EEE521 Final Year Project Report

School of Computing, Engineering & Intelligent Systems

Evan Gallagher B00642761

BSc Hons Computer Science

Scraping Websites for Law Enforcement

Supervisor Professor Kevin Curran
Second Marker Dr Daniel Kelly

03/05/18

Plagiarism Statement

I declare that this is all my own work and does not contain unreferenced material copied from any other source. I have read the University's policy on plagiarism and understand the definition of plagiarism. If it is shown that material has been plagiarised, or I have otherwise attempted to obtain an unfair advantage for myself or others, I understand that I may face sanctions in accordance with the policies and procedures of the University. A mark of zero may be awarded and the reason for that mark will be recorded on my file.

I confirm that the Originality Score provided by TurnItIn for this report is ________.

Evan Gallagher

Acknowledgements

I would like to thank my project supervisor, Professor Kevin Curran for the help and support he has given me over the course of this project.

I would also like to give thanks to my family and friends for supporting me during this year.

Contents

Table of Figures

List of Tables

Abstract

As the Internet and World Wide Web continue to expand and amass more users, increased rates of crime occur. One such crime is modern slavery, or human trafficking. Social media and web forums are often employed by traffickers to recruit and advertise victims anonymously.

While this issue continues to propagate through both the surface web and the dark web, web scraping tools must be developed to extract and analyse the information on these websites to identify traffickers and victims of human trafficking.

This project aims to investigate how crime continues to occur on the Internet and where on the web this is. Solutions to the problem using current web scraping technology are researched, along with other, more freely available web scrapers.

The proposed system for this project is a web scraper that is able to access and extract data from websites using a web application as an interface for user interaction. The extracted data is then stored in a database, as the web application allows the user to search through and query the saved findings. When the system has been fully implemented, a reflection on the completed system takes place, judging to see if a web scraper can successfully be implemented to combat the issue of human trafficking.

1 Introduction

Since the dawn of the digital age, many people have used the Internet and the World Wide Web (web) in their everyday lives. These users often use the web as a means of communication, with forums, chatrooms and even video hosting sites being the key platforms. Users of the Internet are, for the most part, anonymous, which allows users to openly discuss personal topics and allows users to be free of discrimination, as people are equal under anonymity (Palmer and Berglund, 2004). While encouraging users to become anonymous has allowed the World Wide Web to become a success, it has also allowed criminals to use the web to commit crime while hiding behind a wall of anonymity.

A major crime that is committed using the web as a platform, is modern slavery, or human trafficking. More accurately, the web is used to recruit potential victims, luring them to take risks with the promise of a better life (Logan et al., 2009). In 2012, there was an estimated 20.9 million victims of forced labour in the world, many of which are also victims of human trafficking (ILO, 2012). As this is an estimate, real victims are often hard to spot with current methods of detection, as only 21,251 victims were detected between 2014 and 2016 (UNODC, 2016a).

As human traffickers, and perpetrators of other crimes, use the web to commit offences, specialist tools must be used to track down these instances. One such method is to use a web scraper. Web scrapers are closely related to web crawlers and while there are no real industry definitions for these terms, a distinction is often made. Web crawlers, or spiders, are often used in search engines, as they crawl or scan through a website, looking for links for indexing purposes. Web scrapers may still crawl through a website, however a web scraper is a software that extracts data from webpages and stores the data in a database for further manipulation.

Due to the current number of victims and perpetrators of human trafficking that are unaccounted for, law enforcement agencies must turn to specialised tools, like web scrapers, to have a chance of intercepting these cases. Government bodies and non-governmental organisations alike have been recently researching and developing web scrapers for use in law enforcement, as manually searching through websites is highly inefficient and wastes time and money.

1.1 Aims and objectives

This project sets out to conduct research in the area of web scraping and how it can be used as a tool for law enforcement agencies. The main focus of this project is spotting the potential occurrences of human trafficking online through extracting information from websites, however this can be applied to any crime found on the web.

To do this, a web scraper will be created to automate the process of extracting data and information from websites. Any potential occurrences of human trafficking will be stored in a database so that they can be searched and queried from.

The core aims of this project are:

- To build an understanding of the topics of crime on the Internet, web crawling techniques and web scrapers.
- To create a web scraper to crawl through and extract data relating to human trafficking from websites.
- To log the findings in a database for further manipulation.
- To develop a web application to use as a front end and allow the user to operate the web scraper through an intuitive interface.

The deliverables for this project are:

- Web scraper
- Database
- Web application
- Interim report
- Final report

1.2 Literature review

1.2.1 Crime on the Internet

As the Internet sets out to connect the world, more people than ever are using the Internet and the World Wide Web in their everyday lives. Estimates have found that in 2000, there were around 400 million users of the Internet, while in 2015, there were around 3.2 billion users, an increase of 2.8 billion in 15 years (ITU, 2015). This is shown in figure 1.1 along with the proportions of the populations of developed, developing and least developed countries (LDCs) who have access to the Internet.

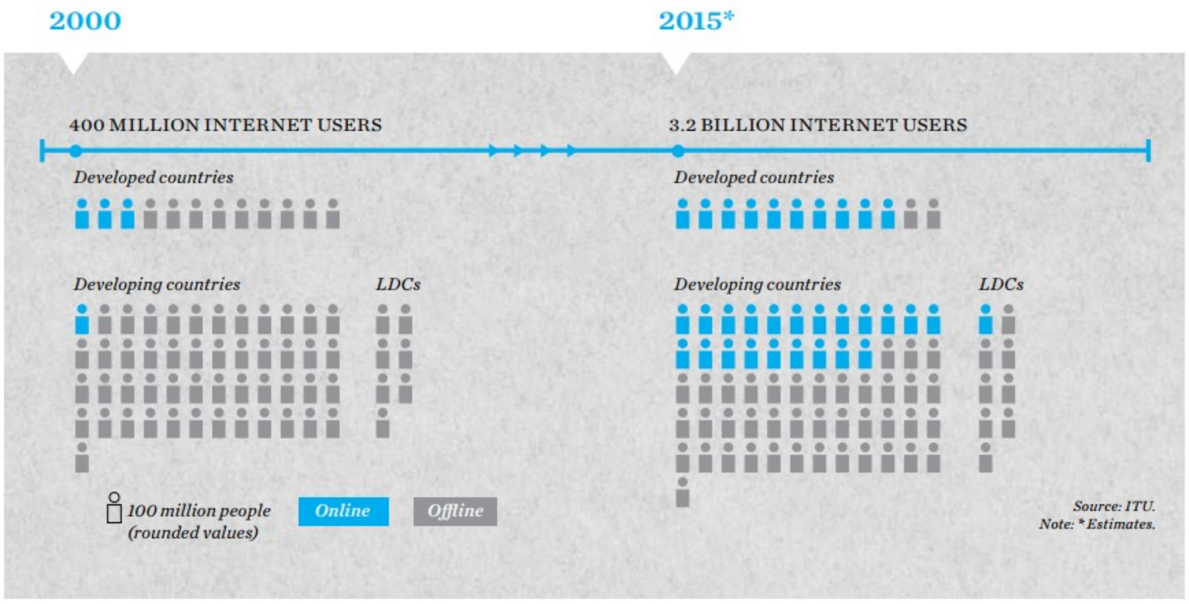
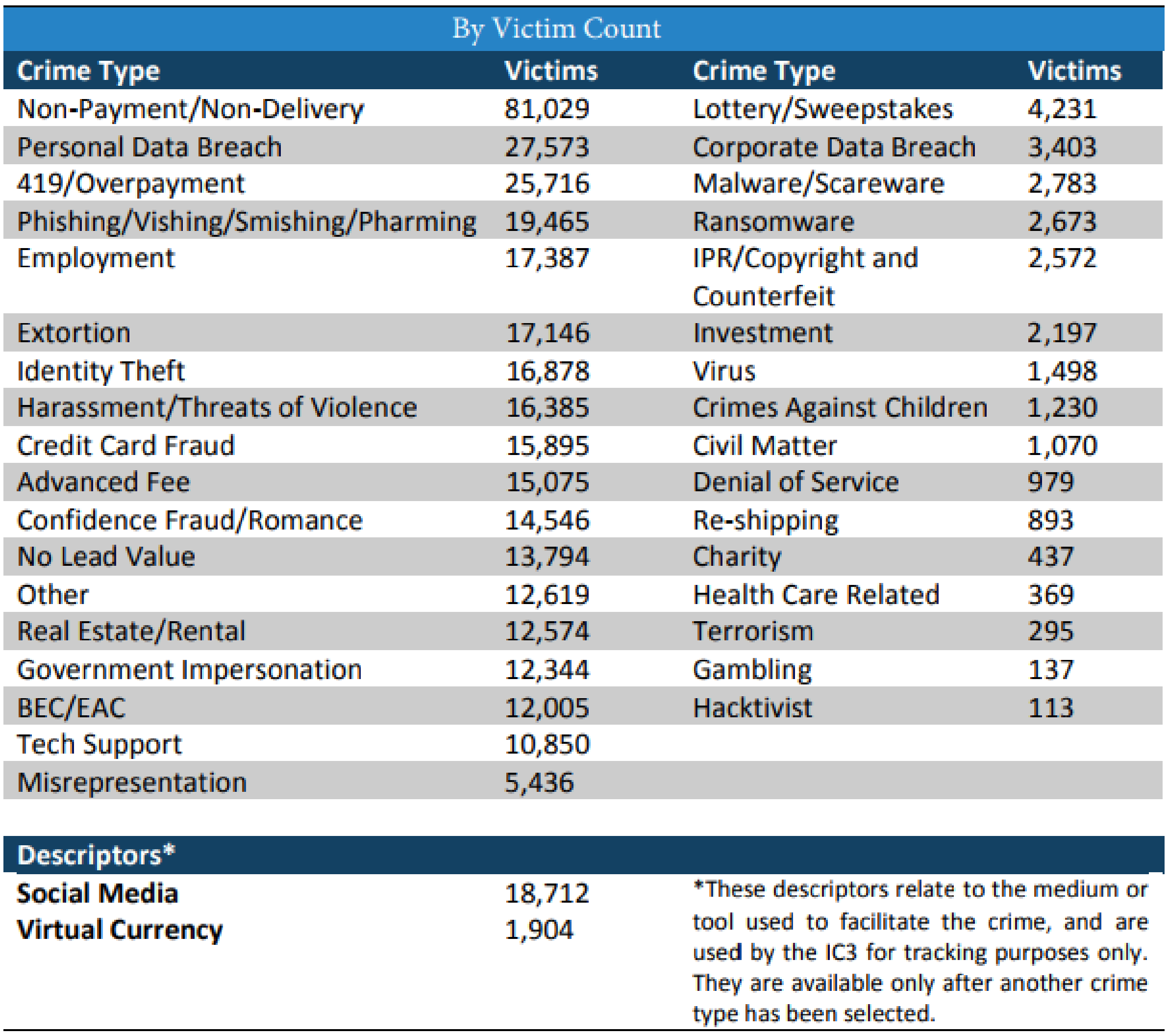


Figure 1.1: Number of users of the Internet from 2000 to 2015 (ITU, 2015)

As the Internet and the World Wide Web continue to grow and amass more users, there is a higher risk of criminal activity. Currently, the web is uncontrolled and unregulated, which allows anyone to utilise it for communication and create their own websites with documents, links and their own content (Albert et al., 1999). This is, in part, due to the lack of centralised control on the web, with no governing bodies to regulate content (Castillo, 2004). What this means is that it is possible for someone with intent to commit a crime to use an existing website or create a website to use as a platform for criminal activities.

There are many types of crime committed on the Internet, ranging from viruses and malware to fraud and extortion. Extortion is defined as “an incident when a cyber criminal demands something of value from a victim by threatening physical or financial harm or the release of sensitive data" (IC3, 2017). This can include denial-of-service (DoS) attacks, government impersonations schemes and sextortion. Sextortion is defined by the Internet Crime Complaint Center (IC3) as "a situation in which someone threatens to distribute your private and sensitive material if you don't provide them images of a sexual nature, sexual favors, or money" (IC3, 2017). Shown in figure 1.2 are types of crime committed in the USA using the Internet as a platform in 2016. It should be noted, however, that this is only from reported crimes, meaning the actual number of victims of these crimes may be higher.



By Victim Count

Crime Type	Victims	Crime Type	Victims
Non-Payment/Non-Delivery	81,029	Lottery/Sweepstakes	4,231
Personal Data Breach	27,573	Corporate Data Breach	3,403
419/Overpayment	25,716	Malware/Scareware	2,783
Phishing/Vishing/Smishing/Pharming	19,465	Ransomware	2,673
Employment	17,387	IPR/Copyright and Counterfeit	2,572
Extortion	17,146	Investment	2,197
Identity Theft	16,878	Virus	1,498
Harassment/Threats of Violence	16,385	Crimes Against Children	1,230
Credit Card Fraud	15,895	Civil Matter	1,070
Advanced Fee	15,075	Denial of Service	979
Confidence Fraud/Romance	14,546	Re-shipping	893
No Lead Value	13,794	Charity	437
Other	12,619	Health Care Related	369
Real Estate/Rental	12,574	Terrorism	295
Government Impersonation	12,344	Gambling	137
BEC/EAC	12,005	Hacktivist	113
Tech Support	10,850		
Misrepresentation	5,436		

Descriptors*		
Social Media	18,712	*These descriptors relate to the medium or tool used to facilitate the crime, and are used by the IC3 for tracking purposes only. They are available only after another crime type has been selected.
Virtual Currency	1,904	

Figure 1.2: Types of crime in 2016 in the USA by number of victims (IC3, 2017)

From figure 1.2, it can be seen that the most common medium used to facilitate crimes is social media, with 18,712 victims in 2016 (IC3, 2017). What this shows is that criminals would often use communication tools to commit offences. One such offence is human trafficking.

There have been numerous estimates of the number of victims of human trafficking in the world. In 2012, there were an estimated 20.9 million victims worldwide (ILO, 2012), while the United Nations High Commissioner for Refugees (UNHCR) estimated that in 2015, more than 65 million victims may exist in the world (UNODC, 2016b). These, however, are estimates, as the number of detected victims between 2014 and 2016 was only 21,251 (UNODC, 2016a). What this shows is that with current methods of detection, many victims go unnoticed. How many people become victims of trafficking is through the promise of a better life. Victims are told to forward their information to make forged documents with the promise that they will have a better life in a new country (Logan et al., 2009). These are simply promises, as in reality, the victims are often paid equivalent to the minimum standard wages in their countries of origin (Europol, 2017).

As the world moves to make use of the Internet and the web in everyday lives, traffickers have also moved to this digital platform. Traffickers are using social media and web forums as online advertisements to recruit victims, and using the web to purchase tickets to transport the victims (McAlister, 2015).

