*Florida International University*

*School of Computing and Information Sciences*

Software Engineering Focus

Final Deliverable

Project Title: Web Page Archiving and Content Analysis 1.0

**Team Members: Juan Alvarado, Mark Fajet**

**Product Owner(s)**: Andres Cremisini

**Mentor(s)**: Mark Finlayson

**Instructor**: Masoud Sadjadi

The MIT License (MIT)

Copyright (c) *2016 Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

***Abstract***

*This document presents the information necessary to gain a good understanding of the project, Web Page Archiving and Content Analysis 1.0. Before this, no solution existed to batch download large sets of web pages into a format that allows easy programmatic access to pages’ component parts. Now, one can use the tool built during this project to download, from a provided set of URLs, a faithful snapshot of news articles or blog posts, with all multimedia in original formats and with original file names, encapsulate this snapshot in a single file from which images, videos, or other multimedia can be easily extracted, enable the archive to be easily opened in a browser for viewing and browsing, with links to external sites preserved, but with links to local multimedia resources pointing to in the in-archive artifacts, identify the main textual content of the article, and extract that into a separate file with appropriate encodings, identify the posting, publishing, and/or correction date of the article*

**Table of Contents**

**Introduction** ……………………………...………………………………………………………….... 5

Current System ……………………………………………………………………………………... 5

Purpose of New System …………….…………..…………………………………………………... 5

**User Stories**

Implemented User Stories …………………………….……………………………………………..6

Pending User Stories …………………………………...……...………………………………..….. 15

**Project Plan**

Hardware and Software Resources ……………………………..……...……………………….… 16

Sprints Plan ………………………………..………………………………………………………. 17

*Sprint 1*  ....………………………………………………………………………………………... 17

*Sprint 2*  ……………....…………………………………………………………………………... 19

*Sprint 3*  ....………………………………………………………………………………………... 20

*Sprint 4*  ……....…………………………………………………………………………………... 21

*Sprint 5*  ……....…………………………………………………………………………………... 22

*Sprint 6*  ………....………………………………………………………………………………... 23

**System Design**

Architectural Patterns …….…………………………………………………………………….. 24

System and Subsystem Decomposition ………………………………………………………….… 21

Deployment Diagram ………………………………………………………………………….…... 27

Design Patterns …………………………………………………………….………………….….... 27

**System Validation**  …………………………………………………………………………………….28

**Glossary**  ……………………………………………………………………………………………….38

**Appendix**  ……………………………………………………………………………………………….39

Appendix A - UML Diagrams ………………………………………………………………………. 39

Appendix B - User Interface Design ……………………………………………………………….. 65

Appendix C - Sprint Review Reports ………………………………………………………………. 69

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents ………………………………………………………………… 75

**References** …………………………………………………………………….……………...………...81

# Introduction

## 

## Current System

Most web browsers have a tool that allows a user to archive web pages, saving a single html file to disk such that, when clicked, it will open a faithful, offline snapshot of the page. Nevertheless, these tools are inadequate for large-scale research activities involving the downloading, archiving, and analysis of thousands or millions of web pages. In addition, many available tools and plugins have been deprecated. In addition, none of these tools accomplished the goals of saving the snapshots in a format allowing easy programmatic access.

## Purpose of New System

The purpose of this new system is to have a tool that will allow researchers to batch download large sets of pages such that a user can easily download, from a provided set of URLs, a faithful snapshot of news articles or blog posts, with all multimedia in original formats and with original file names, encapsulate this snapshot in a single file from which images, videos, or other multimedia can be easily extracted, enable the archive to be easily opened in a browser for viewing and browsing, with links to external sites preserved, but with links to local multimedia resources pointing to in the in-archive artifacts, identify the main textual content of the article, and extract that into a separate file with appropriate encodings, identify the posting, publishing, and/or correction date of the article.

# 

# User Stories

The following section provides the detailed user stories that were implemented in this iteration of the Web Page Archiving and Content Analysis 1.0 project. These user stories served as the basis for the implementation of the project’s features. This section also shows the user stories that are to be considered for future development.

## Implemented User Stories

#666 As a user I should be able to run the WAT CLI and see the list of available options and arguments

### Description:

* As a user I should be able to run the WAT CLI and see the list of available options and arguments

### Acceptance Criteria:

1. Run program by typing python wat.py
2. See list of available options

#667 Download files from list

### Description:

* As a user I should be able to provide a file with a list of links to download and see a list of .html files that I can click to view an offline version of the site

### Acceptance Criteria:

1. Provide list
2. All links downloaded properly

#668 Downloaded file names

### Description:

* As a user, I expect the file names to accurately reflect the source

### Acceptance Criteria:

1. File names same as original

#669 One click to open

### Description:

* As a user, I should be able to open a downloaded website with one click, without having to navigate through the directory structures

### Acceptance Criteria:

1. File opens with only one click

#670 WAT CLI Cross Platform

### Description:

* As a user, I should expect the WAT CLI to work across various systems and operating systems

### Acceptance Criteria:

1. CLI works on Linux
2. CLI works on Mac
3. CLI works on Windows

#671 Rate limit

### Description:

* As a user I should be able to specify a rate limit option to limit the number of requests per second that we make in order to avoid being throttled.

### Acceptance Criteria:

1. CLI has rate limit option

#672 Research Blocking for foreign sites

### Description: As a user, I expect to be able to download and archive for foreign sites

### Acceptance Criteria:

1. Discover problem causing foreign sites to have index.html display garbage
2. Download should work as expected

#673 Single compressed web archive

### Description:

* As a user, I expect the website to be archived in a single, compressed file which can be clicked and opened in the browser

### Acceptance Criteria:

1. Compressed
2. All contained in one file
3. Opened with one click

#674 Correct HTML file

### Description:

* As a user, I would expect the program to fetch the correct HTML file

### Acceptance Criteria:

1. Correct HTML file recognized 100% of the time

#675 Watviewer Application

### Description:

* As a user, I’d expect an app that knows how to handle these custom .wat files

### Acceptance Criteria:

1. Be able to double click a file with “.wat” extension and have it open in the app with no other configuration

#676 Reduce archive download time

### Description:

* As a user, I expect the files to be downloaded in a reasonable time frame

### Acceptance Criteria:

1. Parallelized downloading to eliminate unnecessary wait time
2. Should have options for parallelized or not

#677 Add browser preference to WatViewer

### Description:

* As a user I should be able to set a preference to open a .wat file in the viewer or my own browser

### Acceptance Criteria:

1. In the WATViewer I should see a dropdown to select my preferred .wat file viewer
2. The selected viewer should be persisted after the WATViewer application is closed

#678 Open .wat on preferred application

### Description:

* As a user I should be able to open a .wat file with a single click on my preferred viewing application

### Acceptance Criteria:

1. Double click on a wat file, should open preferred application
2. Open a wat file through the watapp should open preferred application

#679 See options on WatApp GUI

### Description:

* As a user I should see an option to download, and an option to view .wat files when I open the WatApp

### Acceptance Criteria:

1. GUI for downloading
2. All options that are available to CLI should be available on GUI

#680 Add styles to GUI

### Description:

* As a user, I’d expect to see a visually appealing, easy-to-use GUI

### Acceptance Criteria:

1. The watviewer GUI home page should be restructured in order to look more visually appealing
2. The download page should present the download options in an easy to use and appealing way
3. The download page should show an indication of downloads starting and ending.

#683 Download options for translation

### Description:

* As a user, I’d expect to be able to define what language I would like any foreign sites to be translated to

### Acceptance Criteria:

1. Translation options available
2. Multiple languages available

#684 Clean up tmp

### Description:

* As a user I expect the tmp directory where the WAT content files are unzipped to be cleaned after the application is closed.

### Acceptance Criteria:

1. temp directory should have an organizational scheme
2. temp directory should have no trace of wat files when closed

#685 Show progress of downloads

### Description:

* As a user of the WATViewer I expect to see some sort of progress indicator when downloading files

### Acceptance Criteria:

1. Beginning of download indicated
2. Which downloads finished indicated
3. All done indicated

#686 Extract data from metadata

### Description:

* As a user I expect to be able to extract basic data about an article from the HTML metadata

### Acceptance Criteria:

1. Accept wat filetype
2. Accept html filetype
3. include the following properties
   1. published date
   2. created date
   3. modified date
   4. author
   5. title
   6. header
   7. file location
   8. URL
   9. publisher
   10. Language
   11. publisher origin

#687 Research methods to extract body and dates from html contents

### Description:

* As a user I want to have some options regarding ways of extracting the body content from an article in an HTML file

### Acceptance Criteria:

1. Evaluate feasibility of machine learning based approach
2. Evaluate feasibility of implementation of features found in reference papers
3. Evaluate readability js legacy source code as a starting point
4. Decide on approach to pursuit next sprint

#688 Get publisher origin

### Description:

* As a user, I’d expect the origin of the article to be found

### Acceptance Criteria:

1. Be able to find the country of origin of any article
2. Use an IP address lookup to get publisher origin

#689 WAT file restructuring

### Description:

* As a user, I’d expect the compressed wat files to contain more information about its contents.

### Acceptance Criteria:

1. Must include version number of WAT file
2. Must include what file is the index file
3. Must include extracted info
4. Must include extracted contents
5. Should be extendable to include translations

#690 Housekeeping

### Description:

* As a developer, I’d expect clean, readable, code.

