

**Product: S.S. Media**

**Client: Doctor Judith E. Rosenbaum**

**System Design Document**



**Black Bear Analytics**

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

November 10, 2020

## Revision History

Version Number	Release Date	Description
Version 1.0	11/10/2020	Original Release

# Table of Contents

	<u>Page</u>
<b>Cover Page</b>	<b>1</b>
<b>Revision History</b>	<b>2</b>
<b>Table of Contents</b>	<b>3</b>
<b>1. Introduction</b>	<b>4</b>
<b>1.1 Purpose of This Document</b>	<b>4</b>
<b>1.2 References</b>	<b>4</b>
<b>2. System Architecture</b>	<b>5</b>
<b>2.1 Architectural Design</b>	<b>5</b>
<b>2.2 Decomposition Description</b>	<b>8</b>
<b>3. Persistent Data Design</b>	<b>11</b>
<b>3.1 Database Descriptions</b>	<b>11</b>
<b>3.2 File Descriptions</b>	<b>13</b>
<b>4. Requirements Matrix</b>	<b>16</b>
<b>Appendix A – Agreement Between Customer and Contractor</b>	<b>17</b>
<b>Appendix B – Peer Review Sign-off</b>	<b>18</b>
<b>Appendix C – Document Contributions</b>	<b>19</b>

# 1. Introduction

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

## 1.1 Purpose of This Document

The purpose of the Software Design Document (SDD) is to clearly outline the system design agreed upon by both the client (Doctor Rosenbaum) and Blackbear Analytics for the S.S. Media scraping tool. The Software Design will meet all requirements specified in the Software Requirements Specification (SRS). This document will cover all system design details. This includes an Architectural Design Diagram (ADD), a Technology Architecture Diagram (TAD), a Decomposition Description, Database Descriptions, File Descriptions, and a Requirements Matrix.

The intended readership of this document includes Developers, Testers, the Client, and End Users. Developers will review this document to more fully understand the system architecture and to help guide their efforts in the system's creation. Testers can use this document to help guide them in their testing of the application helping guarantee proper functionality of the product. The Client will read this document to have an understanding of the design of the product commissioned as well as guarantee that their development team understands what is needed out of the product. The end user would review this document to better understand the product's system design.

## 1.2 References

Banga, S. (2020). *What is Web Application Architecture? Components, Models, and Types*. Hackr.io.

<https://hackr.io/blog/web-application-architecture-definition-models-types-and-more>.

Fulton, J. (2018, December 12). *Web Architecture 101*.

<https://engineering.videoblocks.com/web-architecture-101-a3224e126947>.

Kerins, I. (2019, July 4). Solution Architecture Part 5: Designing A Well-Optimised Web Scraping Solution.

[https://blog.scrapinghub.com/solution-architecture-part-5-designing-a-solution-estimating-re source-requirements](https://blog.scrapinghub.com/solution-architecture-part-5-designing-a-solution-estimating-re-source-requirements).

Wikimedia Foundation. (2020, October 29). *Web crawler*. Wikipedia.

[https://en.wikipedia.org/wiki/Web\\_crawler](https://en.wikipedia.org/wiki/Web_crawler).

### Other References:

Black Bear Analytics SRS Document

UI Design :

[https://www.figma.com/file/qc0MoZOfnAJrwfYVoisxcN/BlackBearAnalytics\\_Mockup?node-id=0%3A1](https://www.figma.com/file/qc0MoZOfnAJrwfYVoisxcN/BlackBearAnalytics_Mockup?node-id=0%3A1)

AWS : <https://aws.amazon.com/>

## 2. System Architecture

The system architecture describes the flow, processing, and storage of information within the S.S. Media product. The architecture below includes an Architecture Design Diagram for a Scraper System as well as an accompanying Technology Architecture Diagram which illustrates the technology being used at each step in the design process. A Design Class Diagram is included in Section 2.2 which illustrates the object-oriented functionality of S.S. Media with descriptions of each step in the process.