Due to the large proportion of victims of human trafficking who are undetected by the United Nations (UN), the 2015 estimated over 65 million victims (UNODC, 2016b) versus the 21,251 detected victims between 2014 and 2016 (UNODC, 2016a), it can be gathered that the current methods of detection are not able to spot victims who are recruited online.

1.2.2 Surface Web, Deep Web and Dark Web

The web has been estimated to be as large as 6 zettabytes in 2014, constantly growing larger as more content is produced (Zhao et al., 2016). As the web is a large collection of documents, difficulties can arise when trying to index the entire World

Wide Web. Because of this, the web is generally divided into three subsections; the surface web, the deep web and the dark web. An analogy of an iceberg is commonly used when describing the web, with the surface web being the tip of the iceberg, the deep web being the rest of the iceberg lying underwater, and the dark web being the deepest part, lying within the deep web.

The surface web consists of webpages and documents that have been or can be indexed by search engines. This mostly consists of unstructured HTML (Hypertext Markup Language) text and images, with billions of links between them (Bin et al., 2007). Although crime can be easily detected on the surface web, platforms still exist selling counterfeit goods and drugs, among others, and can be found on social media or by using a simple search on a browser (Europol, 2017).

The deep web, not to be confused with the dark web, consists of the largest portion of the web making up 96% of content on the Internet and is 500 to 550 times the size of the surface web, the majority of which is estimated to be stored as data in databases (Zhao et al., 2016). Any website that contains search queries, where a user sends queries to and interacts with an underlying database, is classified as the deep web. As search engines cannot effectively crawl through databases, this content is mostly hidden from users (Bin et al., 2007). What this means is that web crawlers used for indexing, such as a search engine, cannot crawl and index deep web content, as forms would need to be submitted.

The dark web is a section of the deep web that is only accessible through the use of specialised software, such as The Onion Router (TOR), I2P or Freenet. These tools were originally to be used for protecting privacy, business activities and relationships by allowing users to become anonymous, however this opened the door to criminals (Europol, 2017). Criminals typically use the dark web for a variety of activities such as human trafficking, child sexual exploitation, trafficking of drugs and firearms. Shown in figure 1.3 is a representation of the web as an iceberg, showing the types of crime that are commonly found on the surface web and the dark web, or darknet. It can be noted that the severity of the crimes committed increases when moving from the surface web to the dark web.

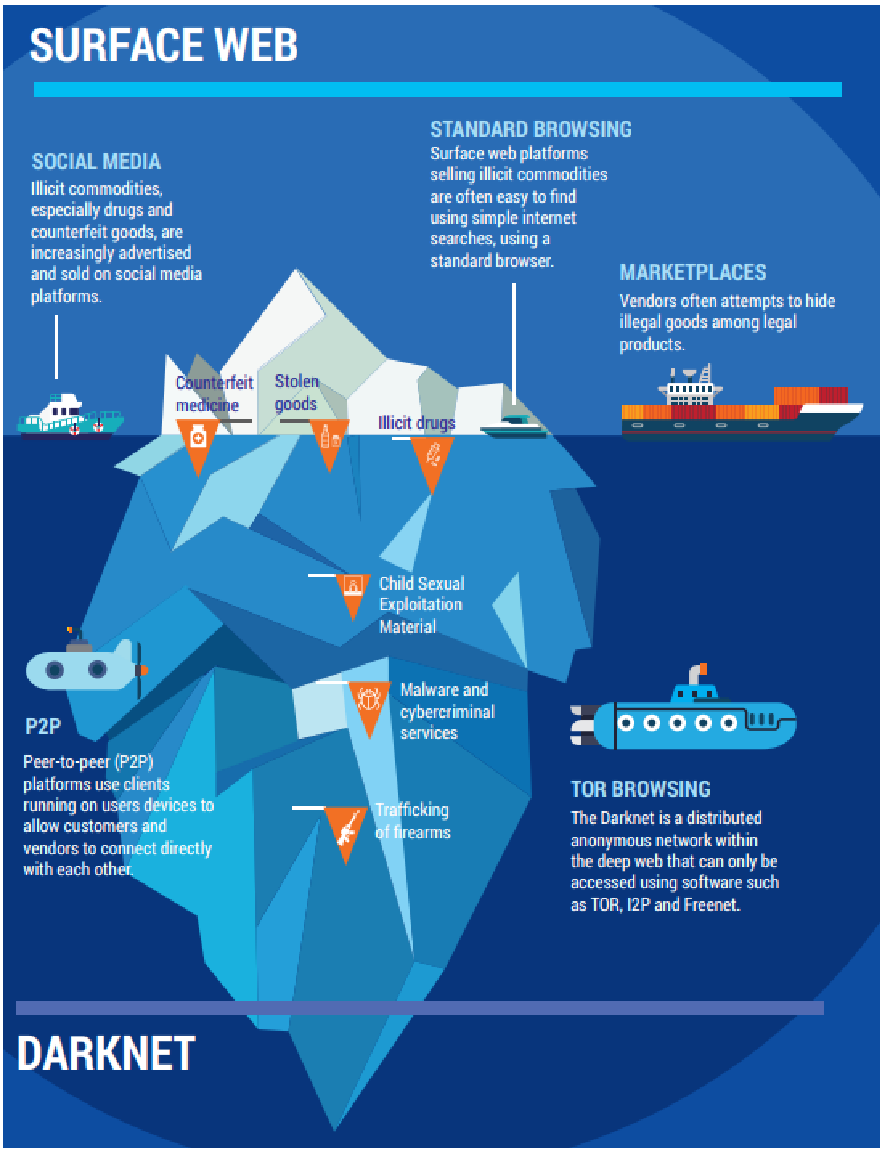

SURFACE WEB
SOCIAL MEDIA
Illicit commodities, especially drugs and counterfeit goods, are increasingly advertised and sold on social media platforms.
STANDARD BROWSING
Surface web platforms selling illicit commodities are often easy to find using simple internet searches, using a standard browser.
MARKETPLACES
Vendors often attempts to hide illegal goods among legal products.
Counterfeit medicine
Stolen goods
Illicit drugs
Child Sexual Exploitation Material
Malware and cybercriminal services
Trafficking of firearms
P2P
Peer-to-peer (P2P) platforms use clients running on users devices to allow customers and vendors to connect directly with each other.
TOR BROWSING
The Darknet is a distributed anonymous network within the deep web that can only be accessed using software such as TOR, I2P and Freenet.
DARKNET

Figure 1.3: Structure of the web as an iceberg (Europol, 2017)

1.2.3 Web Scrapers

As the Internet and the web continue to expand, it can become difficult to access webpages without knowing beforehand the address of the page. This is where search engines come in. Search engines use a process called web crawling, which is an algorithm designed to scan or crawl through a collection of websites, which is indexed and searched (Castillo, 2004). This algorithm has three main components; a webpage is fetched, it is parsed to extract all linked URLs (Uniform Resource

Locator), and for previously unseen URLs, repeat the first two steps (Broder et al.,2003).

A web scraper differs from a web crawler, as a web crawler simply crawls and indexes, while a web scraper is an automated tool that queries a web server to fetch a webpage, and parses the webpage to extract information (Mitchell, 2015). While a web crawler scans many webpages to find links, the format of those links remains the same, however, when using a web scraper to extract data from webpages, the format of the markup changes between different websites (Penman et al., 2009). Despite this, web scrapers are often used to access areas of a website that search engines cannot. For example, when looking for flights, a search engine will only find what the websites say on their content pages, while a web scraper can query the underlying databases and chart the cost of flights over time from a variety of websites (Mitchell, 2015).

Due to this, web scrapers can be very useful as an investigative tool for law enforcement, as, in the case of human trafficking, they can be used to detect signs of trafficking in the recruitment stage before any exploitation takes place (McAlister, 2015).

1.2.4 Memex

The Defense Advanced Research Projects Agency (DARPA) launched the Memex programme in 2014, which "seeks to develop software that advances online search capabilities far beyond the current state of the art" (Shen, 2014). What this means is that Memex is setting out to address the limitations in conventional search engines, such as the manual nature of searching one search term at a time and the lack of indexing of the deep web.

As search engines use web crawlers to crawl through webpages and find links to index, Memex aims to use a web scraper to crawl through websites and store information. This project sets out to improve upon typical search engines, which are limited to the surface web, as they cannot access the deep web by submitting forms to gain entry to the underlying database of a website. It plans to do this by:

- Develop search technologies to discover, organise and present domain-specific content.
- Create a domain-specific algorithm for relevant content discovery and organise it so that it is useful for specific tasks.
- Extend the capabilities of searches on the deep web and multimedia content.
- Allow military, government and commercial enterprises to easily find and organise publicly available content on the Internet.

(Shen, 2014).

While the broader goal is to apply Memex to any public domain content, the main target of this project is human trafficking (Shen, 2014), as the project aims to identify traffickers and buyers, and help victims of trafficking (Mattmann, 2015). As traffickers use social media, forums, advertisements, etc. to propagate the industry of modern slavery (Shen, 2014), an interactive interface for crawling text, images and video is under development, which automatically detects people and objects, to spot traffickers and victims, combatting the issue of human trafficking (Mattmann, 2015).

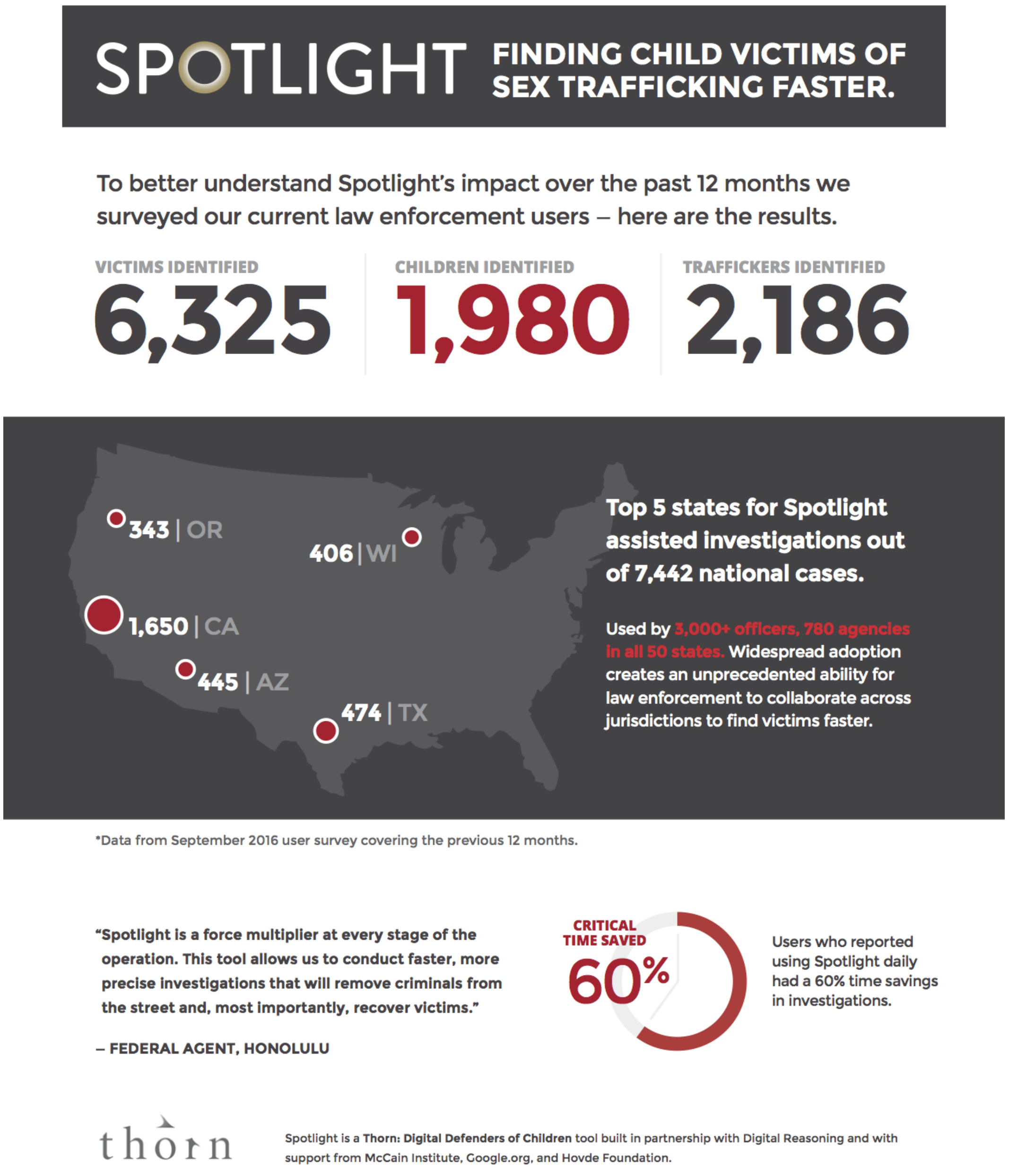
The Memex programme, while currently in development, is a form of search engine with additional capabilities for the deep web and dark web, being able to access and extract data from databases and perform facial recognition on scraped images and videos. While it aims to be a general purpose web scraper, the focus of the project is to detect signs of human trafficking for law enforcement.

1.2.5 Spotlight

Thorn is an organisation dedicated to finding victims of abuse and human trafficking using a technology-driven approach. The Spotlight programme is designed to scrape the dark web to identify victims and since 2016, it identified more than 40 victims of child sexual abuse (Cordua, 2017).

Spotlight is currently in use in law enforcement across all states in the USA and has been used to identify 6,325 victims of human trafficking and abuse; 4,345 adults and 1,980 children, along with 2,186 traffickers. As Spotlight saves 60% of the time spent investigating through the use of automated workflows, it can be gathered that it is an efficient tool for extracting and analysing data from the dark web (Boorse, 2016).

Figure 1.4 shows the number of victims and traffickers detected over 7,442 cases using the Spotlight web scraper between September 2015 and September 2016.


SPOTLIGHT FINDING CHILD VICTIMS OF SEX TRAFFICKING FASTER.

To better understand Spotlight's impact over the past 12 months we surveyed our current law enforcement users – here are the results.

VICTIMS IDENTIFIED	CHILDREN IDENTIFIED	TRAFFICKERS IDENTIFIED
6,325	1,980	2,186

Top 5 states for Spotlight assisted investigations out of 7,442 national cases.

Used by 3,000+ officers, 780 agencies in all 50 states. Widespread adoption creates an unprecedented ability for law enforcement to collaborate across jurisdictions to find victims faster.

*Data from September 2016 user survey covering the previous 12 months.