### Acceptance Criteria:

1. Code should be easily readable and understood
2. Unused code should be removed
3. Installing the project should be simple and easy
4. Files and output produced by the code should be limited to the necessary files and output

#691 Date formatting

### Description:

* As a user, I’d expect that extracted dates are correctly parsed and displayed in consistent format

### Acceptance Criteria:

1. All dates parsed correctly
2. All dates shown in same format
3. Indication of ambiguous dates

#692 Complete text extraction implementation

### Description:

* Body/text of article should be extracted from the article accurately and consistently using the decide approach based of Readability and content scoring techniques

### Acceptance Criteria:

1. Given a random article, main body text is extracted
2. Extraction works consistently

#693 Integrate text extraction with CLI

### Description:

* As a user archiving a page with the CLI or GUI I expect the contents of the body to be extracted while downloading and saved in the wat file.

### Acceptance Criteria:

1. Body or article is extracted on download
2. Extracted text added to the extract file in the .wat file

## Pending User Stories

**#**681 Translate website

### Description:

* As a user, I’d expect to have a translated version of any foreign websites I provide

### Acceptance Criteria:

1. Entire website gets translated accurately to desired language

#682 View translated website

### Description:

* As a user, I’d expect to be able to view a website or its translation

### Acceptance Criteria:

1. Option to view original or translated version
2. Translated version displays properly

# Project Plan

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plannings are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

## Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

* wget
* NPM
  + "decompress-zip": "^0.3.0",
  + "electron-store": "^1.3.0",
  + "fs-extra": "^5.0.0",
  + "electron": "~1.7.8",
  + "electron-builder": "^19.56.0"
* Node
* Python 2.7
  + beautifulsoup4==4.6.0
  + dicttoxml==1.7.4
  + future==0.16.0
  + futures==3.2.0
  + geoip2==2.7.0
  + maxminddb==1.3.0
  + PyInstaller==3.3.1
  + python-dateutil==2.7.0
  + urllib3==1.22
  + lxml=4.2.1
  + Juptyer notebook

## 

## Sprints Plan

### Sprint 1

**Sprint Planning Meeting Minutes (1/22/2018):**

Attendees: Andres Cremisini, Juan Alvarado, Mark Fajet

Start time: 3:45

End time: 4:15

After discussion, the velocity of the team were estimated to be: 35.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* 666. As a user I should be able to run the WAT CLI and see the list of available options and arguments
* 667. As a user I should be able to provide a file with a list of links to download and see a list of .html files that I can click to view an offline version of the site
* 668. As a user, I expect the file names to accurately reflect the source
* 669. As a user, I should be able to open a downloaded website with one click, without having to navigate through the directory structures
* 670. As a user, I should expect the WAT CLI to work across various systems and operating systems
* 671. As a user I should be able to specify a rate limit option to limit the number of requests per second that we make in order to avoid being throttled.
* 672. Investigate User Agent or IP Blocking for foreign sites

The team members indicated their willingness to work on the following user stories.

* Juan Alvarado
* 667. As a user I should be able to provide a file with a list of links to download and see a list of .html files that I can click to view an offline version of the site
* 668. As a user, I expect the file names to accurately reflect the source
* 7. Investigate User Agent or IP Blocking for foreign sites
* Mark Fajet
* 666. As a user I should be able to run the WAT CLI and see the list of available options and arguments
* 669. As a user, I should be able to open a downloaded website with one click, without having to navigate through the directory structures
* 670. As a user, I should expect the WAT CLI to work across various systems and operating systems
* 671. As a user I should be able to specify a rate limit option to limit the number of requests per second that we make in order to avoid being throttled.

### 

### Sprint 2

**Sprint Planning Meeting Minutes (2/5/2018):**

Attendees: Andres Cremisini, Juan Alvarado, Mark Fajet

Start time: 3:45

End time: 4:15

After discussion, the velocity of the team were estimated to be: 40.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* 673. As a user, I expect the website to be archived in a single, compressed file which can be clicked and opened in the browser
* 674. As a user, I would expect the program to fetch the correct HTML file
* 675. As a user, I’d expect an app that knows how to handle these custom .wat files

The team members indicated their willingness to work on the following user stories.

* Juan Alvarado
* 675. As a user, I’d expect an app that knows how to handle these custom .wat files
* Mark Fajet
* 674. As a user, I would expect the program to fetch the correct HTML file
* 673. As a user, I expect the website to be archived in a single, compressed file which can be clicked and opened in the brows

### 

### Sprint 3

**Sprint Planning Meeting Minutes (2/19/2018):**

Attendees: Andres Cremisini, Juan Alvarado, Mark Fajet

Start time: 3:45

End time: 4:15

After discussion, the velocity of the team were estimated to be: 40.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* 676. As a user, I expect the files to be downloaded in a reasonable time frame
* 677. As a user I should be able to set a preference to open a .wat file in the viewer or my own browser
* 678. As a user I should be able to open a .wat file with a single click on my preferred viewing application
* 679. As a user I should see an option to download, and an option to view .wat files when I open the WatApp
* 672. Investigate User Agent or IP Blocking/Pooling for foreign sites such as Russia Today

The team members indicated their willingness to work on the following user stories.

* Juan Alvarado
* 677. As a user I should be able to set a preference to open a .wat file in the viewer or my own browser
* 678. As a user I should be able to open a .wat file with a single click on my preferred viewing application
* 672. Investigate User Agent or IP Blocking/Pooling for foreign sites such as Russia Today
* Mark Fajet
* 676. As a user, I expect the files to be downloaded in a reasonable time frame
* 679. As a user I should see an option to download, and an option to view .wat files when I open the WatApp

### Sprint 4

**Sprint Planning Meeting Minutes (3/5/2018):**

Attendees: Andres Cremisini, Juan Alvarado, Mark Fajet

Start time: 3:45

End time: 4:15

After discussion, the velocity of the team were estimated to be: 40.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* #680 Add styles to GUI
* #682 View translated website
* #681 Translate Website

The team members indicated their willingness to work on the following user stories.

* Juan Alvarado
* #680 Add styles to GUI
* #682 View translated website
* Mark Fajet
* #683 Download options for translation
* #681 Translate Website

### Sprint 5

**Sprint Planning Meeting Minutes (3/19/2018):**

Attendees: Andres Cremisini, Juan Alvarado, Mark Fajet

Start time: 3:45

End time: 4:15

After discussion, the velocity of the team were estimated to be: 40.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* #684 Clean up /tmp
* #685 Show progress of downloads
* #686 Extract metadata from metadata
* #687 Research extracting body contents

The team members indicated their willingness to work on the following user stories.

* Juan Alvarado
* #687 Research methods of extracting body contents
* Mark Fajet
* #684 Clean up /tmp
* #685 Show progress of downloads
* #686 Extract metadata from metadata

### 

### Sprint 6

**Sprint Planning Meeting Minutes (4/2/2018):**

Attendees: Andres Cremisini, Juan Alvarado, Mark Fajet

Start time: 3:45

End time: 4:15

After discussion, the velocity of the team were estimated to be: 40.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* #691 date formatting
* #690 house keeping
* #688 get publisher origin
* #689 wat file restructuring
* #692 complete text extraction implementation
* #693 Integrate text extraction with CLI

The team members indicated their willingness to work on the following user stories.

* Juan Alvarado
* #692 complete text extraction implementation
* #693 Integrate text extraction with CLI
* Mark Fajet
* #691 date formatting
* #690 housekeeping
* #688 get publisher origin
* #689 wat file restructuring

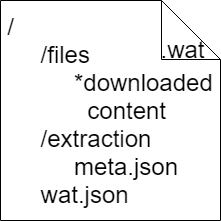
# 

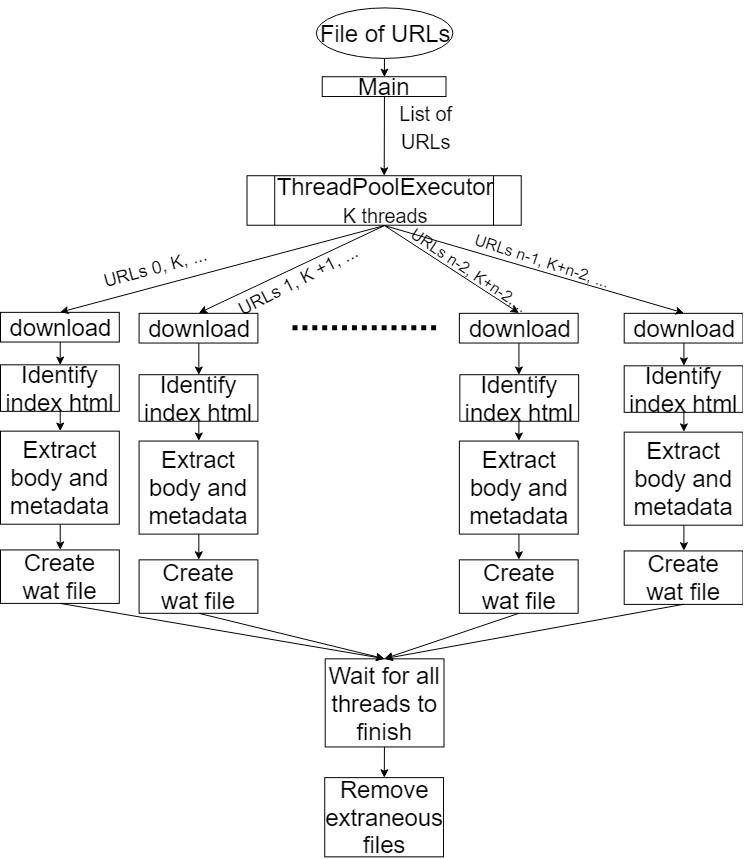
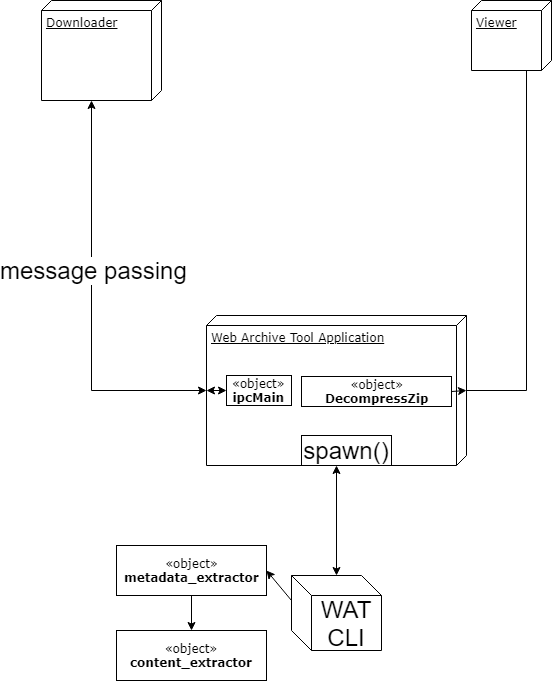
# System Design

