needs to be updated with new dependencies that the team is using so that we can deploy our software to an AWS instance and release it to our client.
  - Category: Logic Flaw

## Appendix A - Coding and Commenting Conventions

The following information defines the coding conventions Black Bear Analytics has taken to improve its product code by increasing readability and coding efficiency. This is a comprehensive list of all standards the team has set and are agreed upon by the whole team. During code review, these standards will be used to define convention errors for both code and comments. Note that the symbol '<-' and anything after it are there for the reader's understanding of the convention and are not part of the convention itself.

### Conventions for Variables

Variables in Python must start with the type of data that the variable will hold. This is followed by an underscore and a meaningful name representing what is stored in it. Note that the name is using snake conventions, where \_ separates words. Example:

```
l_links <- representing a list that contains links
re_email_contact <- representing a regex to match emails
```

Variables in React must start with the type of data that the variable will hold. This is followed by an underscore and a meaningful name representing what is stored in it. Note that the name is using camel-case conventions, where each new word is uppercased. Example:

```
i_thisVariable <- representing an integer variable
```

Specifically for React tag attributes, we define their names following the React variable naming convention. Example:

```
className = "s_loginContainer" <- where the keyword is at the end
```

### Commenting Conventions

When commenting Python functions, triple quotes are used to wrap the comment and will be the first lines within the function. Included in this comment must be: what the function accomplishes (Description), explained parameters (Arguments), and the output of the function (Output). Example:

```
def InstagramURLExtractor():
    """
    Description:

    Arguments:

    Outputs:
    """
```

When commenting React functions, a multiline comment is used and will be placed above the function. Included in this comment must be: what the function accomplishes (Description), explained parameters (Arguments), and the output of the function (Output). Example:

```
/*
 * What it accomplishes:
 *
 * Arguments:
 *
 * Output:
 *
 */
handleHello() {
    console.log("Hello World!");
}
```

### Comments for Classes

In Python, comments at the top of classes and at the top of files must start and end with triple quotes and also who created it initially, the date of creation, the version, and detail what the class does. Example:

```
"""
Name: Abdul Karim
Date Created: 01/01/01
Version: 1.0
Description: foobar
"""
```

In React, comments above classes and the top of files must be wrapped in a multi-line comment and detail who created it initially, the date of creation, the version, and what the class does. Example:

```
/*
 * Name: Abdul Karim
 * Date Created: 01/01/01
 * Version: 1.0
 * Description: foobar
 */
```

In CSS, comments at the top of files must be wrapped in a multi-line comment and detail what page the CSS is for, who created it initially, the date of creation, and the version. Example:

```
/*
 * Name: Abdul Karim
```

```
* Date Created: 01/01/01
* Version: 1.0
* Description: foobar
*/
```

### Inline Comments

Python inline comments must be placed above the line of code it is documenting. Comment must start with a “#” followed by a single space and then a meaningful description of what the code accomplishes. Example:

```
# Object foo calls bar
o_foo.bar()
```

When first initializing a variable in Python, it is appropriate to have the inline comment to the right of the assignment. Example:

```
i_bar = 0      # This is the bar, initialized to 0
```

React inline comments must be placed above the line of code it is documenting. Comment must start with “//” followed by a single space and then a meaningful description of what the code accomplishes. Example:

```
// Object foo calls bar
o_foo.bar()
```

When first initializing a variable in React, it is appropriate to have the inline comment to the right of the assignment. Example:

```
i_bar = 0      // This is the bar, initialized to 0
```

When commenting in React’s render function any comment must start with “{/\*” and end with “\*/}”. Example:

```
{/* Object foo calls bar */}
o_foo.bar()
```

### Function and Class Naming

When naming functions within Python, the return type of the function must be specified with a character followed by an underscore and then the function name. Example:

```
def i_function_name(): <- This function returns an integer
    return 0
```

In the case where the function will be for internal use only, the first character must be a single underscore followed by the described convention above. Example:

```
def _s_function_name(): <- This function is internal and returns a string.  
    return "foobar"
```

When naming functions within React, the return type of the function must be specified with a character followed by an underscore and then the function name. After the functions parameters, there must be a single space before the open curly bracket. Example:


```
i_function_name() { <- This function returns an integer  
    return 0  
}
```

In the case where the function will be for internal use only, the first character must be a single underscore followed by the described convention above. Example:

```
_s_function_name() { <- This function is internal and returns a string.  
    return "foobar"  
}
```

## Appendix B – Team Review Sign-off

All members of Black Bear Analytics have gone through the code review process and agree on both the defined coding conventions in this document and the code review defect results. We have also received a sign-off from the client, approving this document which means they have reviewed and have accepted what is listed above until fixed.

Name	Signature	Date
<b>Customer:</b> <u>Judith E. Rosenbaum</u>		<u>03/16/21</u>
<b>Comments:</b>		
<b>Team:</b>		
<u>Ryan M. Handlon</u> <b>Comments:</b>		<u>March 15th, 2021</u>
<u>Abdullah I. Karim</u> <b>Comments:</b>		<u>March 15th, 2021</u>
<u>Griffin L. Fluet</u> <b>Comments:</b>		<u>March 15th, 2021</u>
<u>Colleen DeMaris</u> <b>Comments:</b>		<u>March 15th, 2021</u>
<u>James West</u>		<u>March 15th, 2021</u>

## **Appendix C - Document Contributions**

Abdullah, Colleen, Griffin, James, and Ryan all contributed equally (20%) to this document. Each member showcased their code for review while others provided egoless comments on how to improve the code. In drafting this document each member listed defects they had, defects that were found during review, and commenting conventions that improve the readability and efficiency of our code. Each member also contributed to the refactoring of product code in light of defects and underutilized coding conventions.