"Spotlight is a force multiplier at every stage of the operation. This tool allows us to conduct faster, more precise investigations that will remove criminals from the street and, most importantly, recover victims."

– FEDERAL AGENT, HONOLULU

CRITICAL TIME SAVED 60%

Users who reported using Spotlight daily had a 60% time savings in investigations.

thorn

Spotlight is a Thorn: Digital Defenders of Children tool built in partnership with Digital Reasoning and with support from McCain Institute, Google.org, and Hovde Foundation.

Figure 1.4: Infographic showing how Spotlight was used from September 2015 to 2016 (Boorse, 2016)

From Spotlight's ability to scrape the dark web in search of information relating to human trafficking, including anonymous peer-to-peer (P2P) networks and identifying victims who frequently change location (Thorn, 2017), it seems to be a very robust tool, able to scrape information that is intentionally hidden and identify those who are anonymous.

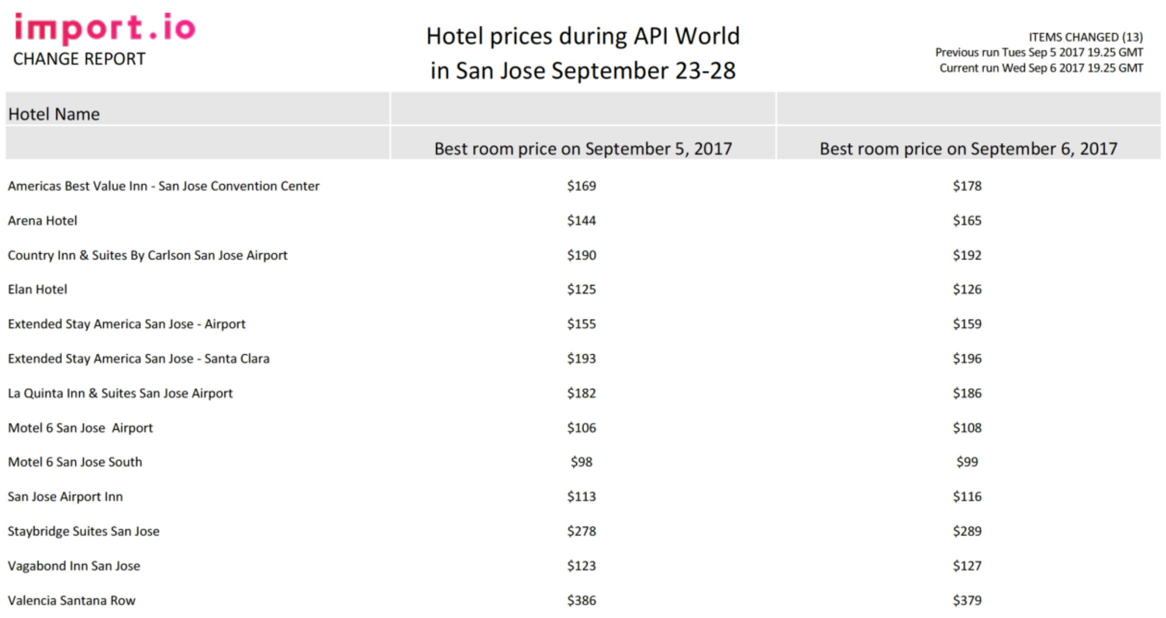
1.2.6 Google's Human Trafficking Group

As Google is known for its search engine, the company has experience with web crawlers. Since 2014, Google has allowed victims of human trafficking to connect to organisations that can help with their issue through the search engine (Google, 2017). This has expanded to the employment of technology that can identify trafficking networks to help with law enforcement cases (Molinari, 2017).

One such tool is used to ban ads relating to human trafficking using automated tools, such as a web scraper, to detect the infringing ads (Google, 2017). This web scraper searches through ads on Google's ad systems and identifies ads related to slavery and human trafficking based on their content and stores this information to be manually checked by experts for violations of Google's ad policies (Google, 2017).

1.2.7 Import.io

Import.io is a web scraping software targeted towards businesses to gather trends and data. This software is a SaaS (Software as a Service) product, meaning it is used through a paid subscription, and allows users to extract and organise data from websites through the use of the interface (Import.io, 2017). Shown in figure 1.5 is a representation of how data can be displayed when comparing data scraped from hotel websites.



import.io
CHANGE REPORT

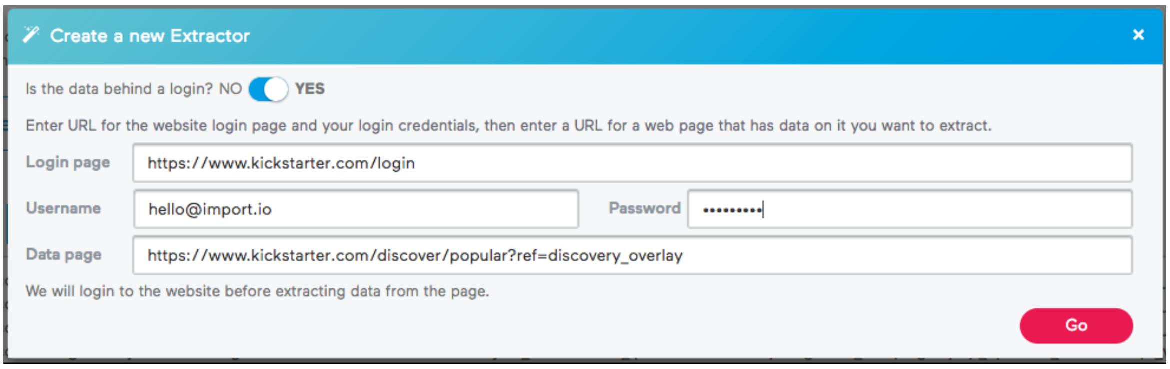
Hotel prices during API World in San Jose September 23-28

ITEMS CHANGED (13)
Previous run Tues Sep 5 2017 19.25 GMT
Current run Wed Sep 6 2017 19.25 GMT

Hotel Name		
	Best room price on September 5, 2017	Best room price on September 6, 2017
Americas Best Value Inn - San Jose Convention Center	$169	$178
Arena Hotel	$144	$165
Country Inn & Suites By Carlson San Jose Airport	$190	$192
Elan Hotel	$125	$126
Extended Stay America San Jose - Airport	$155	$159
Extended Stay America San Jose - Santa Clara	$193	$196
La Quinta Inn & Suites San Jose Airport	$182	$186
Motel 6 San Jose Airport	$106	$108
Motel 6 San Jose South	$98	$99
San Jose Airport Inn	$113	$116
Staybridge Suites San Jose	$278	$289
Vagabond Inn San Jose	$123	$127
Valencia Santana Row	$386	$379

Figure 1.5: Comparison of hotel prices between September 6th to 7th 2017 (Import.io, 2017)

The Import.io web scraper can also be used to access the deep web, as queries to a database can be made and login credentials can be submitted to extract data directly from the underlying database of a website. This includes requesting flight ticket prices or images and other multimedia content (Import.io, 2017). Shown in figure 1.6 is the interface used by Import.io to allow users to submit login credentials to scrape information that is hidden behind a login.



Create a new Extractor

Is the data behind a login? NO YES

Enter URL for the website login page and your login credentials, then enter a URL for a web page that has data on it you want to extract.

Login page: https://www.kickstarter.com/login

Username: hello@import.io Password: •••••••••

Data page: https://www.kickstarter.com/discover/popular?ref=discovery_overlay

We will login to the website before extracting data from the page.

Figure 1.6: Interface for submitting login credentials (Import.io, 2017)

Import.io has many features for a web scraper. It can extract text and images from database search queries and from behind login forms. This access to the deep web gives the user a greater variety of options when scraping data, as it is not limited to the surface web like a search engine web crawler.

1.2.8 Web Scraper (webscraper.io)

While most web scrapers are standalone programs, Web Scraper (not to be confused with the term) is a browser extension for Google Chrome. This software is used via the developer tools on the browser and allows users to determine the structure of the website to be scraped and build a sitemap, telling the program how to traverse the website (Balodis, 2017). Figure 1.7 shows a selector graph, the visual representation of the sitemap, of a test ecommerce website (http://webscraper.io/test-sites/e-commerce/allinone), detailing the path to traverse to find an item's title, price and description.

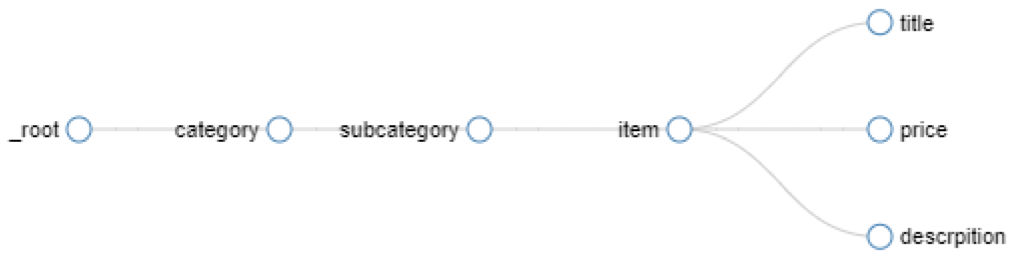
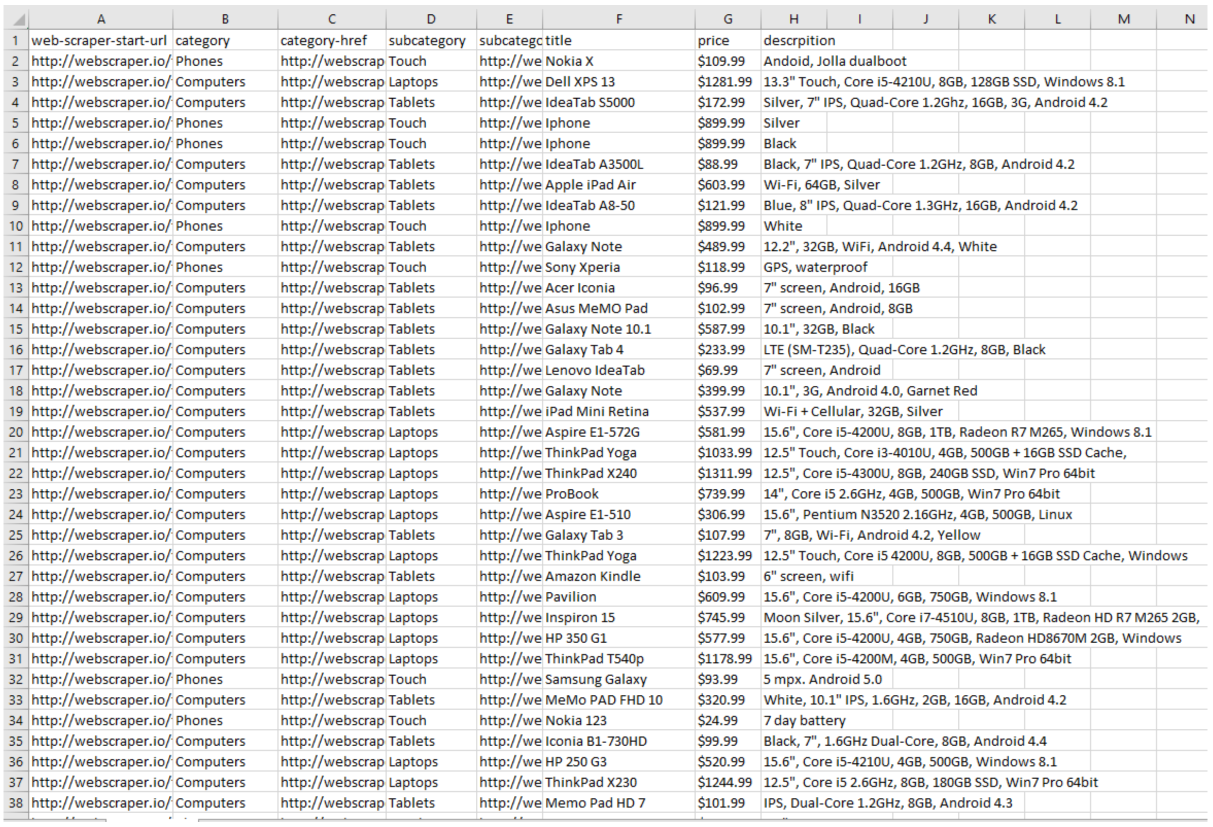


Figure 1.7: Ecommerce website selector graph in Web Scraper

After scraping the desired data from a website, Web Scraper is able to store the findings in a local database, saved as a CSV (Comma-Separated Values) file, or as a CouchDB database to be opened in the Google Chrome browser (Balodis, 2017). Shown is figure 1.8 is an exported CSV file detailing the data scraped from the test ecommerce website, the headings of which correspond to each node of the selector graph in figure 1.7.