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

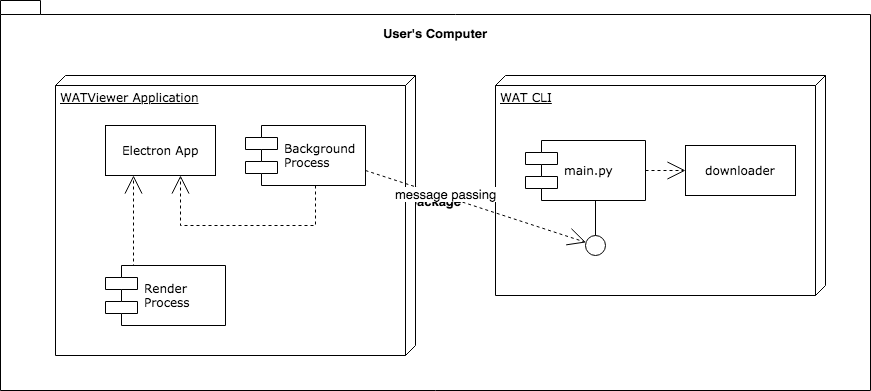
## Architectural Patterns

## System and Subsystem Decomposition





## Deployment Diagram



## Design Patterns

* Master, slave design pattern was used in this project
* Object pool
* Bridge
* Proxy
* Messaging design pattern
* Event-based asynchronous
* Thread-specific storage

# System Validation

**Unit Test**

* Test case ID: 1
* Description/Summary of Test: Extract from HTML
* Pre-condition: html file
* Expected Results: metadata loaded properly
* Actual Result: metadata loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 2
* Description/Summary of Test: Extract title from HTML
* Pre-condition: html file
* Expected Results: title loaded properly
* Actual Result: title loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 3
* Description/Summary of Test: Extract author from HTML
* Pre-condition: html file
* Expected Results: author loaded properly
* Actual Result: author loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 4
* Description/Summary of Test: Extract published\_Date from HTML
* Pre-condition: html file
* Expected Results: published\_Date loaded properly
* Actual Result: published\_Date loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 5
* Description/Summary of Test: Extract modified\_Date from HTML
* Pre-condition: html file
* Expected Results: modified\_Date loaded properly
* Actual Result: modified\_Date loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 6
* Description/Summary of Test: Extract created\_Date from HTML
* Pre-condition: html file
* Expected Results: created\_Date loaded properly
* Actual Result: created\_Date loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 7
* Description/Summary of Test: Extract header from HTML
* Pre-condition: html file
* Expected Results: header loaded properly
* Actual Result: header loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 8
* Description/Summary of Test: Extract publisher from HTML
* Pre-condition: html file
* Expected Results: publisher loaded properly
* Actual Result: publisher loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 9
* Description/Summary of Test: Extract publisher origin from HTML
* Pre-condition: html file
* Expected Results: publisher origin loaded properly
* Actual Result: publisher origin loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 10
* Description/Summary of Test: Extract url from HTML
* Pre-condition: html file
* Expected Results: url loaded properly
* Actual Result: url loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 11
* Description/Summary of Test: Extract language from HTML
* Pre-condition: html file
* Expected Results: language loaded properly
* Actual Result: language loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 15
* Description/Summary of Test: Get inner html for a node
* Pre-condition: beautifulsoup node
* Expected Results: contents of node as string
* Actual Result: contents of node as string
* Status (Fail/Pass): Pass
* Test case ID: 16
* Description/Summary of Test: Get children with content from node
* Pre-condition: beautifulsoup node with children
* Expected Results: children of node with content
* Actual Result: children of node with content
* Status (Fail/Pass): Pass
* Test case ID: 17
* Description/Summary of Test: Get children from node with no content
* Pre-condition: empty beautifulsoup node
* Expected Results: empty list
* Actual Result: empty list
* Status (Fail/Pass): Pass
* Test case ID: 18
* Description/Summary of Test: Get ancestors two levels up
* Pre-condition: beautifulsoup node two levels in depth
* Expected Results: list of two ancestors
* Actual Result: list of two ancestors
* Status (Fail/Pass): Pass
* Test case ID: 19
* Description/Summary of Test: Check if element has single p tag inside
* Pre-condition: beautifulsoup node with single p tag
* Expected Results: True
* Actual Result: True
* Status (Fail/Pass): Pass
* Test case ID: 20
* Description/Summary of Test: Check if element has single p tag inside
* Pre-condition: beautifulsoup node with single p inside extra element
* Expected Results: False
* Actual Result: False
* Status (Fail/Pass): Pass
* Test case ID: 21
* Description/Summary of Test: Check if element has single p tag inside
* Pre-condition: beautifulsoup node with two p tags inside
* Expected Results: False
* Actual Result: False
* Status (Fail/Pass): Pass
* Test case ID: 22
* Description/Summary of Test: Check if element has child block elements
* Pre-condition: beautifulsoup node with an block element tag
* Expected Results: True
* Actual Result: True
* Status (Fail/Pass): Pass
* Test case ID: 23
* Description/Summary of Test: Check if element has child block elements
* Pre-condition: beautifulsoup node with no block element tags
* Expected Results: False
* Actual Result: False
* Status (Fail/Pass): Pass
* Test case ID: 24
* Description/Summary of Test: Check if node contains element that is candidate for empty check
* Pre-condition: beautifulsoup node that is empty eligible tag
* Expected Results: True
* Actual Result: True
* Status (Fail/Pass): Pass
* Test case ID: 25
* Description/Summary of Test: Check if node contains element that is candidate for empty check
* Pre-condition: beautifulsoup node that is not empty eligible tag
* Expected Results: False
* Actual Result: False
* Status (Fail/Pass): Pass
* Test case ID: 26
* Description/Summary of Test: Check if node has no content or non content tags
* Pre-condition: beautifulsoup node that is empty and/or contains br and hr tags
* Expected Results: True
* Actual Result: True
* Status (Fail/Pass): Pass
* Test case ID: 27
* Description/Summary of Test: Remove given node and return sibling next node in a dfs fashion
* Pre-condition: root beautifulsoup node that has a sibling node
* Expected Results: removed root node and returned sibling node
* Actual Result: removed root node and returned sibling node
* Status (Fail/Pass): Pass
* Test case ID: 28
* Description/Summary of Test: return a node’s sibling next node in a dfs fashion
* Pre-condition: root beautifulsoup node that has a child node
* Expected Results: child node
* Actual Result: child node
* Status (Fail/Pass): Pass
* Test case ID: 29
* Description/Summary of Test: return a node’s sibling next node in a dfs fashion
* Pre-condition: root beautifulsoup node that’s a leaf with ancestor sibling
* Expected Results: ancestor sibling
* Actual Result: ancestor sibling
* Status (Fail/Pass): Pass
* Test case ID: 30
* Description/Summary of Test: return a node’s sibling next node in non dfs fashion
* Pre-condition: root beautifulsoup node with sibling
* Expected Results: sibling node
* Actual Result: sibling node
* Status (Fail/Pass): Pass
* Test case ID: 31
* Description/Summary of Test: return class name of node
* Pre-condition: beautifulsoup node with or without classes
* Expected Results: node’s classes as string
* Actual Result: node’s classes as string
* Status (Fail/Pass): Pass
* Test case ID: 32
* Description/Summary of Test: return id of node
* Pre-condition: beautifulsoup node with or without id
* Expected Results: node’s id as string
* Actual Result: node’s id as string
* Status (Fail/Pass): Pass
* Test case ID: 33
* Description/Summary of Test: return role attribute of node
* Pre-condition: beautifulsoup node with or without role attribute
* Expected Results: node’s role attribute as string
* Actual Result: node’s role attribute as string
* Status (Fail/Pass): Pass
* Test case ID: 34
* Description/Summary of Test: check if elem is tag
* Pre-condition: beautifulsoup tag
* Expected Results: True
* Actual Result: True
* Status (Fail/Pass): Pass
* Test case ID: 35
* Description/Summary of Test: check if elem is tag
* Pre-condition: beautifulsoup non tag element
* Expected Results: False
* Actual Result: False
* Status (Fail/Pass): Pass
* Test case ID: 36
* Description/Summary of Test: get element’s first child
* Pre-condition: beautifulsoup node with children
* Expected Results: node’s first child
* Actual Result: node’s first child
* Status (Fail/Pass): Pass
* Test case ID: 37
* Description/Summary of Test: get element’s next sibling
* Pre-condition: beautifulsoup node with sibling
* Expected Results: node’s sibling
* Actual Result: node’s sibling
* Status (Fail/Pass): Pass