### 2.1 Architectural Design

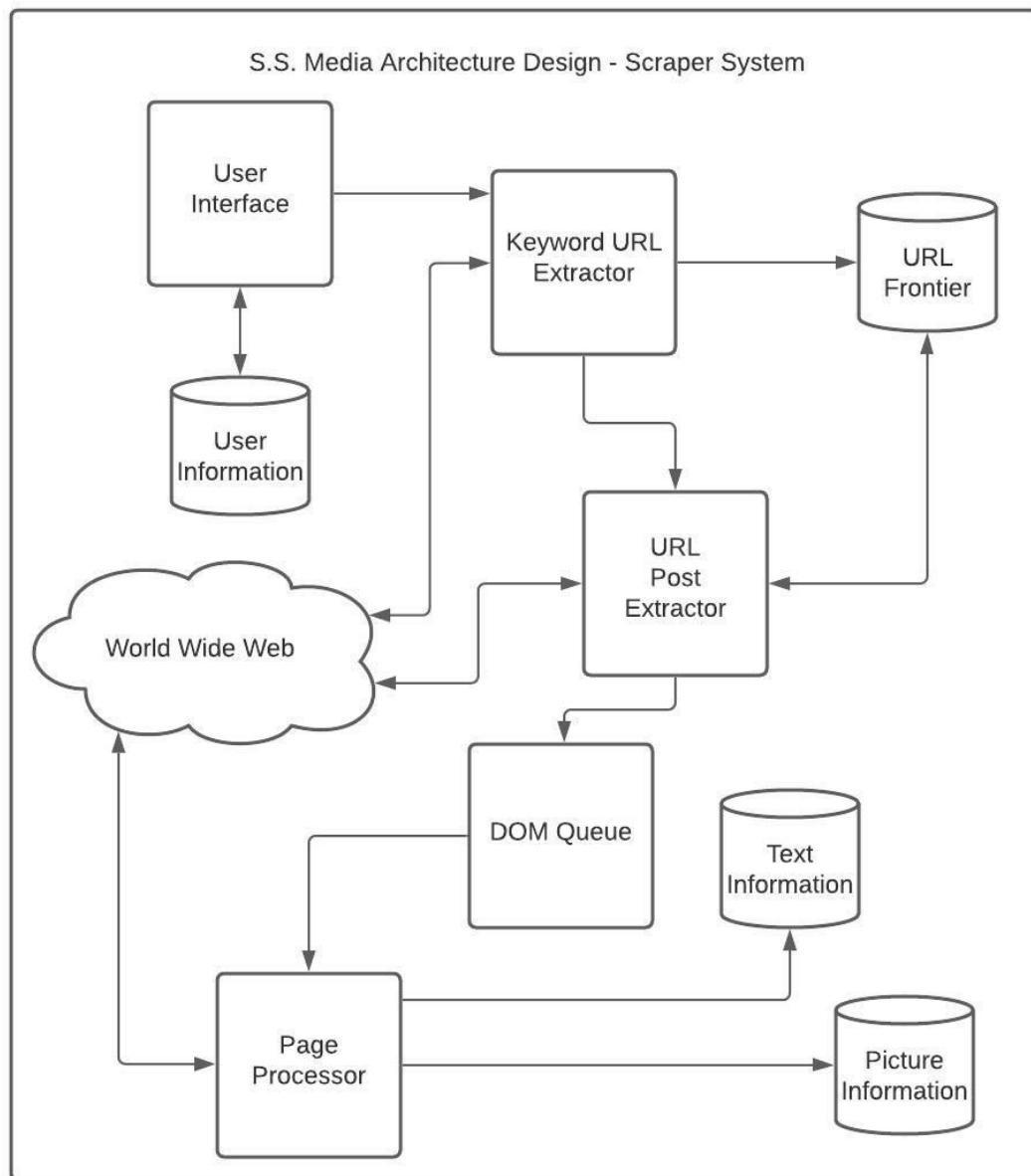


Figure 2.1: S.S. Media Architecture Design

Figure 2.1 shows the S.S. Media Architecture Design. As can be seen in the diagram, the Architecture starts with the User Interface, which manages user information and sends a requested scraped to the Keyword URL Extractor. The Keyword URL Extractor populates the URL Frontier, with URLs needed to be scrapped. From there the URL Post Extractor takes posts from the URL Frontier and creates Document Object Model (DOM) Objects which are added to the DOM Queue. The DOM Queue one by one sends DOMs to the Page Processor. There, the DOM's get parsed for relevant information and which is added to the text and picture information databases.