	A	B	C	D	E	F	G	H
1	web-scraper-start-url	category	category-href	subcategory	subcategc	title	price	descrpition
2	http://webscraper.io/	Phones	http://webscrap	Touch	http://we	Nokia X	$109.99	Andoid, Jolla dualboot
3	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	Dell XPS 13	$1281.99	13.3" Touch, Core i5-4210U, 8GB, 128GB SSD, Windows 8.1
4	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	IdeaTab S5000	$172.99	Silver, 7" IPS, Quad-Core 1.2Ghz, 16GB, 3G, Android 4.2
5	http://webscraper.io/	Phones	http://webscrap	Touch	http://we	Iphone	$899.99	Silver
6	http://webscraper.io/	Phones	http://webscrap	Touch	http://we	Iphone	$899.99	Black
7	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	IdeaTab A3500L	$88.99	Black, 7" IPS, Quad-Core 1.2GHz, 8GB, Android 4.2
8	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Apple iPad Air	$603.99	Wi-Fi, 64GB, Silver
9	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	IdeaTab A8-50	$121.99	Blue, 8" IPS, Quad-Core 1.3GHz, 16GB, Android 4.2
10	http://webscraper.io/	Phones	http://webscrap	Touch	http://we	Iphone	$899.99	White
11	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Galaxy Note	$489.99	12.2", 32GB, WiFi, Android 4.4, White
12	http://webscraper.io/	Phones	http://webscrap	Touch	http://we	Sony Xperia	$118.99	GPS, waterproof
13	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Acer Iconia	$96.99	7" screen, Android, 16GB
14	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Asus MeMO Pad	$102.99	7" screen, Android, 8GB
15	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Galaxy Note 10.1	$587.99	10.1", 32GB, Black
16	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Galaxy Tab 4	$233.99	LTE (SM-T235), Quad-Core 1.2GHz, 8GB, Black
17	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Lenovo IdeaTab	$69.99	7" screen, Android
18	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Galaxy Note	$399.99	10.1", 3G, Android 4.0, Garnet Red
19	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	iPad Mini Retina	$537.99	Wi-Fi + Cellular, 32GB, Silver
20	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	Aspire E1-572G	$581.99	15.6", Core i5-4200U, 8GB, 1TB, Radeon R7 M265, Windows 8.1
21	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	ThinkPad Yoga	$1033.99	12.5" Touch, Core i3-4010U, 4GB, 500GB + 16GB SSD Cache,
22	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	ThinkPad X240	$1311.99	12.5", Core i5-4300U, 8GB, 240GB SSD, Win7 Pro 64bit
23	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	ProBook	$739.99	14", Core i5 2.6GHz, 4GB, 500GB, Win7 Pro 64bit
24	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	Aspire E1-510	$306.99	15.6", Pentium N3520 2.16GHz, 4GB, 500GB, Linux
25	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Galaxy Tab 3	$107.99	7", 8GB, Wi-Fi, Android 4.2, Yellow
26	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	ThinkPad Yoga	$1223.99	12.5" Touch, Core i5 4200U, 8GB, 500GB + 16GB SSD Cache, Windows
27	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Amazon Kindle	$103.99	6" screen, wifi
28	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	Pavilion	$609.99	15.6", Core i5-4200U, 6GB, 750GB, Windows 8.1
29	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	Inspiron 15	$745.99	Moon Silver, 15.6", Core i7-4510U, 8GB, 1TB, Radeon HD R7 M265 2GB,
30	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	HP 350 G1	$577.99	15.6", Core i5-4200U, 4GB, 750GB, Radeon HD8670M 2GB, Windows
31	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	ThinkPad T540p	$1178.99	15.6", Core i5-4200M, 4GB, 500GB, Win7 Pro 64bit
32	http://webscraper.io/	Phones	http://webscrap	Touch	http://we	Samsung Galaxy	$93.99	5 mpx. Android 5.0
33	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	MeMo PAD FHD 10	$320.99	White, 10.1" IPS, 1.6GHz, 2GB, 16GB, Android 4.2
34	http://webscraper.io/	Phones	http://webscrap	Touch	http://we	Nokia 123	$24.99	7 day battery
35	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Iconia B1-730HD	$99.99	Black, 7", 1.6GHz Dual-Core, 8GB, Android 4.4
36	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	HP 250 G3	$520.99	15.6", Core i5-4210U, 4GB, 500GB, Windows 8.1
37	http://webscraper.io/	Computers	http://webscrap	Laptops	http://we	ThinkPad X230	$1244.99	12.5", Core i5 2.6GHz, 8GB, 180GB SSD, Win7 Pro 64bit
38	http://webscraper.io/	Computers	http://webscrap	Tablets	http://we	Memo Pad HD 7	$101.99	IPS, Dual-Core 1.2GHz, 8GB, Android 4.3

Figure 1.8: Exported CSV of scraped data using Web Scraper

While Web Scraper can be a fast and free method of extracting information from websites, it is limited. This limitation is the lack of deep web support, as the tool cannot query a database or submit login credentials, which limits the scope of the tool.

1.3 Summary of the report

Chapter 2 of this report draws a conclusion to the research conducted in the literature review in chapter 1. From this conclusion, a set of requirements are created, including functional, non-functional, hardware and software requirements. Multiple development methodologies are proposed and analysed, with the methodology best suited to this project being chosen. The scope of the project is determined and shown using a Gantt chart. Using these requirements, the system design and architecture are developed, including the database schema and class specification in chapter 3. Chapter 4 focuses on the implementation of the proposed designs, delivering a working spider and web application. Testing of the developed software is then carried out to conclude the development phase. Chapter 5 then reflects back on the processes involved during this project and evaluates the product against the specified requirements.

2 Analysis

Conclusions can be drawn from the research conducted in the previous chapter. Many crimes are committed using the Internet as a platform, one of the most common methods is via social media. In the USA in 2016, there were 18,712 detected victims of crime committed using social media platforms (IC3, 2017). What this shows is that many crimes committed are in plain view in the surface web, allowing this information to be easily scraped.

Other crimes, such as human trafficking, is more prominent in the dark web, so more specialised web scrapers, such as Spotlight and Memex, are used by governmental agencies and law enforcement agencies. As these web scrapers are for use by law enforcement, they are not available to the public or to anyone who requests to sample them, meaning very little information is known on how and where they operate. Due to this, more accessible options were investigated, such as Import.io and Web Scraper (webscraper.io), to research how web scrapers typically function.

The findings of this indicate that it is common for web scrapers to access the deep web through database queries and form submission. As the vast majority of the web is part of the deep web, estimated to be 96% (Zhao et al., 2016), it is important for a web scraper to access this to be able to extract the most meaningful information.

For the creation of the web scraper and the web application, several tools must be utilised. To keep all code and software produced in a safe environment with maximum compatibility, a virtual machine (VM) running the Ubuntu 16.04.3 distribution of Linux. This VM is used for portability so that when moving between machines, the VM box can be copied and transferred to the new machine. For the creation of the web scraper and the web application, Python 3.6.3 will be used, with Scrapy 1.4 for the web scraper, and Flask 0.12.2 and Twitter Bootstrap v4.0 for the web application. The latest stable releases of each language and software will be used to ensure code will be up to date and will not be deprecated in the foreseeable future.

Based on the research conducted in the previous chapter, requirements can be produced.

The functional requirements of the project are:

- The system should prompt the user to log in before use for confidentiality, as the data is sensitive.
- The system should allow the user to enter a website to scrape.
- The system should crawl through and extract data from the desired website.
- The system should store data in a database with timestamps.
- The system should allow users to access and query the database from within the web application.

The non-functional requirements of the project are:

- The system should be user friendly and easy to navigate.
- The system should be easily maintained with minimal maintenance required.
- The system should be scalable and work efficiently under heavy workloads.
- The system should be easy to install and set up.
- The handling of user and scraped data should conform to the Data Protection Act 1998.

The hardware requirements of the project are:

- A PC that is connected to a network and can be used as a server to host the application.

The software requirements of the project are:

- An Ubuntu 16.04.3 VM as a safe environment for compatibility and portability.
- Python 3.6.3, Scrapy 1.4, Flask 0.12.2, Twitter Bootstrap v4.0 and their dependencies.
- Sublime Text 3 for use in development.
- Web browsers for testing, primarily Google Chrome, Firefox and Microsoft Edge.

When planning a project, it is important to use a systems development life cycle (SDLC) to successfully take a project through the various stages of development:

- Planning
- Analysis
- Design
- Implementation
- Testing

While planning a project, there are two main SDLC methodologies available; waterfall and agile.

Waterfall Methodology

The waterfall methodology is a traditional type of SDLC that functions much like a waterfall when moving from one stage to the next. This greatly reduces flexibility, as when one stage is complete, development moves on to the next stage and rarely goes back. As requirements may change during a project, the waterfall method is unable to deal with this and must either carry on to the end or start over from the planning stage. Shown in figure 2.1 is the waterfall method and how it rigidly moves from one phase to the next.

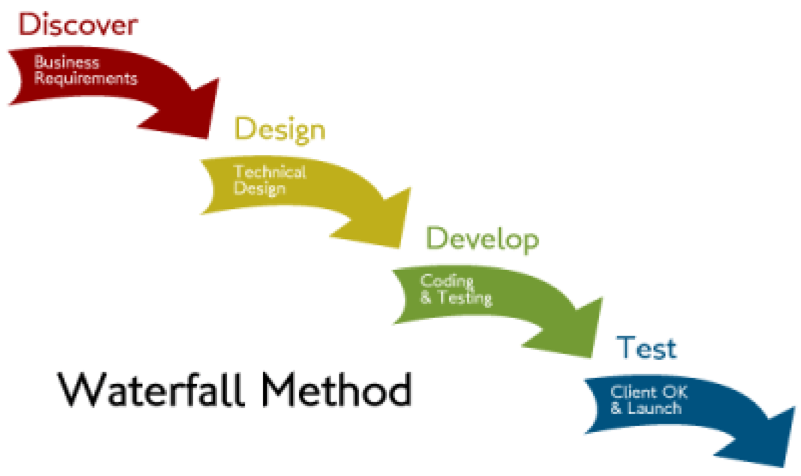


Figure 2.1: Representation of the Waterfall Methodology

Agile Methodology

The agile methodology is used when the requirements may change during the development of a project. Projects using agile development are broken up into sprints, each with their own analysis, design, implementation and testing phases. These sprints begin with a meeting, detailing the work to be done in the sprint with daily meetings to keep the project on target. Essentially, this breaks the project into smaller segments with each segment having its own development phase, then at the end, the segments are combined to form the finished product. Figure 2.2 shows the agile method as a series of sprints with their own development life cycles.

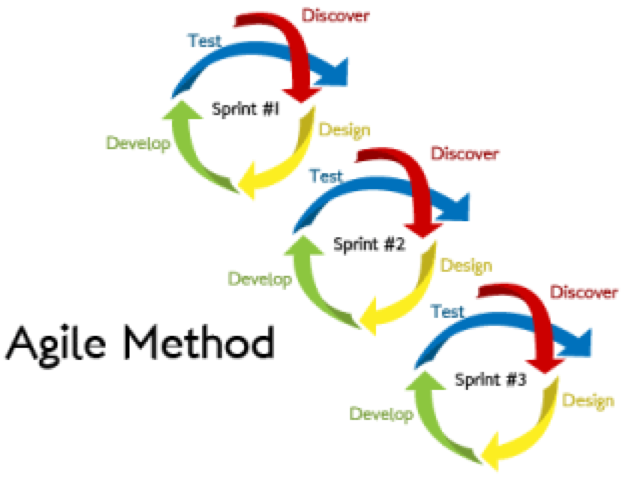
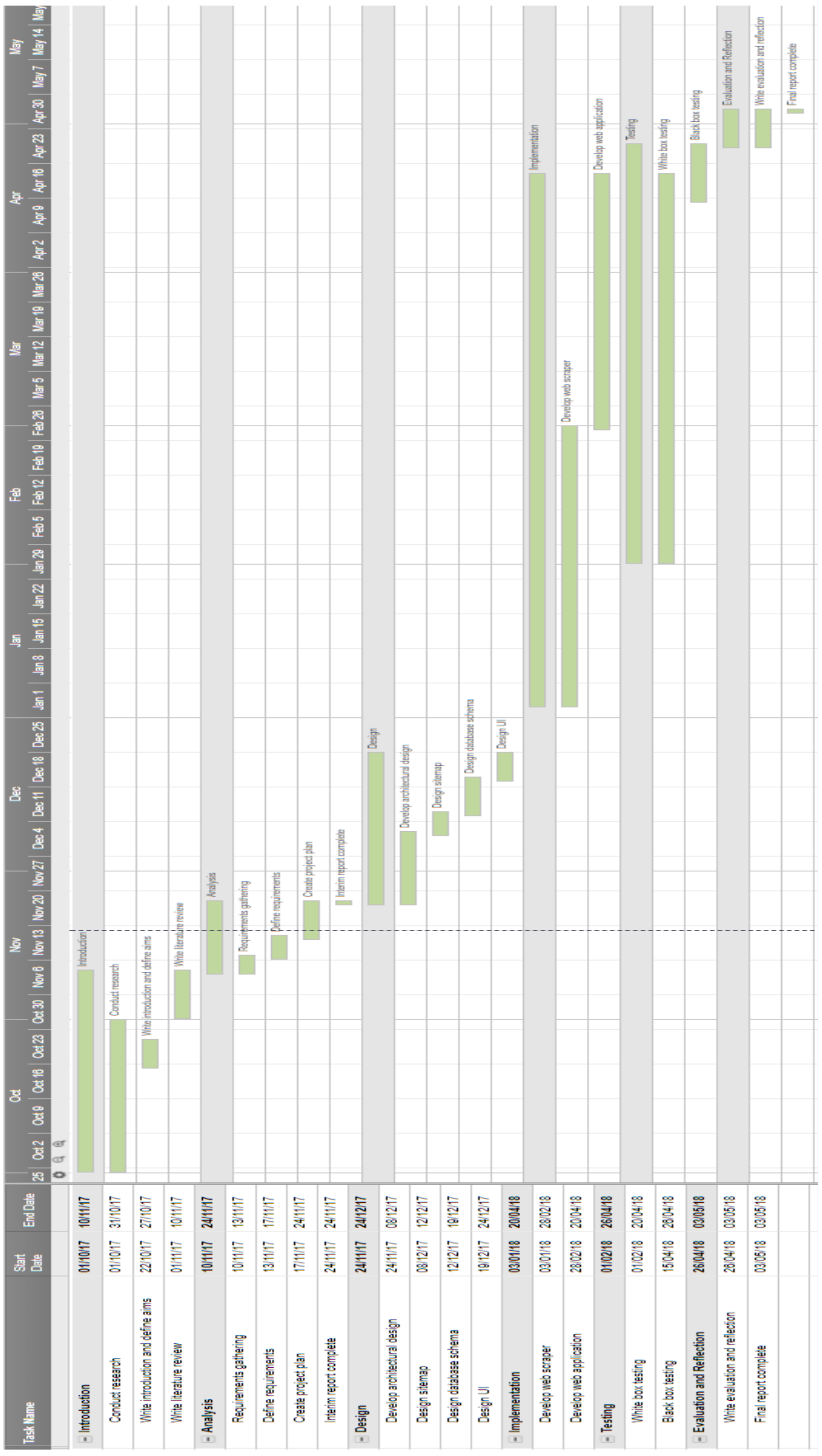


Figure 2.2: Representation of the Agile Methodology

For this project, the agile methodology will be utilised. As there is no project team, meetings on each sprint cannot be held, however, the method of breaking down the project into smaller, more manageable segments will be used. This will allow the project to be able to adapt to any changes in the requirements. As this project consists of a web crawler and a web application, the agile methodology is better suited to handling multiple deliverables than the waterfall method.

To keep track of development, a Gantt chart, shown in figure 2.3, has been developed to make sure each step of the project is on target.