**Integration Test**

* Test case ID: 12
* Description/Summary of Test: Extract from WAT file
* Pre-condition: wat file
* Expected Results: metadata loaded properly
* Actual Result: metadata loaded properly
* Status (Fail/Pass): Pass
* Test case ID: 13
* Description/Summary of Test: Download list of web pages
* Pre-condition: valid txt file containing URLs separated by new line
* Expected Results: Download finished with code 0
* Actual Result: Download finished with code 0
* Status (Fail/Pass): Pass
* Test case ID: 14
* Description/Summary of Test: Download invalid list of web pages
* Pre-condition: invalid txt file not containing URLs separated by new line
* Expected Results: Download finished with non-zero exit code
* Actual Result: Download finished with non-zero exit code
* Status (Fail/Pass): Pass
* Test case ID: 38
* Description/Summary of Test: Parse a simple document and get it’s contents
* Pre-condition: simple constructed html file
* Expected Results: constructed contents
* Actual Result: constructed contents
* Status (Fail/Pass): Pass
* Test case ID: 38
* Description/Summary of Test: Parse a simple document and get it’s contents
* Pre-condition: invalid file
* Expected Results: Null
* Actual Result: Null
* Status (Fail/Pass): Pass
* Test case ID: 39
* Description/Summary of Test: Parse article should extract content
* Pre-condition: Valid article’s html file
* Expected Results: Article’s contents
* Actual Result: Article’s contents
* Status (Fail/Pass): Pass

# 

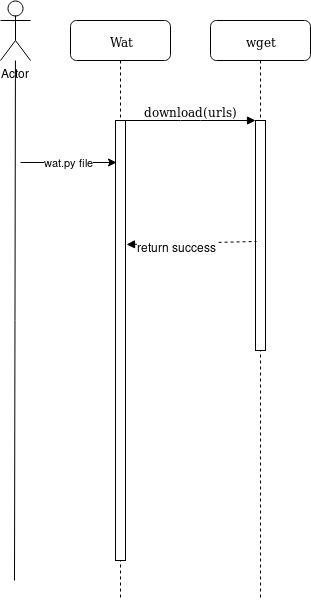
# Glossary

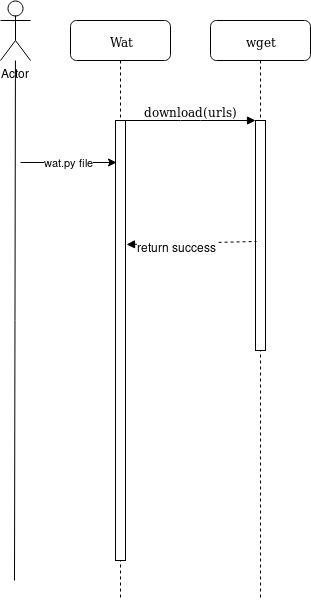
* WAT: Web Archive Tool
* WAT file: Custom file format created during the project that hold all files of a website and various meta information about the files
* WATviewer: GUI for downloading and viewing WAT files
* Metadata of article: descriptive information like author, publisher, published date, modified date, header, title, url, publisher origin, language
* Content of article: Referring to main body of article

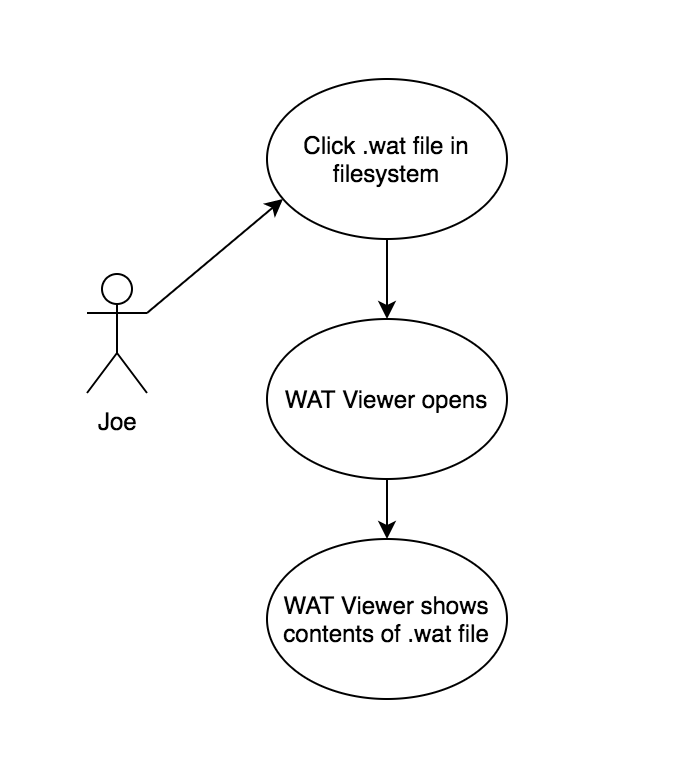
# Appendix

## Appendix A - UML Diagrams

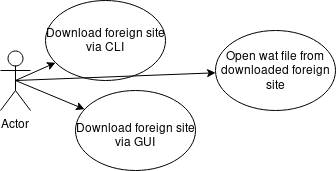
## #666

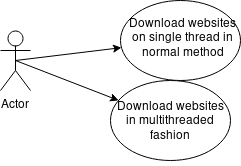
#667

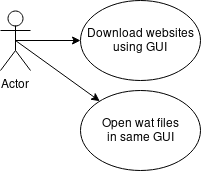
#668

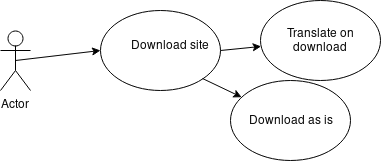
#669

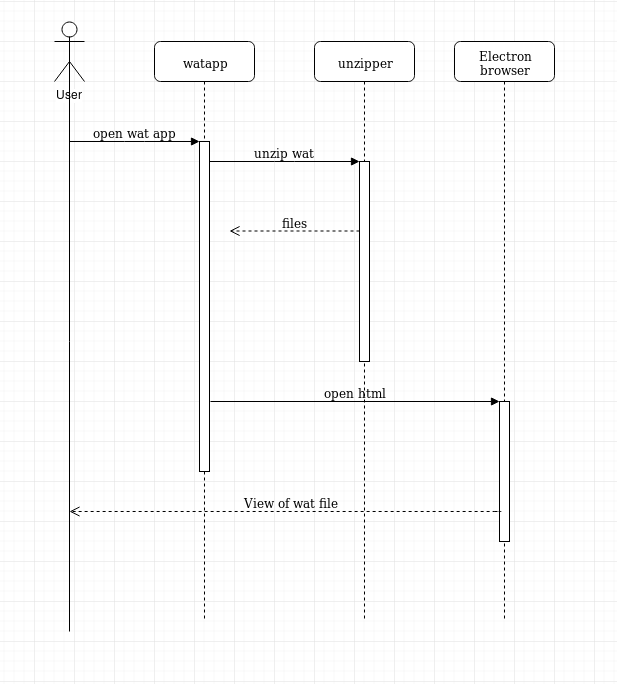
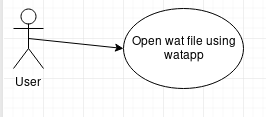
## #675

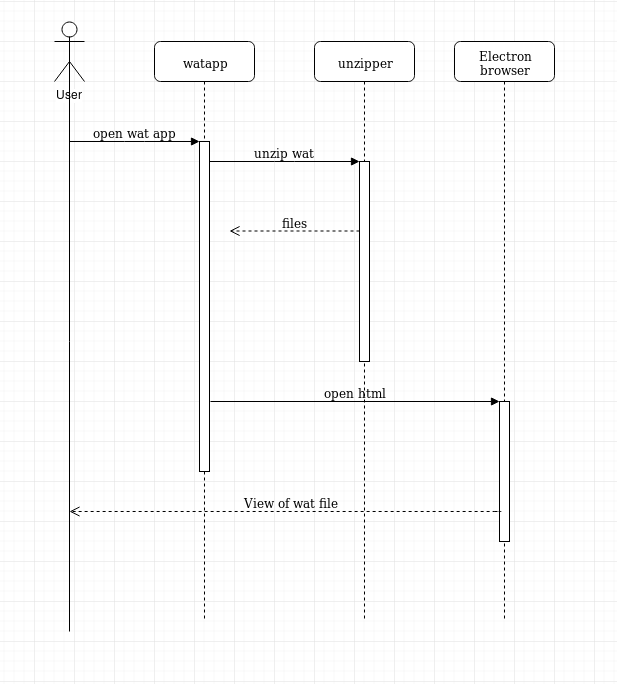
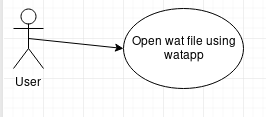
#672