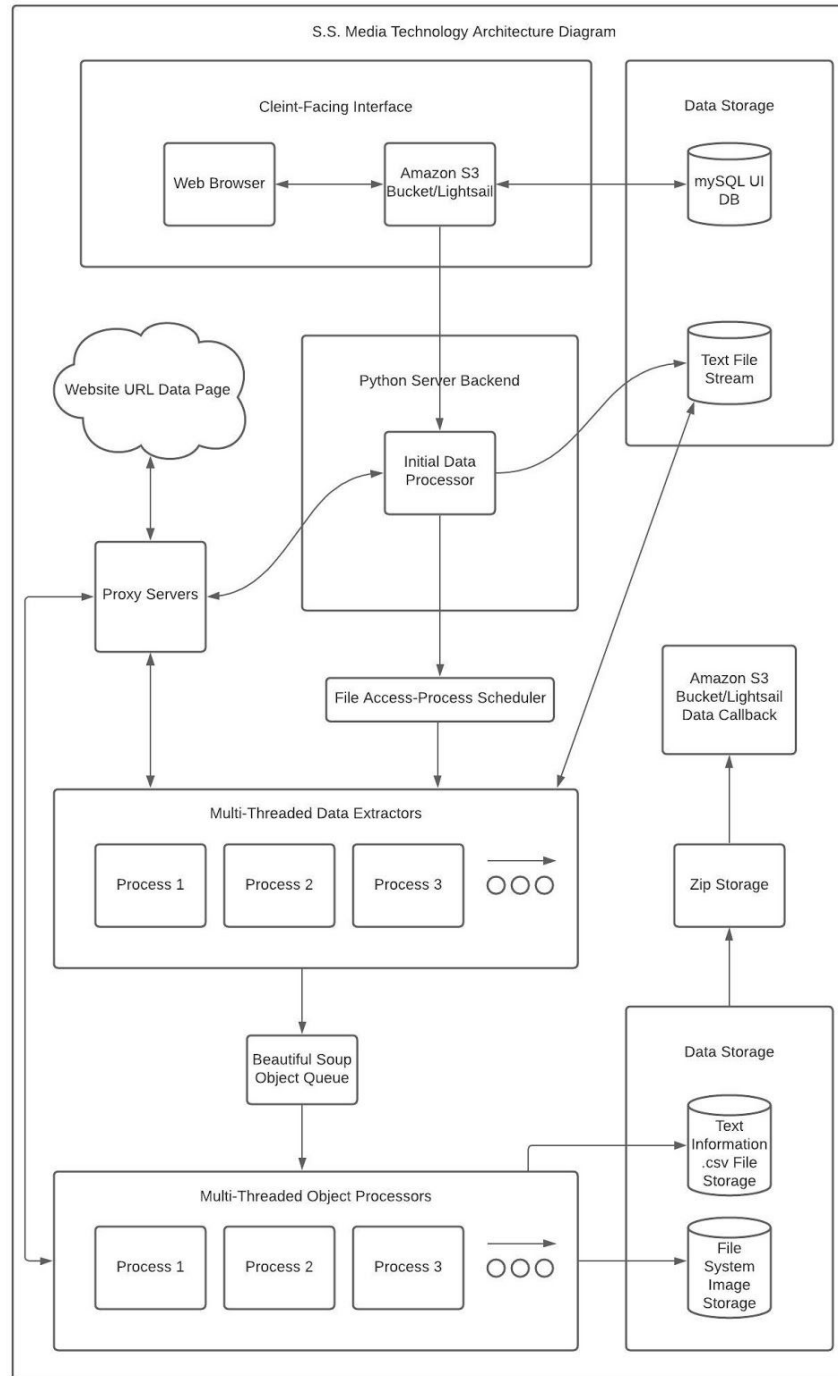


Figure 2.2: S.S. Media Technology Architecture Diagram

Figure 2.2 depicts the specific technologies that play into each part of S.S. Media's design. The user interface a client sees is accessed through a browser hosted through Amazon Web Services (AWS). The AWS will be a Python Web App service (through the Lightsail service) that the system will use to create a page where information can be input and dynamically output. AWS also provides an S3 bucket service that allows for object storage which can be used for data storage at runtime.

Any user inputs flow through the system into a Python backend where Initial data processing occurs and the URL frontier is made into a text file which the application uses to get each post's information. Information then flows through Data Extractors which create the Beautiful Soup Objects (Document Object Models, or DOMs) and are Queued up for multiple processors to then grab and process information from. DOMs are a representation of a post that processors can use to extract post information. Note that as Processors gather information, the product connects to web pages via a proxy to maintain the integrity of the root IP should an IP ban be issued by a social media platform.

Information is then stored into .csv files if they are text and into an image file system otherwise. The storage units are zipped into a file that the client can access via the user interface.