Task Name	Start Date	End Date
Introduction	**01/10/17**	**10/11/17**
Conduct research	01/10/17	31/10/17
Write introduction and define aims	22/10/17	27/10/17
Write literature review	01/11/17	10/11/17
Analysis	**10/11/17**	**24/11/17**
Requirements gathering	10/11/17	13/11/17
Define requirements	13/11/17	17/11/17
Create project plan	17/11/17	24/11/17
Interim report complete	24/11/17	24/11/17
Design	**24/11/17**	**24/12/17**
Develop architectural design	24/11/17	08/12/17
Design sitemap	08/12/17	12/12/17
Design database schema	12/12/17	19/12/17
Design UI	19/12/17	24/12/17
Implementation	**03/01/18**	**20/04/18**
Develop web scraper	03/01/18	28/02/18
Develop web application	28/02/18	20/04/18
Testing	**01/02/18**	**26/04/18**
White box testing	01/02/18	20/04/18
Black box testing	15/04/18	28/04/18
Evaluation and Reflection	**26/04/18**	**03/05/18**
Write evaluation and reflection	28/04/18	03/05/18
Final report complete	03/05/18	03/05/18

Figure 2.3: Gantt chart showing the stages of the project

3 Design

3.1 Use Case

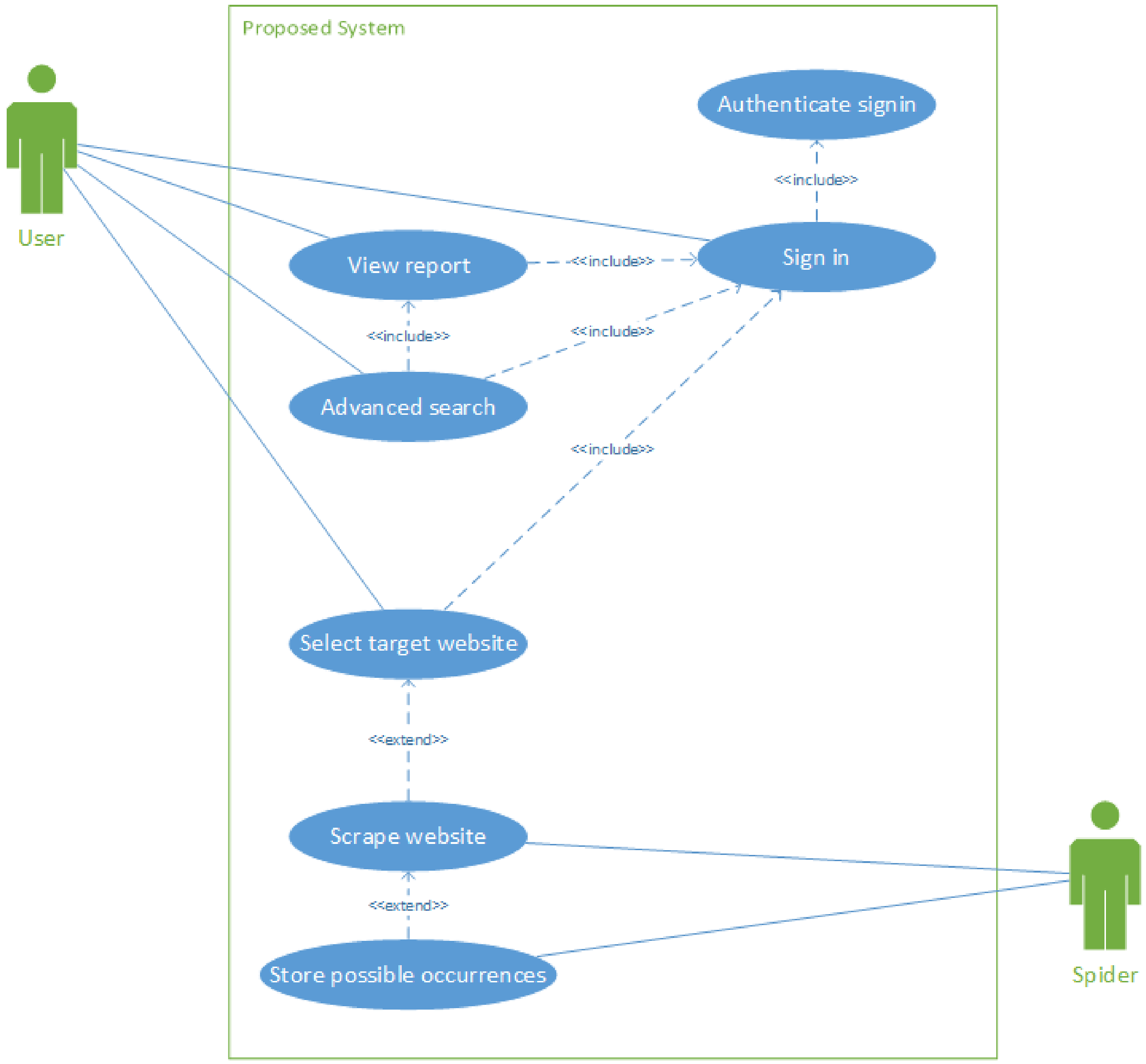


Figure 3.1: Use case diagram of the proposed system

Shown in figure 3.1 is the use case diagram for this proposed web application. See Appendix A for the use case descriptions.

3.2 System Architecture

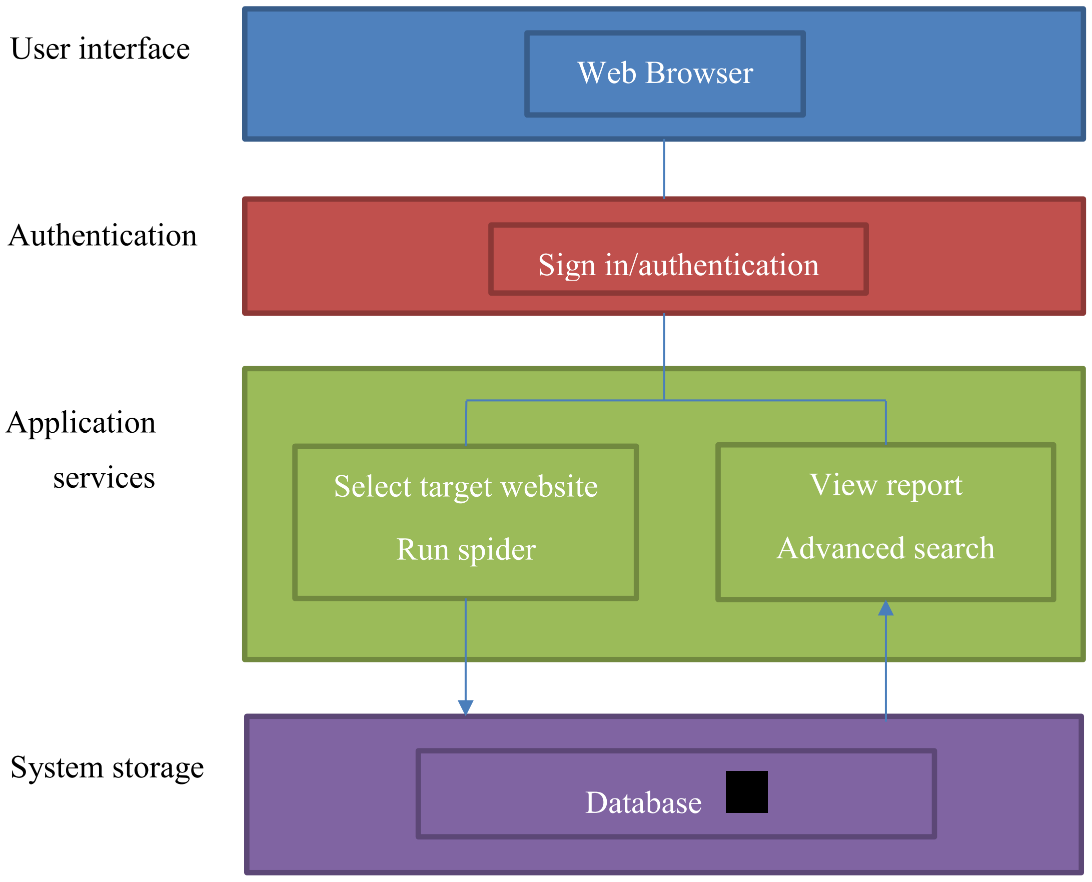


Figure 3.2: Architecture diagram of the proposed system

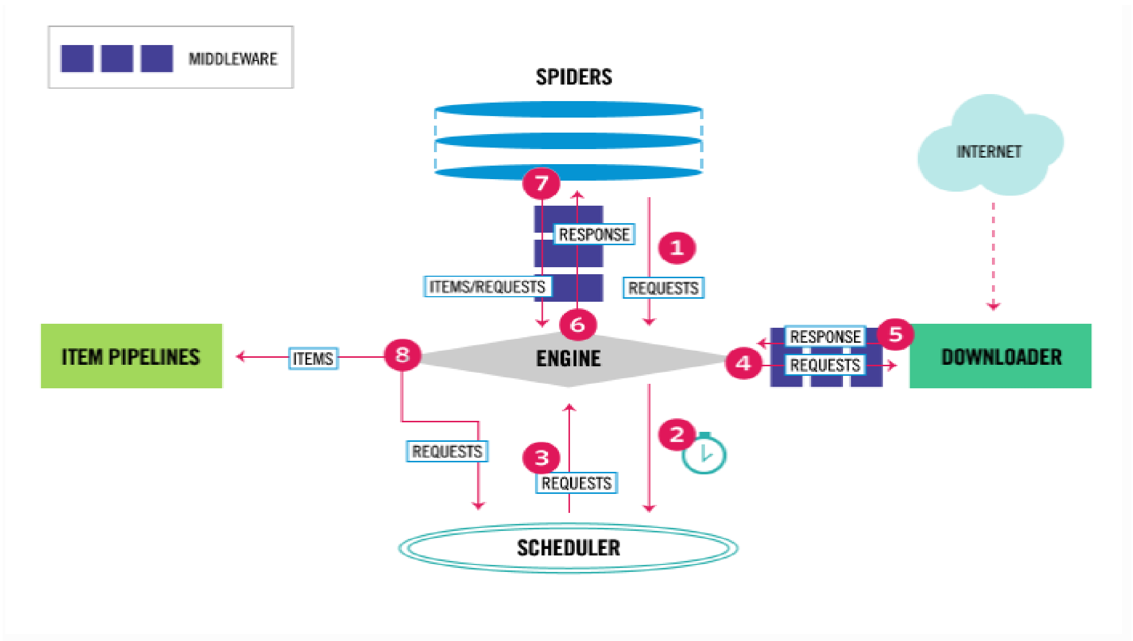
Shown in figure 3.2 is the architecture diagram for the proposed web scraping system. Due to the need for a user to be authenticated before accessing the web application, a layered architecture was chosen, as each layer can only communicate with the layer one above or below. This means that the core functionality of the web application can take place close to the database after authentication, as the application makes use of information stored in the database for all aspects of the application services layer.

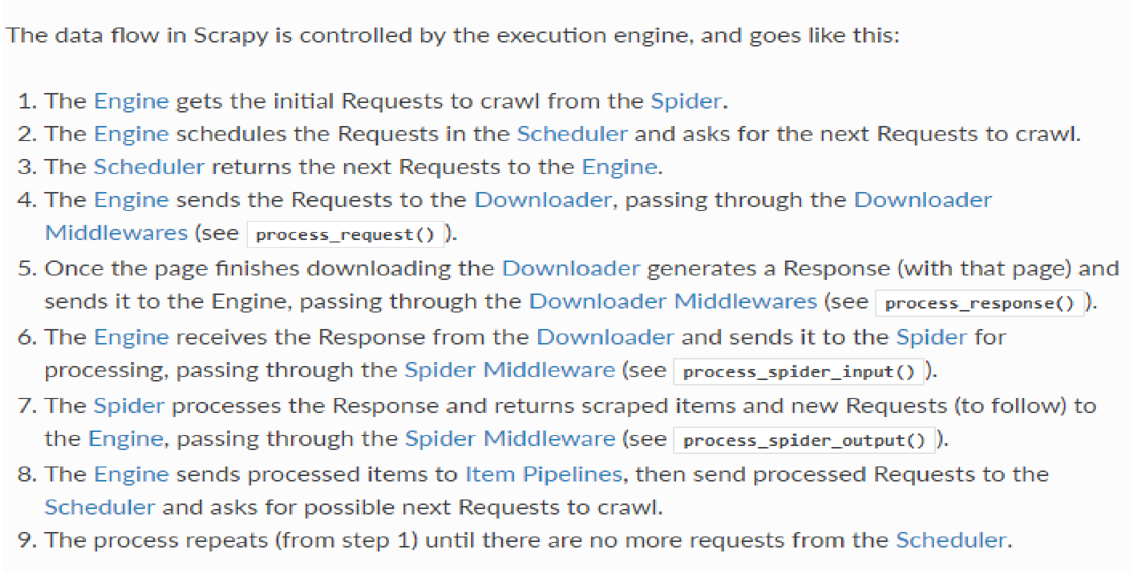
As authentication is a must for a system dealing with highly sensitive information, i.e. human trafficking, the application services can only be accessed after authentication takes place.

Within the application services layer, there are two types of services; services writing to the database, and services reading from the database.

The services writing to the database are those that interact with the spider, as it scrapes a user specified website and stores any findings in the database. The services reading from the database are those that generate reports from the stored findings of the spider. This includes a generic report with all information displayed, and custom reports generated by a search query.

The spider itself has a documented architecture, as scraping websites has a set of required steps for getting and parsing the data found online. The architecture of a Scrapy spider along with a description of the data flow is shown in figure 3.3.




The data flow in Scrapy is controlled by the execution engine, and goes like this:

1. The Engine gets the initial Requests to crawl from the Spider.
2. The Engine schedules the Requests in the Scheduler and asks for the next Requests to crawl.
3. The Scheduler returns the next Requests to the Engine.
4. The Engine sends the Requests to the Downloader, passing through the Downloader Middlewares (see `process_request()`).
5. Once the page finishes downloading the Downloader generates a Response (with that page) and sends it to the Engine, passing through the Downloader Middlewares (see `process_response()`).
6. The Engine receives the Response from the Downloader and sends it to the Spider for processing, passing through the Spider Middleware (see `process_spider_input()`).
7. The Spider processes the Response and returns scraped items and new Requests (to follow) to the Engine, passing through the Spider Middleware (see `process_spider_output()`).
8. The Engine sends processed items to Item Pipelines, then send processed Requests to the Scheduler and asks for possible next Requests to crawl.
9. The process repeats (from step 1) until there are no more requests from the Scheduler.