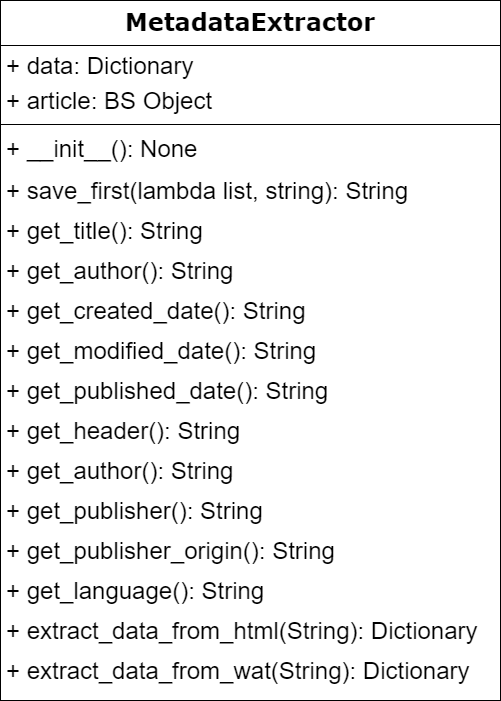
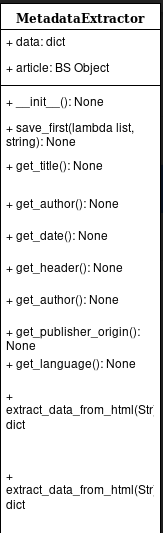
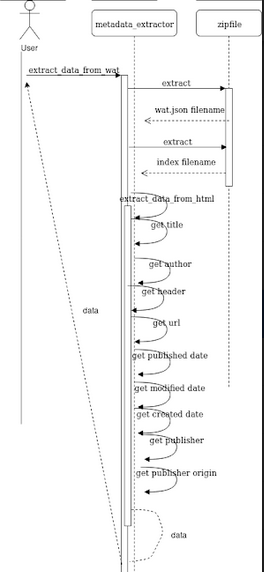
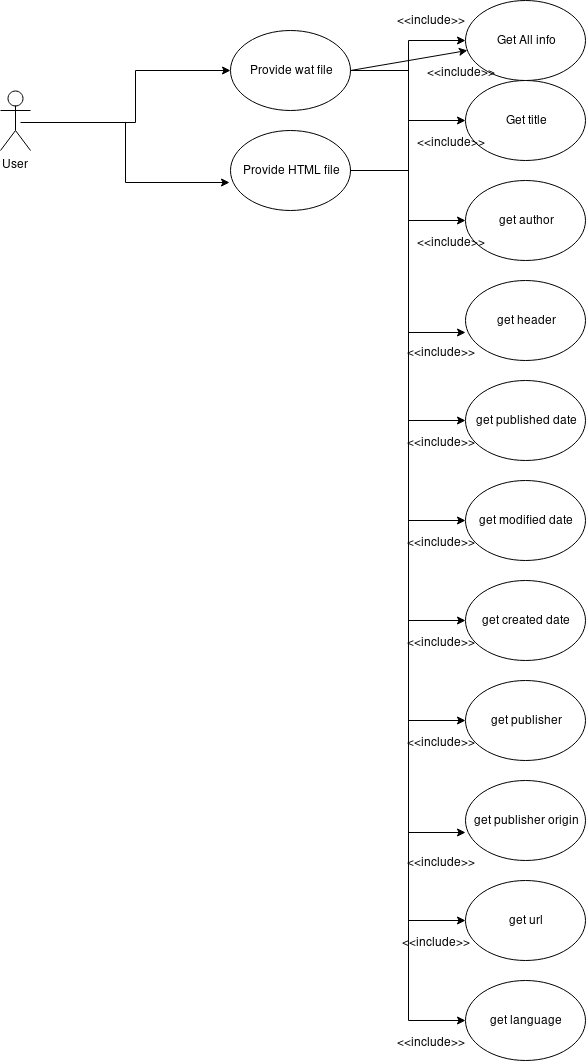
#676

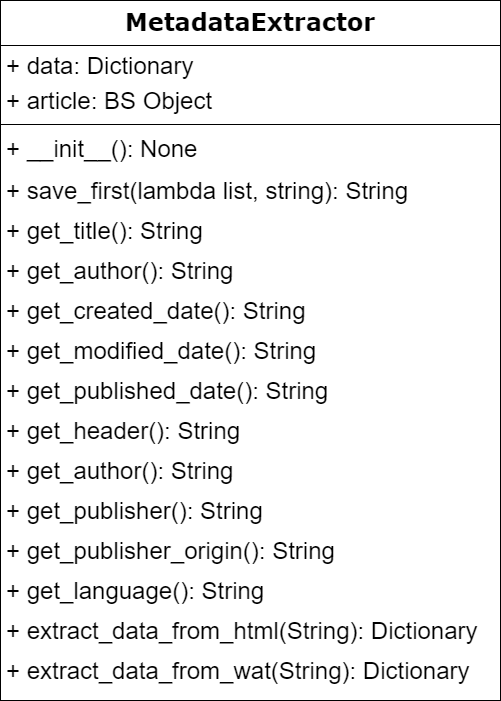
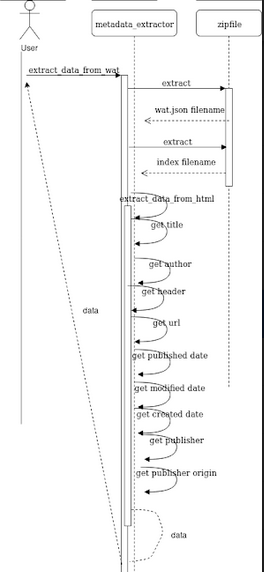
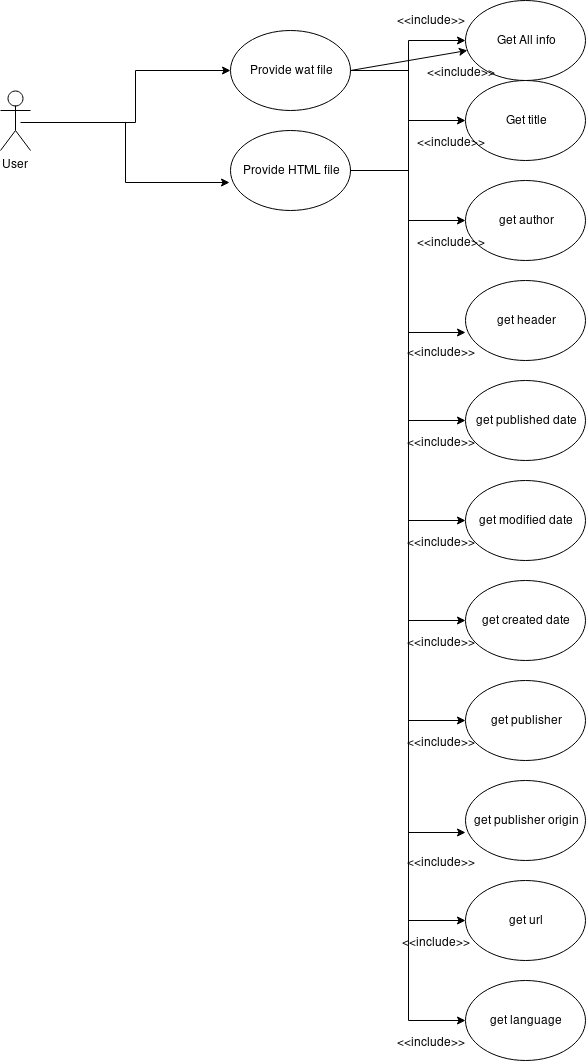
#679