## 2.2 Decomposition Description

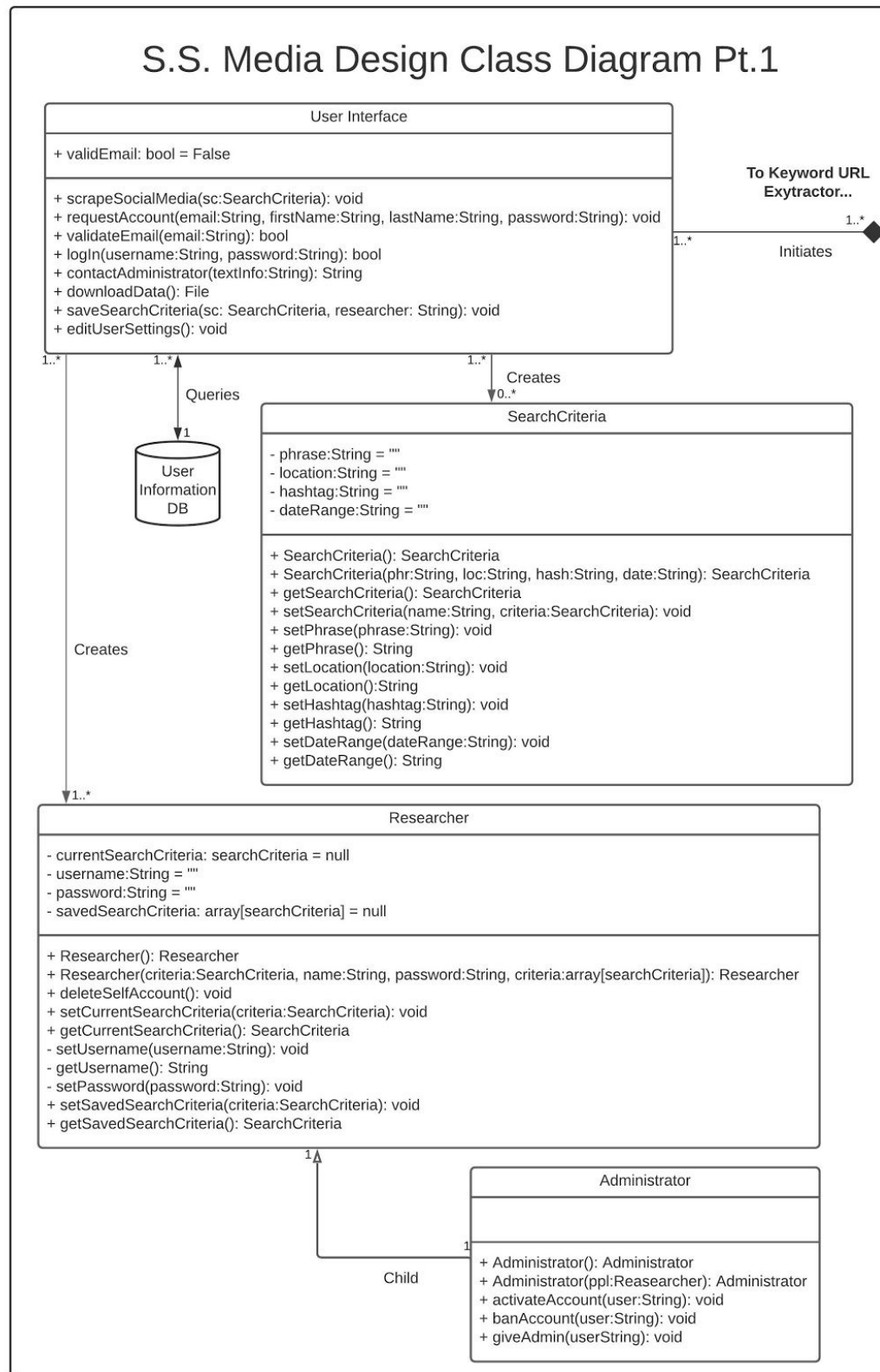


Figure 2.3: S.S. Media Design Class Diagram Part 1



## S.S. Media Design Class Diagram Pt.2

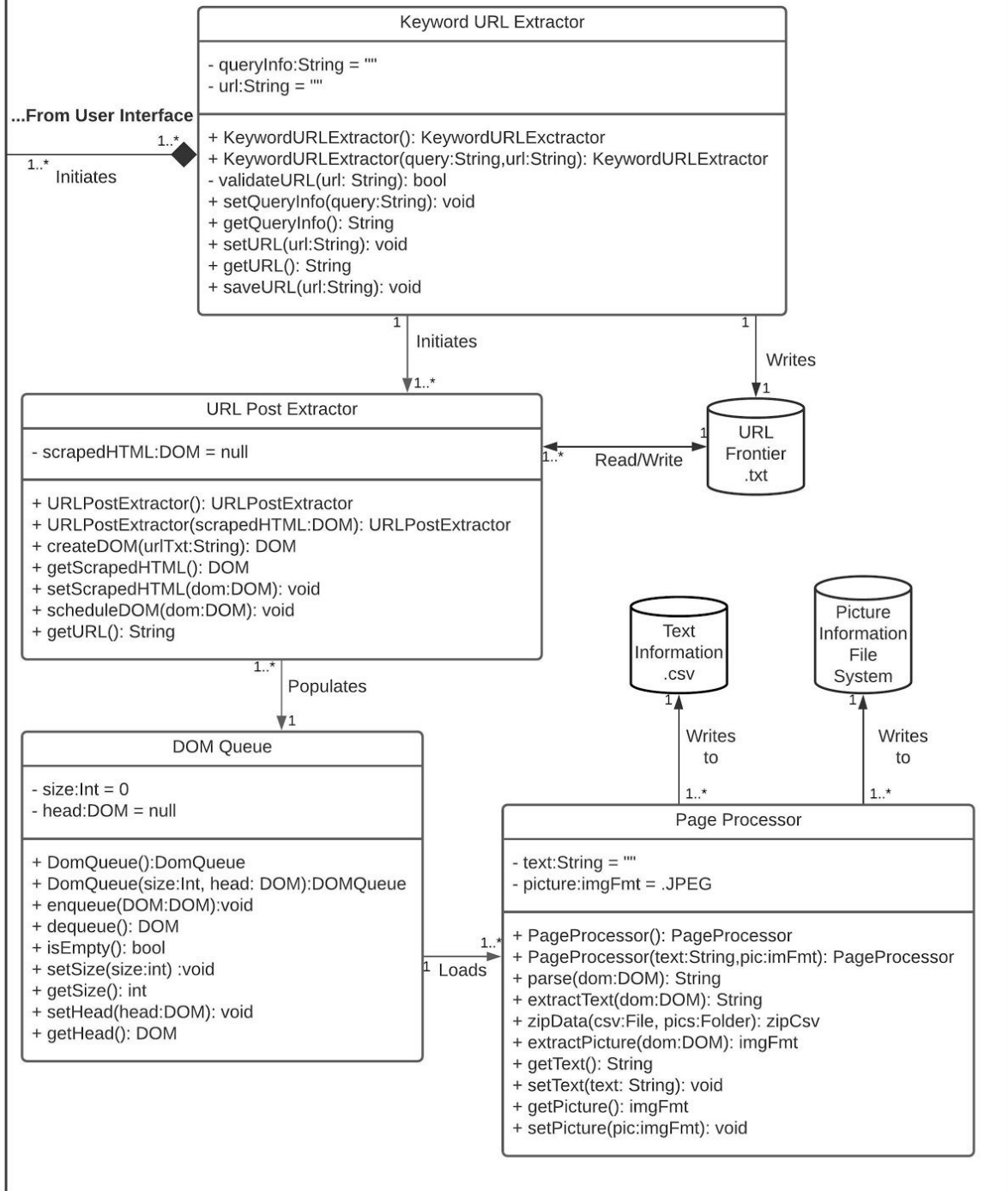


Figure 2.4: S.S. Media Design Class Diagram Part 2

Figures 2.3 and 2.4 show the planned S.S. Media Design Class Diagram (DCD). It has 11 main components which are: the User Interface, the Keyword URL Extractor, the URL Frontier file, the

URL Post Extractor, the DOM Queue, the Page Processor, the Text Information .csv file , the Picture Information File System, the User Information Database, the Search Criteria Object, and the Researcher Object. The following paragraphs each describe a main component into more detail including information such as what their job is and how they will operate. The storage will not be included in this description as they are described above and/or in Section 3.