Figure 3.3: Architecture of Scrapy spiders (Scrapy, 2017)

3.3 Website Design

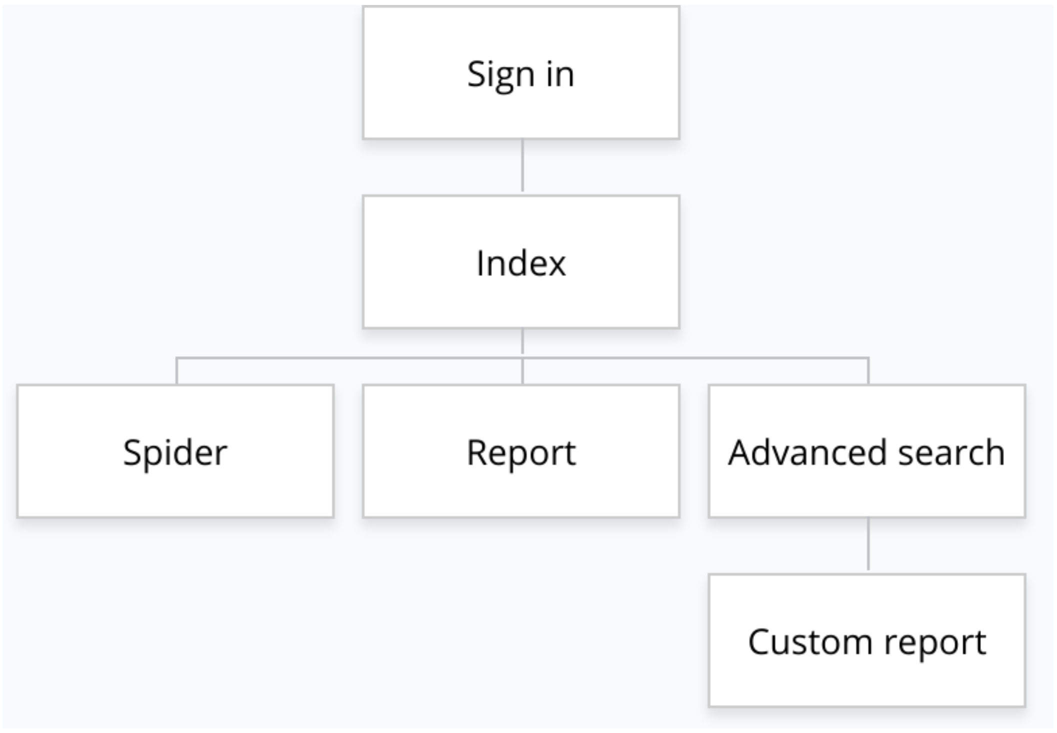


Figure 3.4: Sitemap of the proposed system

Users must first sign in to the application before use, as the information this system deals with is sensitive. Once the user has signed in, they are taken to the index, or main, page, where there is a field to enter the website the user wishes to scrape and a navigation bar at the top of the page, which allows users to access the other pages in the application. If the user chooses to scrape a website, the spider is initialised and the application prompts the user that the target website is being scraped. Once this is finished, a report is generated, showing any possible occurrence of human trafficking related messages. The user may also query the database and generate a custom report using the advanced search feature. This allows the user to search for specific terms, such as a particular buzzword, an author, a target website, or the date of capture.

Shown in figures 3.5 to 3.10 are screenshots of the web application, tentatively named "Spyder". Figure 3.5 shows the sign in page with text fields for the user to enter a username and password. Figure 3.6 shows the index, or main, page, which has a field to enter the target website to scrape and a submit button to launch the spider. The web application displays a message as the spider scrapes this website, shown in figure 3.7. Shown in figure 3.8 is the general report that is generated after a spider is finished scraping and by generating a report without searching. Figures 3.9 and 3.10 show the advanced search and the report generated from this search respectively. The user can search for a specific buzzword, author, date, or website, and the corresponding report for this query will be generated.

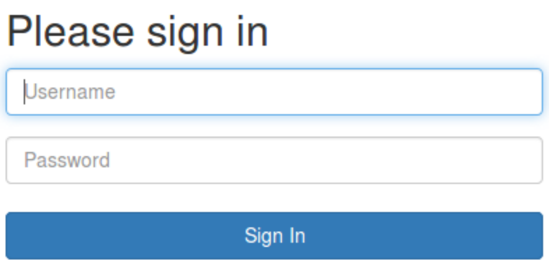


Figure 3.5: Sign in page

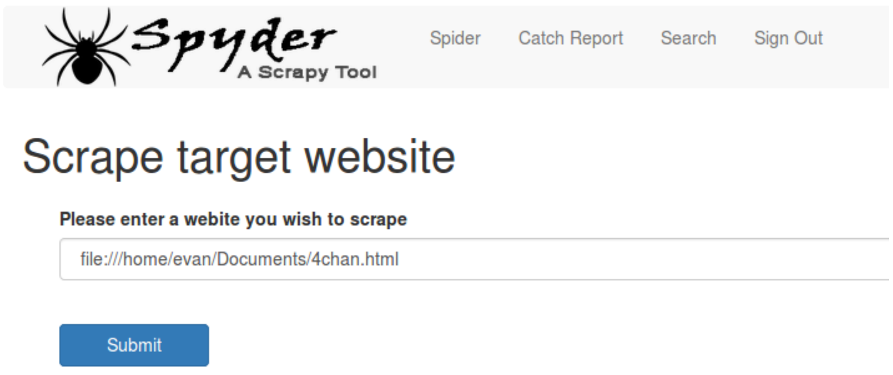

Spyder
A Scrapy Tool
Spider
Catch Report
Search
Sign Out
Scrape target website
Please enter a webite you wish to scrape
file:///home/evan/Documents/4chan.html
Submit

Figure 3.6: Index page

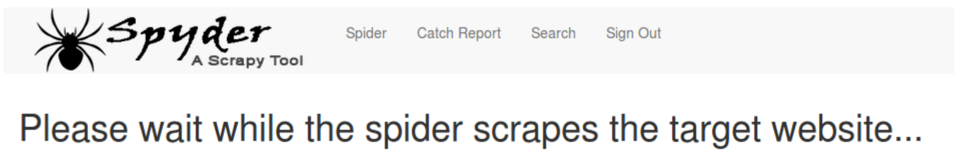
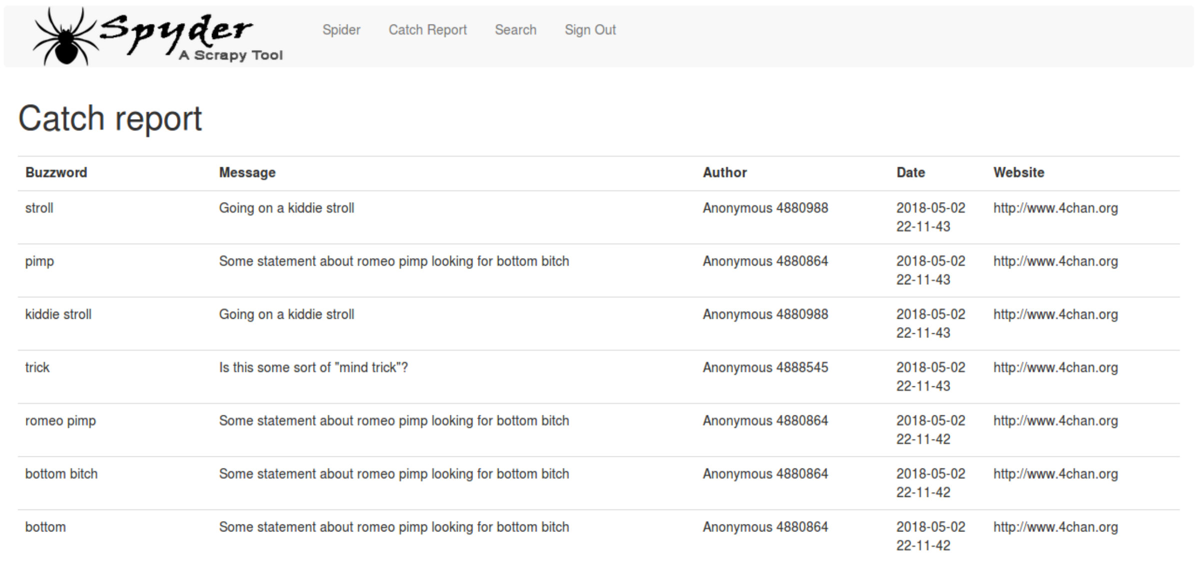


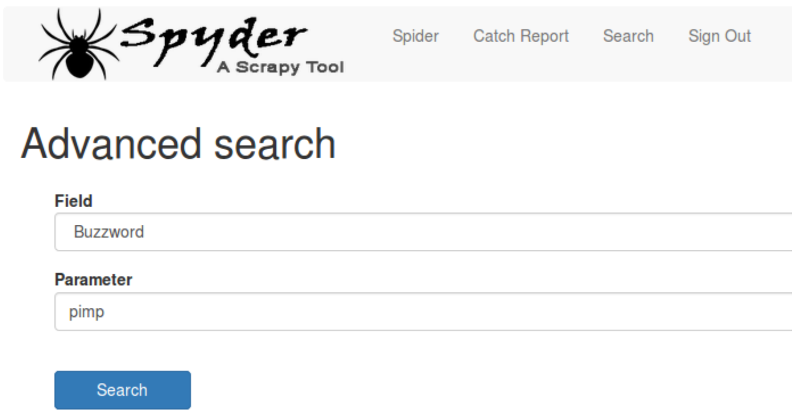
Figure 3.7: Message displayed when the spider scrapes a website



Catch report

Buzzword	Message	Author	Date	Website
stroll	Going on a kiddie stroll	Anonymous 4880988	2018-05-02 22-11-43	http://www.4chan.org
pimp	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-43	http://www.4chan.org
kiddie stroll	Going on a kiddie stroll	Anonymous 4880988	2018-05-02 22-11-43	http://www.4chan.org
trick	Is this some sort of "mind trick"?	Anonymous 4888545	2018-05-02 22-11-43	http://www.4chan.org
romeo pimp	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-42	http://www.4chan.org
bottom bitch	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-42	http://www.4chan.org
bottom	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-42	http://www.4chan.org

Figure 3.8: General report

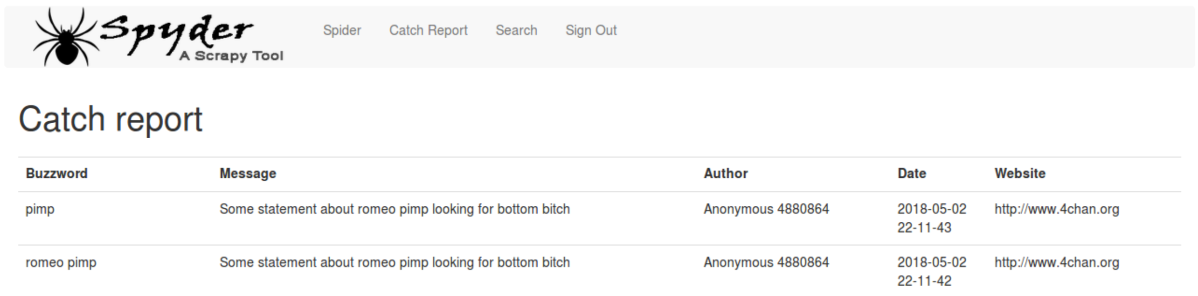


Advanced search

Field: Buzzword

Parameter: pimp

Figure 3.9: Advanced search page



Catch report

Buzzword	Message	Author	Date	Website
pimp	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-43	http://www.4chan.org
romeo pimp	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-42	http://www.4chan.org

Figure 3.10: Custom report

3.4 Database Schema

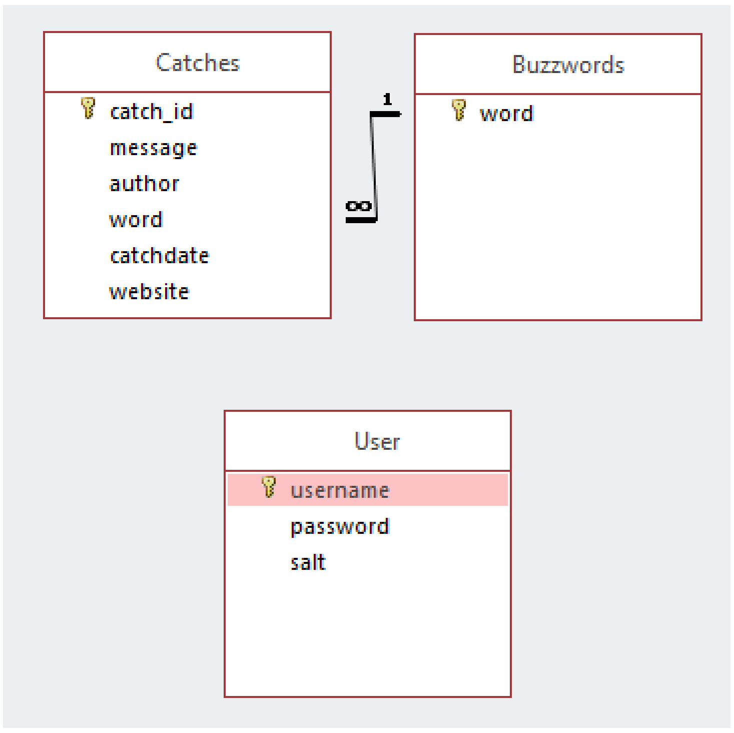


Figure 3.11: Database schema of the proposed system

Shown in figure 3.11 is the database schema of the proposed system. In the database, there are three tables: Buzzwords, Catches, and User. These tables are enough to meet all the requirements of the application.

Buzzwords contains each of the buzzwords being used by the spider to compare to the messages within a website to see if there is a match.

Catches contains the possible occurrences of human trafficking-related messages the spider finds as it crawls through a website. The catch_id field uniquely identifies the possible occurrence. The message and the author of said message is stored, so that a human may assess the possible occurrence of human trafficking by viewing the message using database queries. The buzzword is stored as this would assist in assessing the severity of the stored message by showing what word or words the spider flagged up. For archival purposes, the date, time and website are stored, allowing the user to search for any possible occurrences of human trafficking from a particular website or a certain date.

User contains the username, password and salt corresponding to a user to gain access to the application. The username is stored as text, however, the password is stored as a hashed value made up of the plaintext password and the salt, and is hashed using the SHA-256 hash function. This increases security, allowing the hashed password to be stored safely in the database, and is a safeguard against any attackers attempting to gain access to the application. The salt is a randomly generated string to assist in the hashing of the password. This is stored in the database because when a user attempts to sign in, the user-entered password is combined with the salt, hashed with the same SHA-256 hash function and compared to the stored password hash.

This database schema is in third normal form (3NF), as there are no repeating groups, and each element in each table is dependent on the primary key of that table.