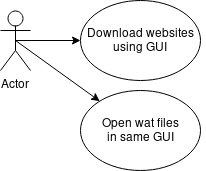
#683

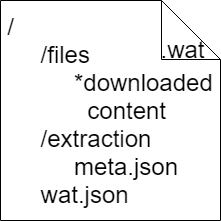
#684

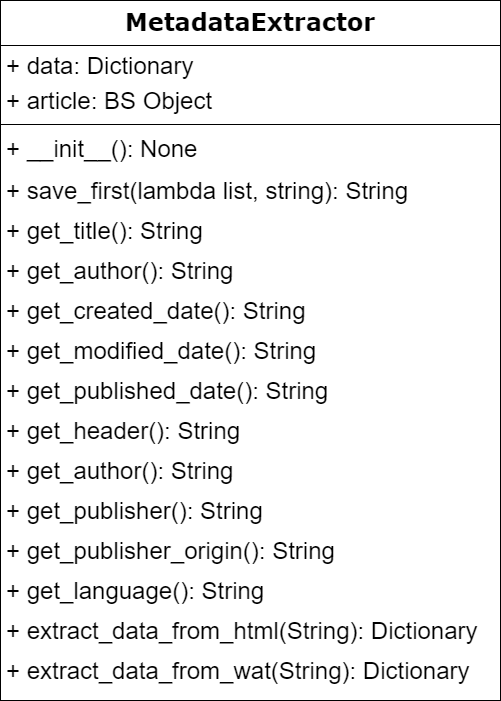
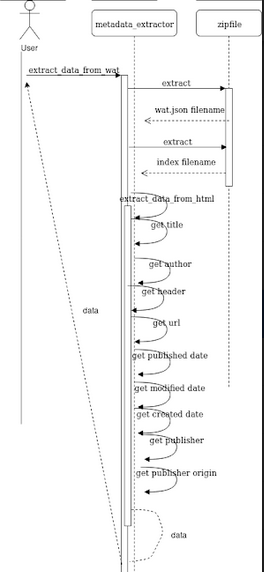
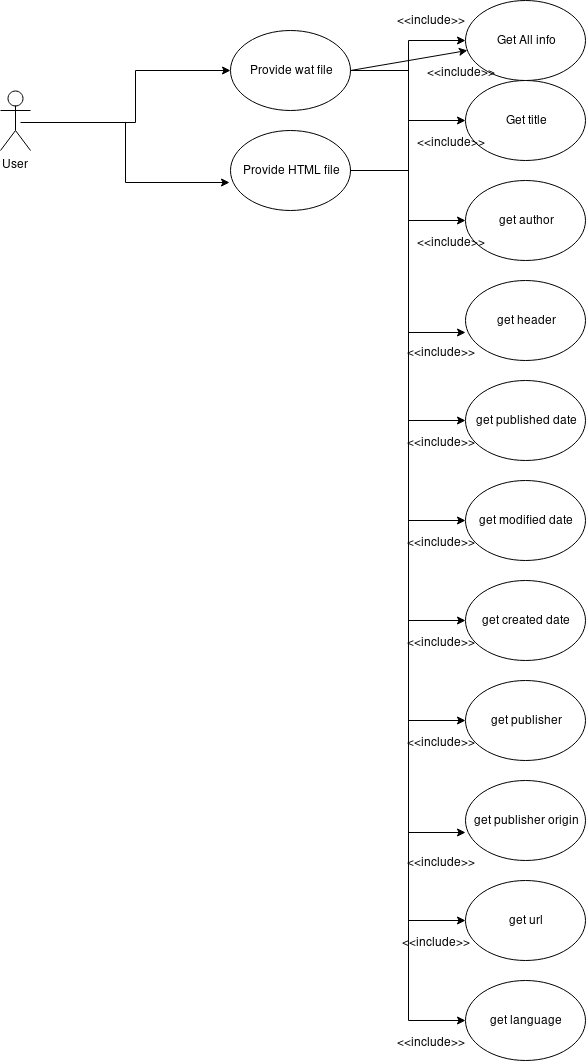
#685 

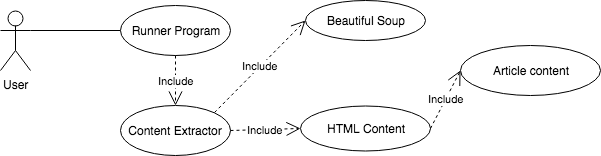
#686

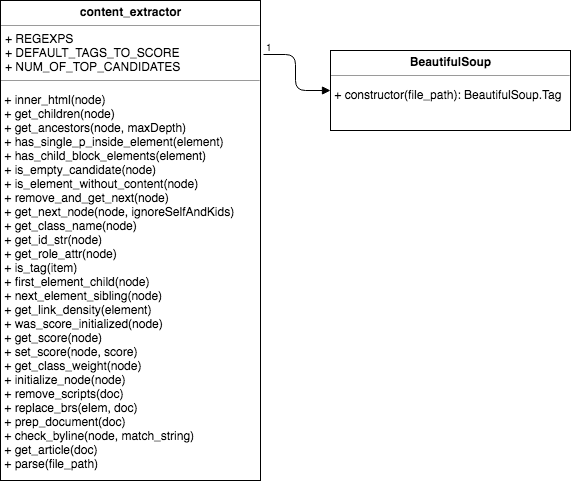
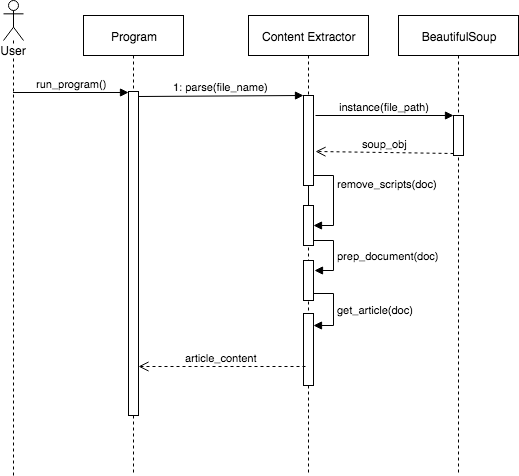
#688 

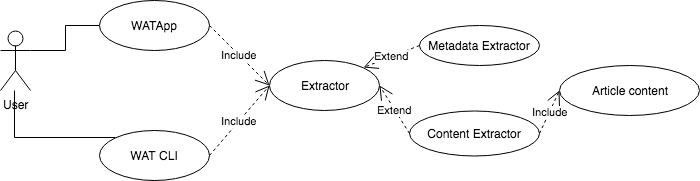
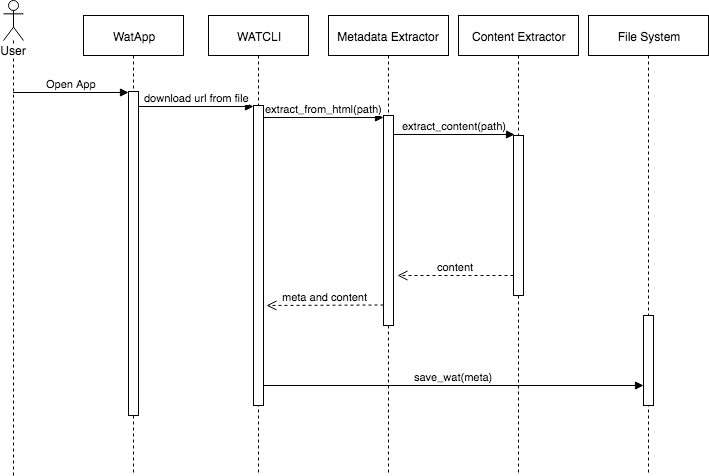
#689

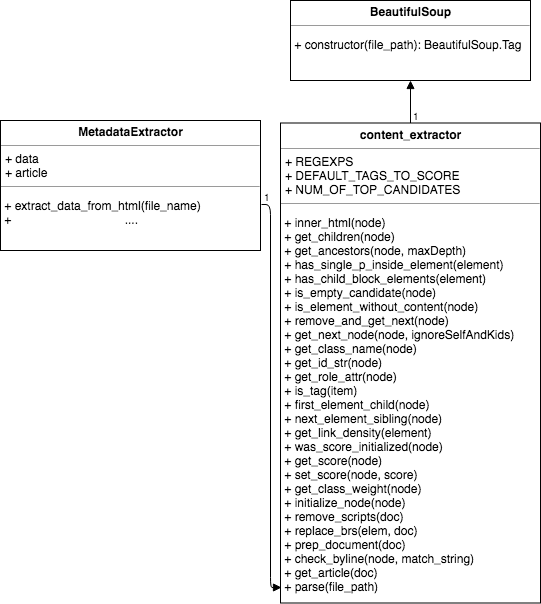
#690

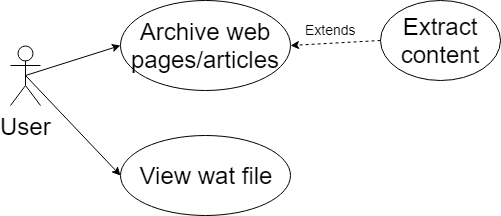
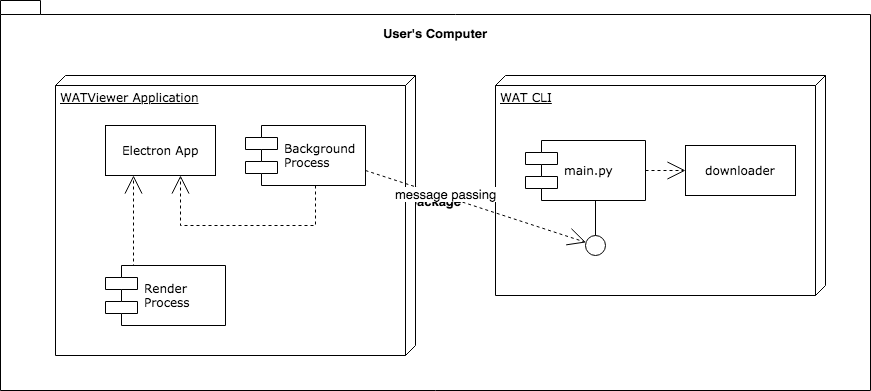
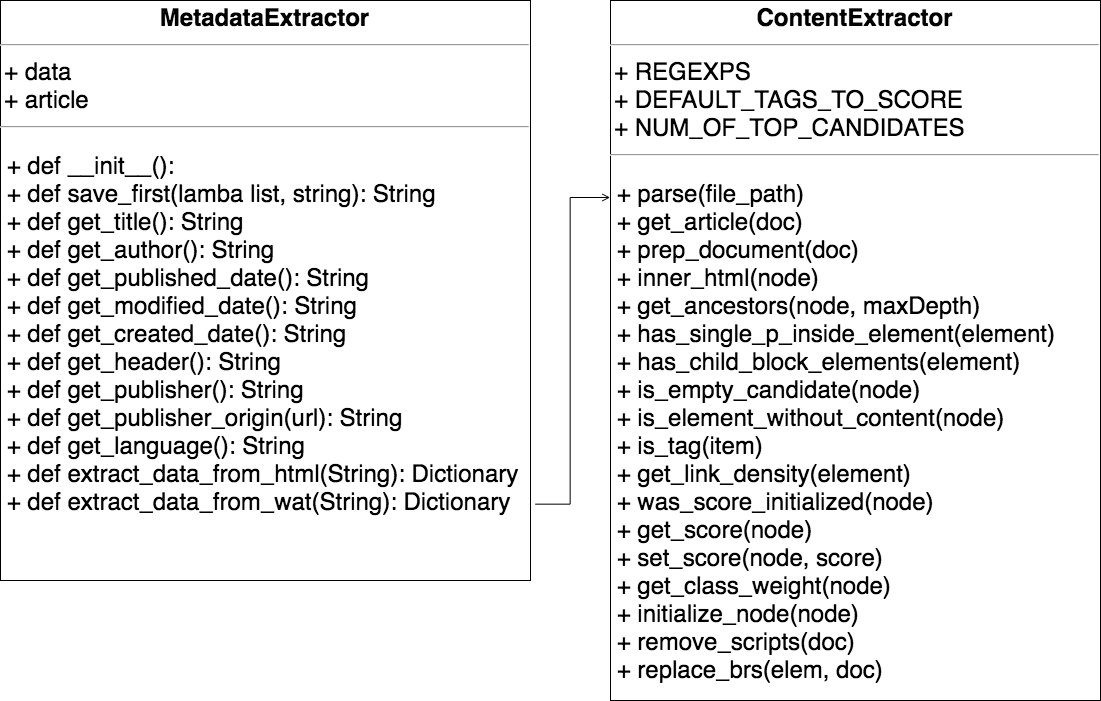
#691