The User Interface will be the only thing the user will ever interact with. It's main purpose is to manage the user and their information as well as initiate the scraping process. It will allow the user to do tasks such as log in, contact an administrator, select search criteria, save search criteria, start a scrape, etc. The User interface will be able to store information such as usernames, passwords, saved scrape criteria, settings, etc. in a User Information Database. Upon the start of a scrape, the User Interface will send the user's search criteria to the Keyword URL Extractor for processing.

The Keyword URL Extractor will be used to handle the front lines of the S.S. Media's scraping. After receiving a set of search criteria from the User Interface, it's job is to create a URL Frontier, a text file of the URLs of each post to be scraped later. This will be done by creating a URL associated with the specified search criteria and scraping that URL's HTML. The Keyword URL Extractor then parses the HTML for links to posts that need to be scraped and validates those links. After validation, each link gets added to the URL Frontier Database.

The URL Post Extractor's job is to extract post HTML and send it to the DOM Queue (Document Object Model). The URL Post Extractor will iterate through the URL Frontier. For each URL, it will visit the social media post's web page, scrape it's HTML, build that HTML into a DOM Object, send the DOM Object to the DOM Queue, and remove the URL from the URL Frontier.

The DOM Queue's job is to manage DOM Objects while the URL Post Extractor and Page Processor run. As they run, it will receive DOM Object's from the URL Post Extractor and add them to the end of the Queue while concurrently sending DOM Objects at the head of the Queue to the Page Processor.

The Page Processor's job is to parse the DOM Objects. It will go through all information in the DOM Object and scrape the information that is requested. If this includes links to pictures, the Page Processor will download the pictures from the internet. The scraped information will be saved into a Text Information .csv file. The pictures will be saved to a Picture information folder with a unique ID number as the file name linking it to its corresponding text information.

The Search Criteria Object manages a user's specified search criteria. It will hold data values such as Hashtags, Phrases, and Locations. It will also include functions to manipulate those parameters, create Search Criteria Objects, and save search criteria for later use.

The Researcher Object will be used to manage user accounts. It will hold data values such as current search criteria, username, password, and saved search criteria. It will also include functions to manipulate those parameters, create a new Researcher Object, and delete the Researcher Object. The Researcher Object also has a child Administrator Object. The Administrator Object contains all functionality of the Researcher Object, but also includes functions to create an administrator account, approve user accounts, edit account names, ban accounts, and give administrator privileges to user accounts.

### 3. Persistent Data Design

The Persistent Data Design section details how system databases and files used by S.S. Media will be designed. Each database and file description will also include a diagram for further detail.

#### 3.1 Database Descriptions

The database structure consists of one SQL database, containing {x} tables. The first table is the user table, with its purpose being to contain any information necessary in order to validate/invalidate a user when they try to log in. This table has columns for a name, username, password (hashed), type of account (admin, basic), and email address. The string variables will be stored in a VARCHAR format, which dynamically adjusts to the length of the string. The size of a VARCHAR is the length of the data stored, plus two bytes.

**Table 3.1 - User Table Schema**

Name of Field	Data Type	Max Data Size	Description
Primary Key: unique_ID	String: VARCHAR	22 bytes	The primary key of the table. Each user has a specific key randomly generated upon account creation.
first_name	String: VARCHAR	22 bytes	The first name of the user.
last_name	String: VARCHAR	22 bytes	The last name of the user.
hashed_password	String: VARCHAR	52 bytes	The password of the user, chosen by them and encrypted before being stored.
account_type	int	4 bytes	The type of account for the user. Basic (represented by 0) accounts can adjust personalization settings and can run search criteria and gain results. Admin (represented by 1) accounts can do everything that Basic accounts can, as well as assign other users as Admins.
Foreign Key: email_address	String: VARCHAR	52 bytes	The email address of the user, in order to validate the identity of the person if they forget their password or type it wrong too many times. This is also used for the user to log into their account, rather than a username.

The next table is dedicated to saved search criterias. The purpose of this table is to keep a record of all the searches that want to be saved. If the user wants to make a search happen once a day every day for a week, this is where the information gets stored in order to tell the program when to search. This table includes hashtags, locations, platform being scraped, and date range.