3.5 Class Specification

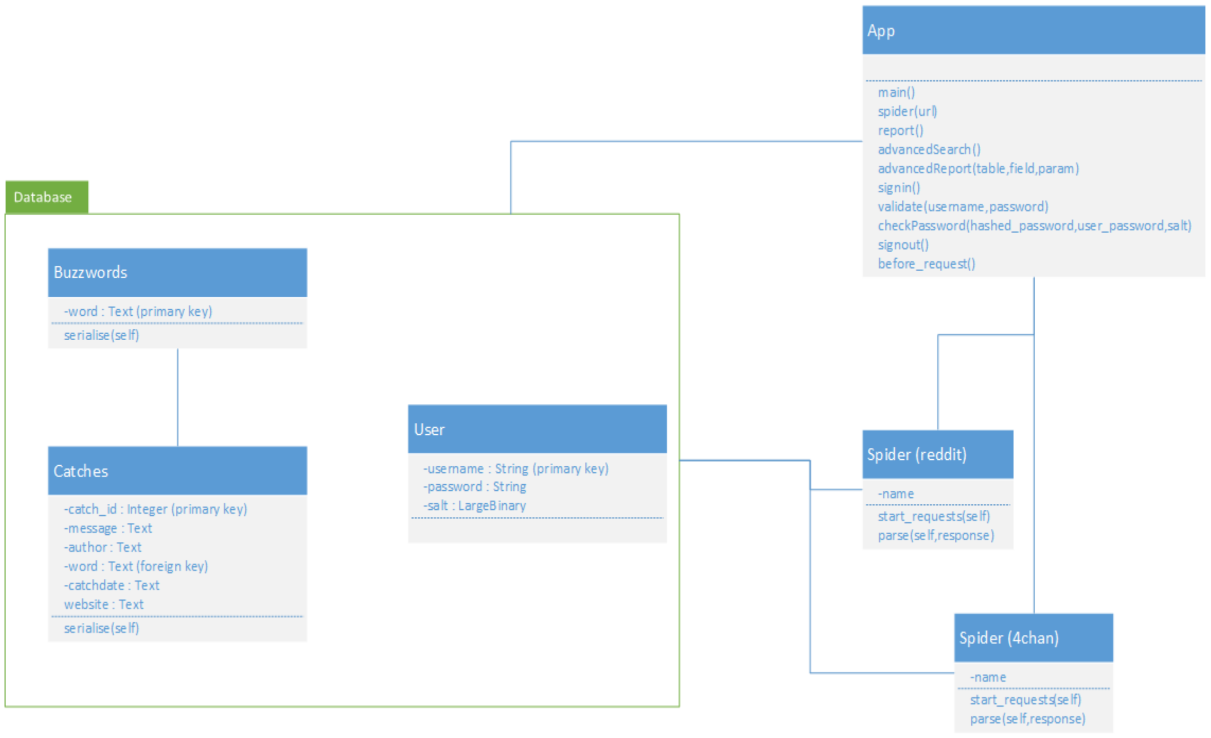


Figure 3.12: Class diagram of the proposed system

The class diagram for the proposed system is shown in figure 3.12. While the database schema, and therefore variables, has already been identified, the Catches and Buzzwords classes have a serialize(self) method. This method allows the data to be serialized, allowing the data to be accessed and displayed by the web application.

The App class deals with the running and directing of the web application. The main() method relates to the index, or main, page and deals with routing to the index.html page, and redirecting to the spider(url) method when a Universe Resource Locator (URL) to scrape is submitted by the user. This spider(url) method uses the URL parameter to run the corresponding spider and display a message to the user, telling them that the spider is running. The report() method renders the report.html page with a query to display on the page. This query fetches all entries in the Catches table of the database. The advancedSearch() method renders the advanced search to allow users to generate a query to display as a report in the advancedReport(table,field,param) method. This method reuses the report.html page, passing in the user-specified search query and generating the corresponding report. The signin() method allows users to sign in to the application and, if successful, creates a session for the authenticated user. When the user attempts to sign in, the validate(username,password) method is called, which checks to see if the credentials are valid and correspond to those in the database. While doing this, the checkPassword(hashed_password,user_password,salt) method is called. This method uses the same SHA-256 hash function and salt to compare the hash, or digest, of the user-entered password with the hashed password stored in the database. The signout() method deletes the user session, signing the user out. The before_request() method is a Flask method that is called before a page is accessed to ensure that a user session is active.

In this class diagram, there are two spiders; one for reddit.com, and another for 4chan.org. This is due to the way Scrapy works, as the class names of HTML elements are generally unique to a website, meaning that a spider will work for one website and not for another. Due to this, multiple spiders are required for a web scraper dealing with multiple websites, however the structure of each spider remains the same. The name variable exists to give the spider a name so that it can be accessed using this name from the command line or application. The start_requests(self) method gets the URL to scrape and runs the request. The parse(self,response) method finds the required information located in the desired website and compares each message to the buzzwords stored in the database. If a buzzword is found in a message, the message, the author, the buzzword, the date and time of capture, and the website URL are stored in the Catches table of the database.

4 Implementation and Testing

4.1 Development Approach

The approach taken during the development of the web application was an incremental one. In this approach each segment was completed before moving on to another. The first segment was building the first spider, focusing on a local page taken from reddit.com, where an understanding of how Scrapy works was gained. After this, the database was developed, and the spider was adapted to write its findings to the database. Once this was completed, work began on the web application, where Flask was set up and the first webpages were made; index.html and layout.html. The layout page is a page that never loads by itself, as it provides the layout of the parts of the webpages that do not change between pages. An example of this is the menu, or navbar, at the top of each page, which remains constant no matter which page the application loads. After this, the spider(url) method was created to allow the spider to be activated from within the web application. The report.html page was then created to display the findings of the spider in an easy-to-read fashion. A second spider was then created, this time targeted towards a local page taken from 4chan.org. Lastly, the advanced search was developed to allow a user to send queries to the database and generate custom reports.

The tools and languages used to develop this system were as follows:

- Backend: Python and its libraries; Scrapy, Flask and SQLAlchemy
- Frontend: HTML, CSS, jQuery, and Twitter Bootstrap

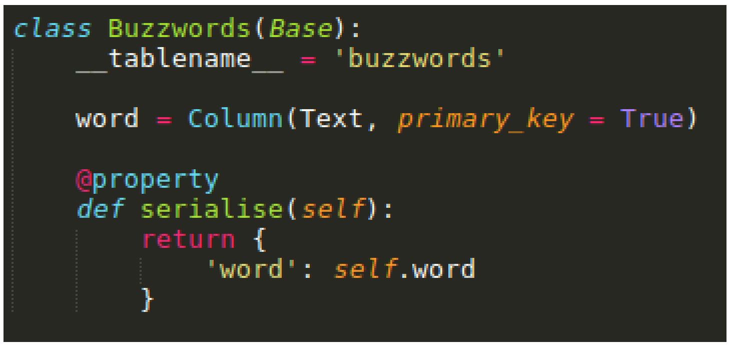
These technologies were chosen for two reasons. The first is due to already having experience in using these languages and libraries for developing web applications, with the exception of the Scrapy library, and the second is that Scrapy is a Python library, so the application also had to be built using Python.

As Scrapy was the only unfamiliar tool, time was spent learning how to use the framework at the beginning of development before any work began on the other, more familiar components.

4.2 Implementation

4.2.1 Database

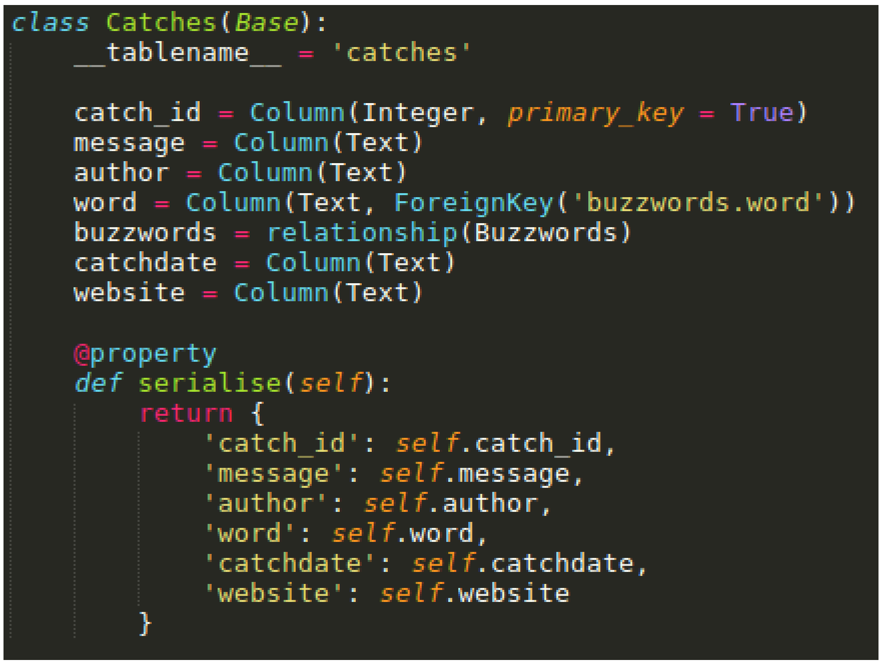
The database, called scrape.db, was developed using SQLAlchemy, an object-relational mapper, ORM, in Python. This uses classes to create database tables, as shown with the Buzzwords class below.



```python
class Buzzwords(Base):
    __tablename__ = 'buzzwords'

    word = Column(Text, primary_key = True)

    @property
    def serialise(self):
        return {
            'word': self.word
        }
```

This creates a table in the database called "buzzwords", with a "word" column being the primary key and only field. The serialise(self) method allows the data to be accessed and displayed by the web application.

The Catches class creates a table in the database containing a catch_id as the primary key, a scraped message, the author, the buzzword as a foreign key to the Buzzwords class, or table, the catchdate, and the website.



```python
class Catches(Base):
    __tablename__ = 'catches'

    catch_id = Column(Integer, primary_key = True)
    message = Column(Text)
    author = Column(Text)
    word = Column(Text, ForeignKey('buzzwords.word'))
    buzzwords = relationship(Buzzwords)
    catchdate = Column(Text)
    website = Column(Text)

    @property
    def serialise(self):
        return {
            'catch_id': self.catch_id,
            'message': self.message,
            'author': self.author,
            'word': self.word,
            'catchdate': self.catchdate,
            'website': self.website
        }
```

The catchdate field is stored as a Text field rather than a DateTime because a format of “YYYY-MM-DD hh-mm-ss” was desired and DateTime fields can cause issues when attempting to display the field in a web application.

The User class creates a table with a username, password and salt. This class does not require a serialise method, as user information will not be displayed at any point in the application for security concerns.

```
class User(Base):
    __tablename__ = 'user'

    username = Column(String(20), primary_key = True)
    password = Column(String(20), nullable = False)
    salt = Column(LargeBinary)
```

The database is populated with buzzwords prior to use to ensure that the spiders can function as intended.

```
words = ['automatic', 'bottom', 'bottom bitch', 'branding', 'caught a case', 'choosing up', '
    circuit', 'coercion', 'commercial sex act', 'cousin-in-law', 'cousin in law', 'daddy', '
    date', 'exit fee', 'facilitaor', 'family', 'folks', 'finesse pimp', 'romeo pimp', 'force'
    , 'fraud', 'gorilla pimp', 'guerilla pimp', 'head cut', 'human smuggling', 'human
    traffick', 'in-pocket', 'in pocket', 'john', 'trick', 'kiddie stroll', 'loose bitch', '
    lot lizard', 'madam', 'out of pocket', 'pimp', 'pimp circle', 'pimp partner', 'quota', '
    eyeballing', 'renegade', 'seasoning', 'serving a pimp', 'squaring up', 'stable', 'the
    game', 'the life', 'track', 'stroll', 'blade', 'trade up', 'trade down', 'traficker', '
    turn out', 'the wire', 'wifey', 'wife-in-law', 'wife in law', 'sister wife']

for word in words:
    buzz = Buzzwords(word=word)
    session.add(buzz)
    session.commit()
```

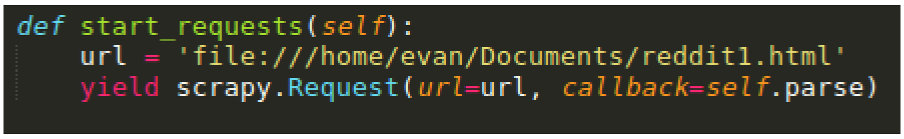
A list of words relating to sex trafficking is created. This list is then looped through and each individual word, or phrase, is added to the Buzzwords table.

A user is then created with the username “admin” and the password “admin123”. A random 8-byte string is create for use as a salt. The password is then combined with the salt and they are hashed using the SHA-256 hash function. This is then stored as the password for the user and the record is added to the database.

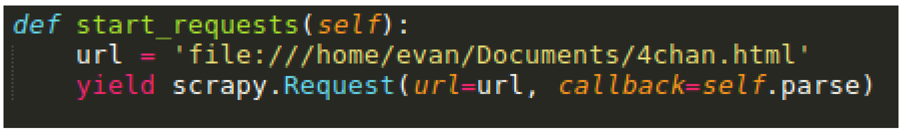
```
password = b"admin123"
salt = urandom(8)
hash_object = hashlib.sha256(password.encode() + salt)
admin = User(username=u"admin", password=hash_object.hexdigest(), salt=salt)
session.add(admin)
session.commit()
```

4.2.2 Spider

The spiders for this application were developed using Scrapy, a Python framework for scraping websites. Within the Spider class, the start_requests(self) method is automatically called. This method begins the request by passing a URL and calling the parse(self,response) method.