#692****



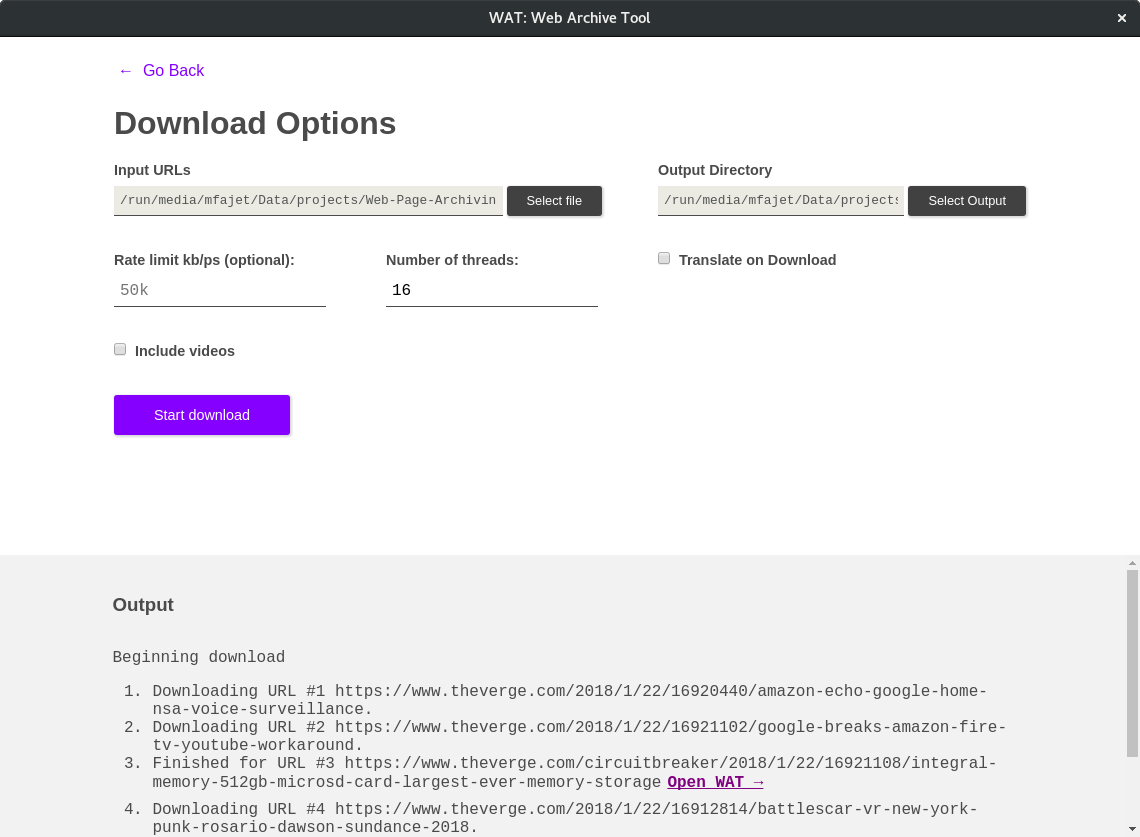
#693****

****



## Appendix B - User Interface Design





## 

## 

## 

## 

## 

## Appendix C - Sprint Review Reports

**Sprint 1 Review Meeting Minutes**

Attendees: Mark Fajet, Juan Alvarado, Andres Cremisini, Mark Finlayson, Pedro Torres

Start time: 11:15

End time: 11:30

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* 1. As a user I should be able to run the WAT CLI and see the list of available options and arguments
* 2. As a user I should be able to provide a file with a list of links to download and see a list of .html files that I can click to view an offline version of the site
* 3. As a user, I expect the file names to accurately reflect the source
* 4. As a user, I should be able to open a downloaded website with one click, without having to navigate through the directory structures
* 5. As a user, I should expect the WAT CLI to work across various systems and operating systems
* 6. As a user I should be able to specify a rate limit option to limit the number of requests per second that we make in order to avoid being throttled.

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* None
* How this should be reflected on the user story definition in Mingle:
  + N/A

**Sprint 2 Review Meeting Minutes (2/16/18)**

Attendees: Mark Fajet, Juan Alvarado, Andres Cremisini, Mark Finlayson, Pedro Torres

Start time: 11:15

End time: 11:30

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* 8. As a user, I would expect the program to fetch the correct HTML file
* 10. As a user, I expect the website to be archived in a single, compressed file which can be clicked and opened in the browser
* 11. As a user, I’d expect an app that knows how to handle these custom files

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* None
* How this should be reflected on the user story definition in Mingle:
  + N/A

**Sprint 3 Review Meeting Minutes (3/2/18)**

Attendees: Mark Fajet, Juan Alvarado, Andres Cremisini, Mark Finlayson, Pedro Torres

Start time: 11:15

End time: 11:30

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* 677. Add browser preference to WatViewer
* 678. Open .wat on preferred application
* 679. See options on WatApp GUI
* 676. Reduce archive download time
* 672. Research Blocking for foreign sites

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* None
* How this should be reflected on the user story definition in Mingle:
  + N/A

**Sprint 4 Review Meeting Minutes (3/9/18)**

Attendees: Mark Fajet, Juan Alvarado, Andres Cremisini, Mark Finlayson, Pedro Torres

Start time: 11:15

End time: 11:30

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

* 683. Download options for translation
* 680. Add styles to GUI

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* 681. Translate website
* 682. View translated website
* How this should be reflected on the user story definition in Mingle:
  + These did not get started as a result of a shorter sprint. Moved to product backlog

**Sprint 5 Review Meeting Minutes (3/30/18)**

Attendees: Mark Fajet, Juan Alvarado, Andres Cremisini, Mark Finlayson, Pedro Torres

Start time: 11:15

End time: 11:30

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

* 686. Extract data from metadata
* 685. Show progress of downloads
* 684. Clean up tmp
* 687. Research methods to extract body and dates from html

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* None

**Sprint 6 Review Meeting Minutes (4/13//18)**

Attendees: Mark Fajet, Juan Alvarado, Andres Cremisini, Mark Finlayson, Pedro Torres

Start time: 11:15

End time: 11:30

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

* #691 date formatting
* #690 house keeping
* #688 get publisher origin
* #689 wat file restructuring
* #692 complete text extraction implementation
* #693 Integrate text extraction with CLI

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* None

## 

## Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

**User Manuals:**

In Reference Work:

* python\_files/sputnik.py generated everything in /example\_output. the code currently dumps text files in the archive folder, but this should happen in a separate folder with corresponding file names, as you’ll see in /example\_output.
* python\_files/WikiRefGetter.py generated everything in to\_parse. The first step of this project will probably be extracting all of the text from the files named index.html. Note that this code does not name the files index.html, as this was done manually. Your downloader code should do this automatically.

Watapp can be run by using “npm run start” if you switch the variable noPython to false or by running “npm run dist” and running the application that creates if noPython is true (default).

Pressing “Alt+Left” will go back a page

CLI usage: wat.py [-h] -f FILE [-d OUTDIR] [--rate\_limit RATE\_LIMIT] [--videos]

[-m THREADS]

optional arguments:

-h, --help show this help message and exit

-f FILE, --file FILE File containing list of URLs

-d OUTDIR, --outdir OUTDIR

Output directory

--rate\_limit RATE\_LIMIT

Value to limit requests. Ex 50k for 50kb/s

--videos Include videos and audio

-m THREADS, --threads THREADS

Download sites using multiple threads

When downloading a list of urls from a file with either the CLI or the GUI, file should be a txt file with each url on its own line, no commas or other delimiters.