**Table 3.2 - User Input Schema**

<b>Name of Field</b>	<b>Data Type</b>	<b>Max Data Size</b>	<b>Description</b>
Primary Key: email_address	String: VARCHAR	52 bytes	The primary key of the table. Each search criteria has a specific key based upon a user's email address
hashtag(s)	String: VARCHAR	252 bytes	This is the list of hashtags in the saved search. They are separated by the hashtag symbol and are in a String format. For example, #nike#boycott#boycottnike
location(s)	String: VARCHAR	252 bytes	The locations of the places in the media being scraped
platform	String: VARCHAR	22 bytes	The specific platform to scrape, either Twitter or Instagram
date_start	timestamp	8 bytes	The start date of the search, as a timestamp
date_end	timestamp	8 bytes	The end date of the search. If no end date is specified, then the search is a one-time search
scrape_frequency	int	4 bytes	The frequency of the scrape between the start and end times.
scrape_interval	time	8 bytes	The amount of time between each scrape.

## 3.2 File Descriptions

The files used by the system include a JSON file, text file, a picture file, and a csv file. The JSON file is used to store the user configuration settings. These settings will be set up as labels with an associated text value. The settings text will be read into the system as Strings that vary in length or as a Boolean value, depending on the setting. The following include the configurable settings with the label and default text associated with the label.

**Table 3.3 - Settings**

<b>Label</b>	<b>Data Type</b>	<b>Default Value</b>	<b>Size</b>
download_location	String	C:\Downloads	256 bytes
view_scrape_history	Boolean	true	1-bit
advanced_search_default	Boolean	true	1-bit
email_notifications	Boolean	true	1-bit
email	String	User's initial sign-up email	256 bytes

The text file being used by the system is a file that will temporarily store URL links in a URL Frontier. This file will be used as an intermediary to pass data from our URL Extractor to our URL Post Extractor.

**Table 3.4 - Text Data Format**

<b>URL</b>	<b>Data Type</b>	<b>Size</b>
twitter.com/example1	String	512 bytes
twitter.com/example2	String	512 bytes

The Picture file will be used to collect and organize all pictures stored in a given scrape. The Picture file will be pushed as a download to the user's configured download location upon completion of a scrape. Each picture will be saved with a unique name corresponding to it's scraped text information in the .csv file.

**Table 3.5 - Picture Data Format**

<b>Picture File</b>	<b>Data Type</b>	<b>Size</b>
pitureID#1919193.png	Varying (.png,.jpg,.jpeg,.gif,etc)	Up to 12mb
pitureID#1903577.png	Varying (.png,.jpg,.jpeg,.gif,etc)	Up to 12mb

The .csv file will be used as a final way to condense, sort, and organize the data collected. This .csv file will be pushed as a download to the user's configured download location upon

completion of a scrape. The scraped data .csv file structure consists of n columns depending on the website.

**Table 3.6 - Text File Structure**

<b>Twitter Headers</b>	<b>Data Type</b>	<b>Size</b>	<b>Desc.</b>	<b>Instagram Headers</b>	<b>Data Type</b>	<b>Size</b>	<b>Desc.</b>
ID	String	128 bytes	The tweet's ID	ID	String	128 bytes	The instagram post's ID
Date	int	4	Date tweet was posted	Date	int	4	Date photo was posted
Time	int	4	Time tweet was posted	Time	int	4	Time photo was posted
Timezone	String	128 bytes	Timezone tweet was posted in	Timezone	String	128 bytes	Timezone photo was posted in
User_ID	String	256 bytes	User's 'handle' that posted the tweet	User_ID	String	256 bytes	User's 'handle' that posted the photo
Username	String	256 bytes	User's display name	Likes	int	16 bytes	Number of likes
Tweet	String	560 bytes	Tweet contents	Replies	String	512 bytes	Content of replies
Replies	String	560 bytes	Replies' content	Location	String	256 bytes	Location the user was at when posting the photo
Retweets	int	16 bytes	Number of times retweeted	Hashtags	String	256 bytes	Hashtags used in the description
Likes	int	16 bytes	Number of times liked	Link	String	512 bytes	Link to the post
Location	String	256 bytes	Location	Image_ID	UUID	128 bytes	An ID for

			the user was at when the tweet was posted				the photo associated with the post
Hashtags	String	560 bytes	Hashtags used in the tweet	Description	String	512 bytes	Contents of description
Link	String	512 bytes	A link to the tweet				

## 4. Requirements Matrix

This section links the functionality described in the SRS document with the System Components and functionality designed in this SDD document. Below is a table that illustrates requirement-component connections. Components are taken directly from the Design Class Diagram (Figure 2.3 and 2.4), so looking at that for reference will help.