The app is simple to use. Simply, click the icon then select the action you want to do. If you want to download URLs then you need to provide a file as described above, an output directory and the number of threads. Other options can be ignored or you can specify those as well. If you want to view a wat file, you click the button and a file dialog will appear for you to select one. Depending on the user preferences which can be set at the bottom of the home screen, it will either open in the browser or in the watapp. At any point, you can use “Alt+Left” on your keyboard to move back a screen. When viewing a wat file in the watapp, you can use “Alt+U” in order to view the extracted content on another page.

**Running tests:**

* Go into the Analysis folder and run “python test\_extractor.py” to test metadata extraction
* Go into the Analysis folder and run “python test\_content\_extractor.py” to test content extraction
* Go into the Analysis folder and run “python extract\_metadata\_wat.py” to run metadata and content extraction on the documents in Reference Work

**Installation instructions:**

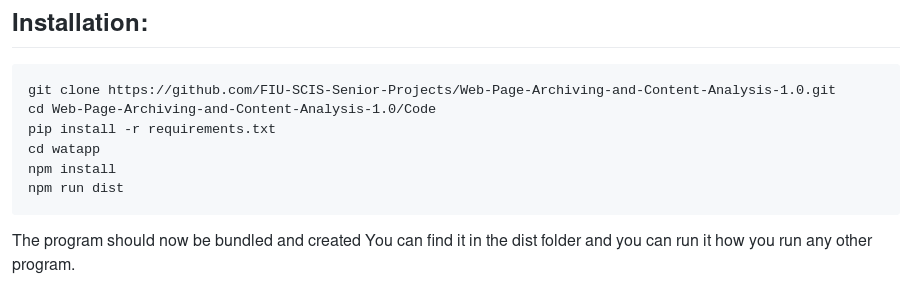
City database needs to be downloaded from here and placed in "Code/Analysis/geo\_db"

wget must be installed

npm must be installed

node must be installed (latest version. If you have too early a version you won't be able to build)

Then do the following:

****

On Windows and Mac, it may be necessary to change the noPython variable to False and run the application with “npm run start”

**Maintenance:**

The CLI folder includes all of the code for the downloading CLI. In addition, it includes various test files with URLs to use on the downloader. Any changes will likely be in wat.py or download.py. The download.py has imports that may seem extraneous as they are not used in that file like bs4; however, these imports being there are necessary for pyinstaller to find them properly since they are dependencies of the meta\_data extractor.

The watapp folder contains all of the code for the GUI. Changes will likely most take place in main.js, download.js, download.html, and index.html. The GUI downloader is dependent on the CLI downloader. When the application is built, it also builds an executable form of the CLI. The purpose of this was to remove python (and all the python dependencies) as a dependency of the application. This makes the application a lot more shareable which was an important aspect for us. This is done with something called pyinstaller. The actual command run during build can be found in the package.json file. The flags used are important and should not be removed without careful consideration and analysis.

The Analysis folder contains all of the python files and tests used for both the metadata extraction of an html or wat file and the content extraction of an html or wat file. The CLI is dependent on these files because it imports the metadata extractor. Changes to these files will likely cascade throughout all the way to the GUI.

There is an important difference between running the application using the application built by “npm run dist” and “npm run start”. Mainly, the differences the developer will notice will only appear in the file locations used in the app. Under “npm run start,” one can rely on the idea that “./” refers to the current directory, but with the built application that won’t necessarily be true as it may refer to wherever the program is installed on your machine.

The Research folder contains various files that came about through research on solutions to various problems solved throughout the semester. Most of the successful aspects were taken out of there and incorporated into the various parts of the application. However, the files in here still have merit. The extractor\_nbs folder has an ipython notebook that was used for developing the content extractor. The main\_html\_finder directory contains hmtl templates and a simple express server that hosts the files on localhost. These files were used to create html files that heavily referred to each other with iframes in order to test wget’s download of multiple html files. This will still be useful if the future work decides to incorporate a wget-free main html finding. The html\_root\_finder.py file in CLI has this work partially complete already with the function find\_root\_html, however it is not perfect. It uses a graph theory algorithm we created in order to find the root html file based off of the assumption that the root html file will be the only one that has no incoming references to it. This is mainly true and only became false when a wget download would rename a file that was referenced. The various other files in Research are for creating wat fie as a recognized file type by linux.

In the Reference Work section, there are two python files created by Andres Cremisini, the project owner. These were files specifically for downloading content from two websites. In addition, there are tons of html and css files downloaded and stored here and not pushed to github because it is over 5GB of data and pushing that is bad practice. It is these files that the test “extract\_metadata.py” is supposed to work on.

**Wishlist**:

* Translation on download
* View translated website
* Fix Mac build resources path
  + See line 272 of main.js
* View extracted data when viewing web page in application
  + Currently only done with “Alt+U” key press
* Create an API/Python module that allows easy access to the contents of a wat file
* Improve content extraction
* Improve backwards functionality
  + Store state of download so it remains when returning
* Add a menu to the GUI so that the keyboard shortcuts aren’t necessary
* Improved message passing between electron app and download script.
* Register the .wat file with os so that it recognizes the file type
  + A basic linux version of this is in Research folder
* Get icons to show up under built version of application on platforms other than Linux
* Create a “distWindows” script in package.json similar to current dist, but the unix file delimiter “:” should be replaced with “;”
* Create wat file from an already downloaded directory of html files and css files.
  + See maintenance section about Research for some more details about this
* A speedup in download time could occur if the content extraction can begin once the html file is downloaded instead of waiting for all files to be downloaded. Then extraction can be run in parallel with the extraction.

# 

# References

Papers:

1. Gupta, S., Kaiser, G.E., Grimm, P. et al, “Automating Content Extraction of HTML Documents”, World Wide Web (2005) 8: 179. <https://doi.org/10.1007/s11280-004-4873-3>
2. Riboni, Daniele, “Feature Selection for Web Page Classification”, 2002, <https://pdfs.semanticscholar.org/1614/3fb51e3278148b1a611cc5b226b1448b4750.pdf>
3. Sandeep Sirsat. "Extraction of Core Contents from Web Pages", International Journal of Engineering Trends and Technology(IJETT), V8(9),484-489 February 2014.
4. Pasternack, Jeff, and Dan Roth. “Extracting Article Text from the Web with Maximum Subsequence Segmentation.” Proceedings of the 18th International Conference on World Wide Web - WWW 09, 2009, doi:10.1145/1526709.1526840.
5. Gondse, Pranjali G, and Anjali B Raut. “Main Content Extraction From Web Page Using Dom.” International Journal of Advanced Research in Computer and Communication Engineering, vol. 3, no. 3, 3AD.

Reference content extractor implementation for the browser:

<https://code.google.com/archive/p/arc90labs-readability/>

Documentation of dependencies:

GEO\_IP database: <http://geoip2.readthedocs.io/en/latest/>

Futures (thread pool executor) <https://pythonhosted.org/futures/#threadpoolexecutor-objects>

Electron: <https://electronjs.org/>

Electron builder: <https://www.electron.build/>

Beautiful soup documentation: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Node: <https://nodejs.org/en/>

DecompressZip documentation: <https://www.npmjs.com/package/decompress-zip>

Dateutil.parse documentation: <http://dateutil.readthedocs.io/en/stable/parser.html>

Wget manual: <https://www.gnu.org/software/wget/manual/wget.html>

Jupyter notebook documentation: <http://jupyter.org/documentation>

Fs-extra documentation: <https://www.npmjs.com/package/fs-extra>

Dicttoxml documentation: <https://pypi.python.org/pypi/dicttoxml>

Pyinstaller documentation: <https://pyinstaller.readthedocs.io/en/v3.3.1/>

Lxml documentation: <http://lxml.de/>

Electron store code and documentation: <https://github.com/sindresorhus/electron-store>

Electron local shortcut documentation: <https://www.npmjs.com/package/electron-localshortcut>

Argparse documentation: <https://docs.python.org/2/library/argparse.html>

Python os documentation: <https://docs.python.org/2/library/os.html>

Python sys documentation: <https://docs.python.org/2/library/sys.html>

Python shutil documentation: <https://docs.python.org/2/library/shutil.html>

Python time documentation: <https://docs.python.org/2/library/time.html>

Python re documentation: <https://docs.python.org/2/library/re.html>

Python platform documentation: <https://docs.python.org/2/library/platform.html>

Python subprocess documentation: <https://docs.python.org/2/library/subprocess.html>

Python zipfile documentation: <https://docs.python.org/2/library/zipfile.html>

Python json documentation: <https://docs.python.org/2/library/json.html>

Python socket documentation: <https://docs.python.org/2/library/socket.html>

Python copy documentation: <https://docs.python.org/2.7/library/copy.html>

Python tempfile documentation: <https://docs.python.org/2/library/tempfile.html>

Python csv documentation: <https://docs.python.org/2/library/csv.html>

Python xml.dom.mindom.parsestring documentation: <https://docs.python.org/2/library/xml.dom.minidom.html#xml.dom.minidom.parseString>

Python codecs documentation: <https://docs.python.org/2/library/codecs.html>

Python unittest documentation: <https://docs.python.org/2/library/unittest.html>