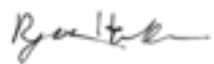


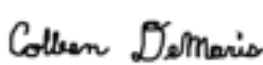
**Table 4.1 - Requirement-Component Connection**

<b>Functional Requirements</b>	<b>System Components -&gt; Function</b>
Use Case #1: Log In	User Interface -> logIn(), Researcher -> getUsername(), Researcher -> setUsername(), Researcher -> getPassword(), Researcher -> setPassword()
Use Case #2: Request Account	User Interface -> contactAdministrator(), User Interface -> validateEmail(), Administrator -> approveAccount(), Administrator -> giveAdmin()
Use Case #3: Select Search Criteria	User Interface -> ScrapeSocialMedia(), Search Criteria -> All accessor/constructors/mutators()
Use Case #4: Save Search Criteria	User Interface -> saveSearchCriteria()
Use Case #5: Contact Administrator	User Interface -> contactAdministrator()
Use Case #6: Scrape Social Media	User Interface -> scrapeSocialMedia(), Keyword URL Extractor -> All functions(), URL Post Extractor -> All functions(), DOM Queue -> All functions(), Page Processor -> All functions()
Use Case # 7: Download Scraped Data	User Interface -> downloadData(), Page Processor -> downloadText(), Page Processor -> downloadPictures()
Use Case #8: Ban Account	Administrator -> banAccount()
Use Case #9: Delete Account	Researcher -> deleteSelfAccount()
Use Case #10: Activate User Account	Administrator -> activateAccount()



## Appendix A – Agreement Between Customer and Contractor

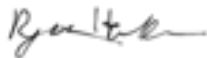
The client (Doctor Rosenbaum) and Blackbear Analytics agree that the system's architecture meets all requirements signed in the SRS document and will be implemented in good faith. All designs will be implemented as close to the specifications in the system architecture as possible, with margins for error being slim or none. In the event that implementation must diverge from the design, the development team will notify the client as soon as possible and work together with the client until a suitable agreement is reached. If the client would like to add designs/requirements for the system's architecture after signing this document, Blackbear Analytics bears no responsibility for the failure to implement new requirements, but will attempt to add them in good faith. Should any new requirements be added, the deliverables schedule in section 2 and 3 of the SRS document will be updated immediately to reflect changes, and the designs/requirements discussed will be reflected in this document by the next deliverable deadline and changes will be finalized with the client.

Name	Signature	Date
<b>Customer:</b>		
<u>Judith E. Rosenbaum</u>		<u>11/09/20</u>
<b>Comments:</b>		
<b>Team:</b>		
<u>Ryan M. Handlon</u>		<u>November 8<sup>th</sup>, 2020</u>
<u>Abdullah I. Karim</u>		<u>November 8<sup>th</sup>, 2020</u>
<u>Griffin L. Fluet</u>		<u>November 8<sup>th</sup>, 2020</u>
<u>Colleen DeMaris</u>		<u>November 8<sup>th</sup>, 2020</u>
<u>James West</u>		<u>November 8<sup>th</sup>, 2020</u>

## Appendix B – Team Review Sign-off

By signing your name below, you acknowledge that you are a member of Blackbear Analytics and have read the document with an in-depth comprehension of the scribed materials. You agree to complete all designs stated on this document as is, in good faith. Should any new designs come up, you agree to assist in re-drafting this document for approval as described in Appendix A. You agree that it is not required to complete designs added after this version is released, but understand that all designs, new or old, must be implemented in good faith.

Ryan M. Handlon



November 8<sup>th</sup>, 2020

Abdullah I. Karim




November 8<sup>th</sup>, 2020

Griffin L. Fluet



November 8<sup>th</sup>, 2020

Colleen DeMaris



November 8<sup>th</sup>, 2020

James West



November 8<sup>th</sup>, 2020

**Comments:**

## **Appendix C – Document Contributions**

Each member contributed to drafting this document evenly (20% each). Ryan Handlon created the Introduction (Section 1), aided in the creation of the System Architecture (Section 2.1), made the description for the Decomposition Description (Section 2.2), and helped in the revision process. Abdullah Karim helped in the creation of the Architectural Design Diagrams (Section 2.1), the development of the Decomposition Diagram (2.2), and the Requirements Matrix (Section 4). Griffin Fluet created the Design Class Diagram (Section 2.2), and helped in the revision process. Colleen DeMaris created the Persistent Data Design (Section 4) and helped in the revision process. James West also created the Persistent Data Design (Section 4) and helped in the revision process. The Appendices were appended and modified from the SRS document and Appendix C have been read and reviewed by the whole team.