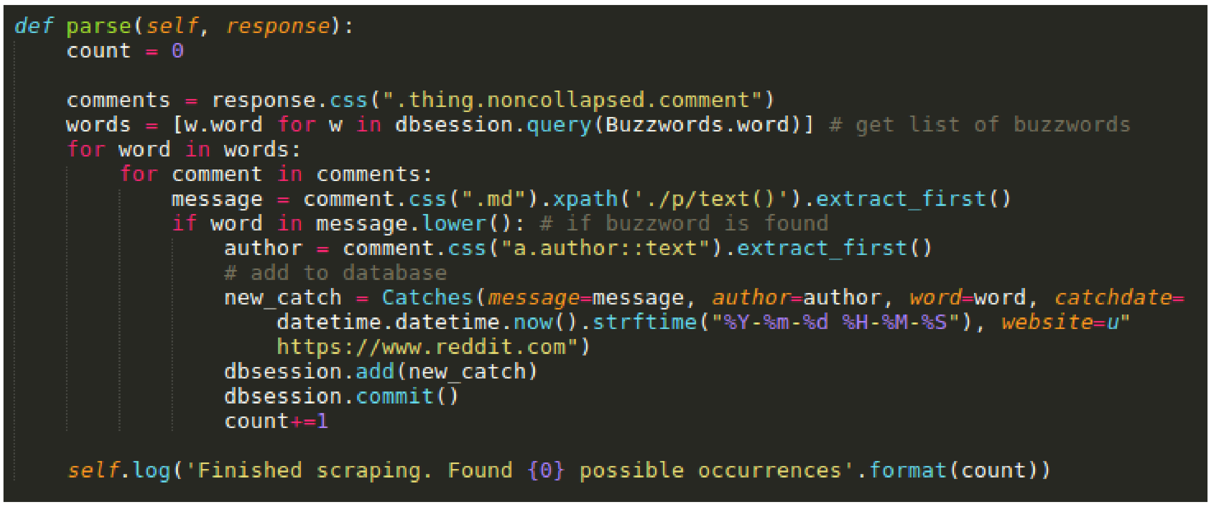


```
def start_requests(self):
    url = 'file:///home/evan/Documents/reddit1.html'
    yield scrapy.Request(url=url, callback=self.parse)
```

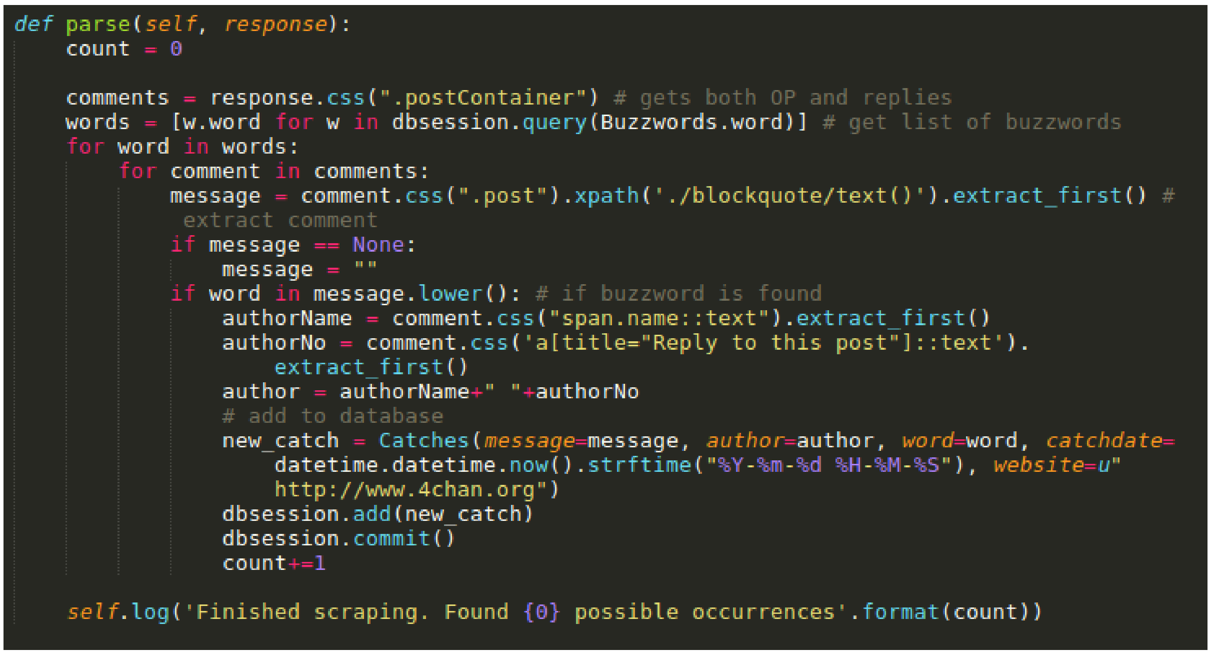


```
def start_requests(self):
    url = 'file:///home/evan/Documents/4chan.html'
    yield scrapy.Request(url=url, callback=self.parse)
```

The parse(self,response) method uses the class names of div elements in a webpage to find the comments people have sent. A list of buzzwords is created using the buzzwords stored in the database. These words are then looped through and a nested loop is made to go through each comment, extract the message and compare to the buzzword. If the buzzword is found in the message, it is stored in the database along with the author of that comment, the buzzword, the date and time of capture, and the website. In the 4chan spider, the author number (post ID) is also extracted, as most of the authors of comments are anonymous, so this number can uniquely identify the author of a comment.



```
def parse(self, response):
    count = 0

    comments = response.css(".thing.noncollapsed.comment")
    words = [w.word for w in dbsession.query(Buzzwords.word)] # get list of buzzwords
    for word in words:
        for comment in comments:
            message = comment.css(".md").xpath('./p/text()').extract_first()
            if word in message.lower(): # if buzzword is found
                author = comment.css("a.author::text").extract_first()
                # add to database
                new_catch = Catches(message=message, author=author, word=word, catchdate=
                    datetime.datetime.now().strftime("%Y-%m-%d %H-%M-%S"), website=u"
                    https://www.reddit.com")
                dbsession.add(new_catch)
                dbsession.commit()
                count+=1

    self.log('Finished scraping. Found {0} possible occurrences'.format(count))
```

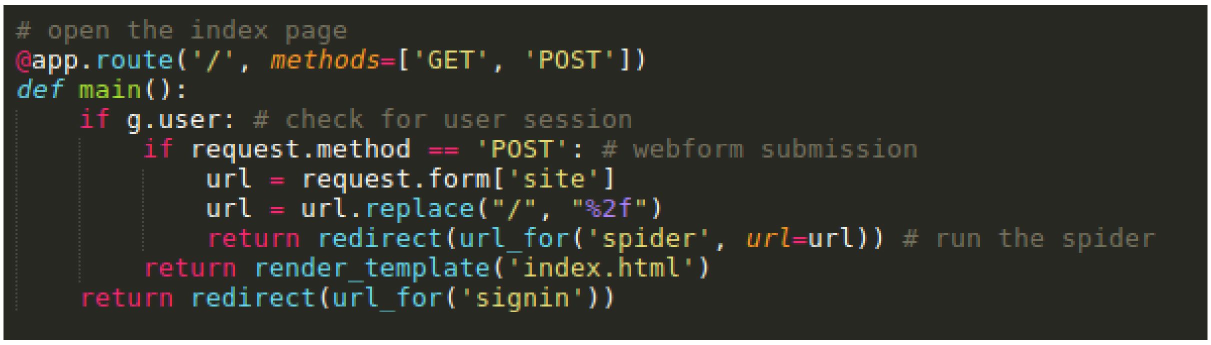


```
def parse(self, response):
    count = 0

    comments = response.css(".postContainer") # gets both OP and replies
    words = [w.word for w in dbsession.query(Buzzwords.word)] # get list of buzzwords
    for word in words:
        for comment in comments:
            message = comment.css(".post").xpath('./blockquote/text()').extract_first() #
             extract comment
            if message == None:
                message = ""
            if word in message.lower(): # if buzzword is found
                authorName = comment.css("span.name::text").extract_first()
                authorNo = comment.css('a[title="Reply to this post"]::text').
                    extract_first()
                author = authorName+" "+authorNo
                # add to database
                new_catch = Catches(message=message, author=author, word=word, catchdate=
                    datetime.datetime.now().strftime("%Y-%m-%d %H-%M-%S"), website=u"
                    http://www.4chan.org")
                dbsession.add(new_catch)
                dbsession.commit()
                count+=1

    self.log('Finished scraping. Found {0} possible occurrences'.format(count))
```

4.2.3 Web Application

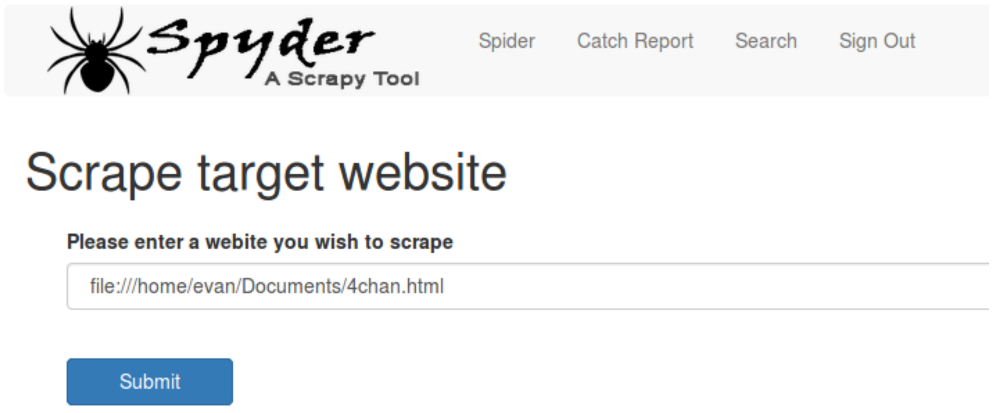
The web application was developed using the Flask framework for Python in the backend, and HTML pages in the frontend. For all HTML pages, see Appendix B.

The main() method renders the index.html page if the user is signed in. The route is the path in the URL bar of a browser. For example, to reach the index page, “localhost:5000/” is entered, and to reach the sign in page, “localhost:5000/signin/” is entered.



```
# open the index page
@app.route('/', methods=['GET', 'POST'])
def main():
    if g.user: # check for user session
        if request.method == 'POST': # webform submission
            url = request.form['site']
            url = url.replace("/", "%2f")
            return redirect(url_for('spider', url=url)) # run the spider
        return render_template('index.html')
    return redirect(url_for('signin'))
```

This method checks to see if a user is signed in by looking for an active session. If the user submits the form on the index.html page, the entered website is sent to the spider to be scraped. Shown below is the index.html page.

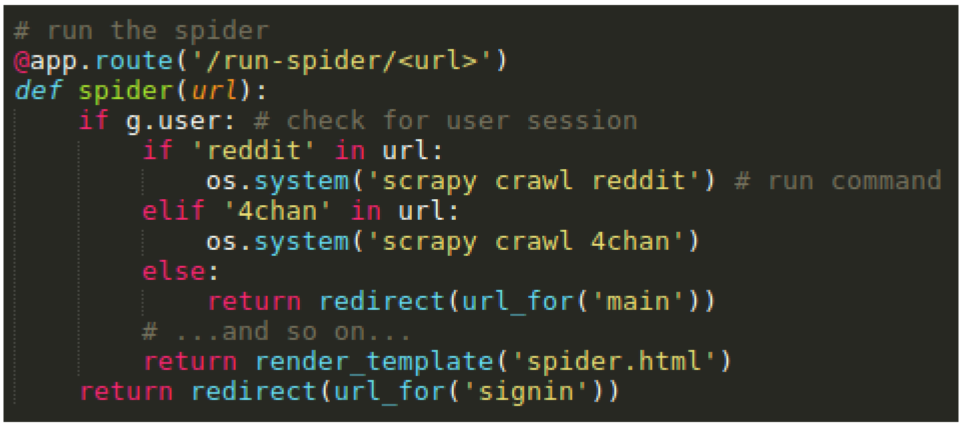


Scrape target website

Please enter a webite you wish to scrape

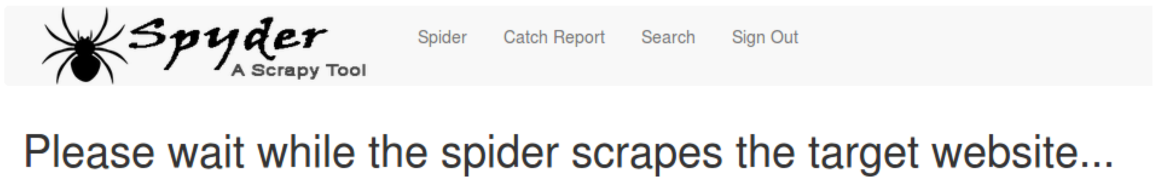
file:///home/evan/Documents/4chan.html

The spider(url) method handles the execution of the spiders. Due to the limitations with Scrapy spiders discussed previously, separate spiders are used, therefore, there are checks to see what URL was entered and the corresponding spider is executed.



```
# run the spider
@app.route('/run-spider/<url>')
def spider(url):
    if g.user: # check for user session
        if 'reddit' in url:
            os.system('scrapy crawl reddit') # run command
        elif '4chan' in url:
            os.system('scrapy crawl 4chan')
        else:
            return redirect(url_for('main'))
        # ...and so on...
        return render_template('spider.html')
    return redirect(url_for('signin'))
```

Shown below is the spider.html page displaying a message to the user, telling them that the spider is active.

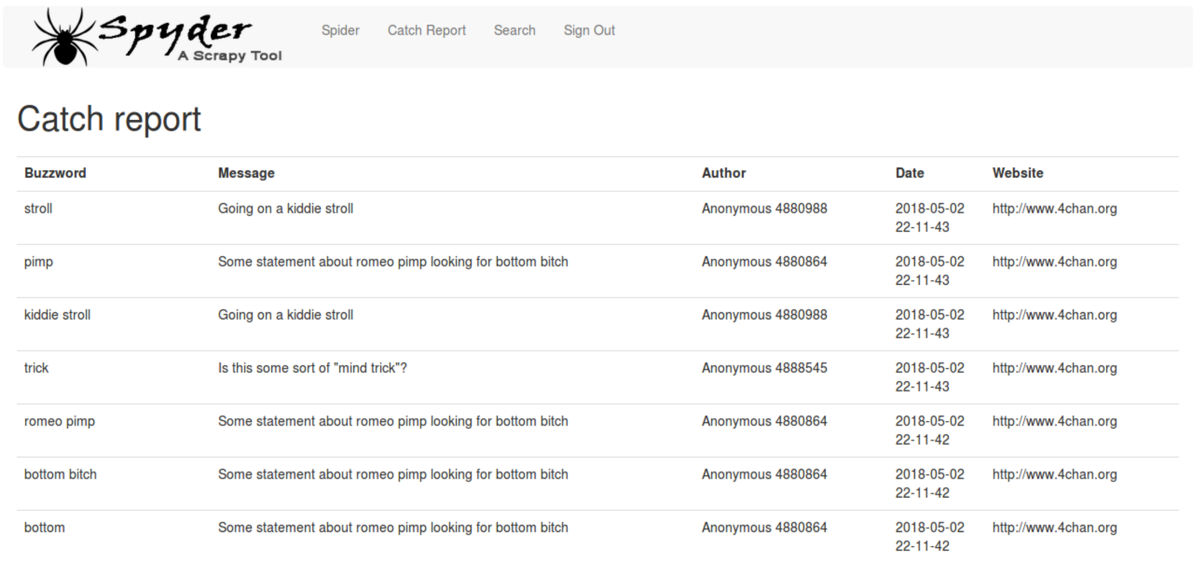


Please wait while the spider scrapes the target website...

The report() method returns all Catches and displays them in a table in the report.html page.

```
# display report
@app.route('/report/')
def report():
    if g.user: # check for user session
        query = dbsession.query(Catches)
        return render_template('report.html', query=([catch.serialise for catch in query]))
    return redirect(url_for('signin'))
```

Shown below is the general report generated in report.html by this method.



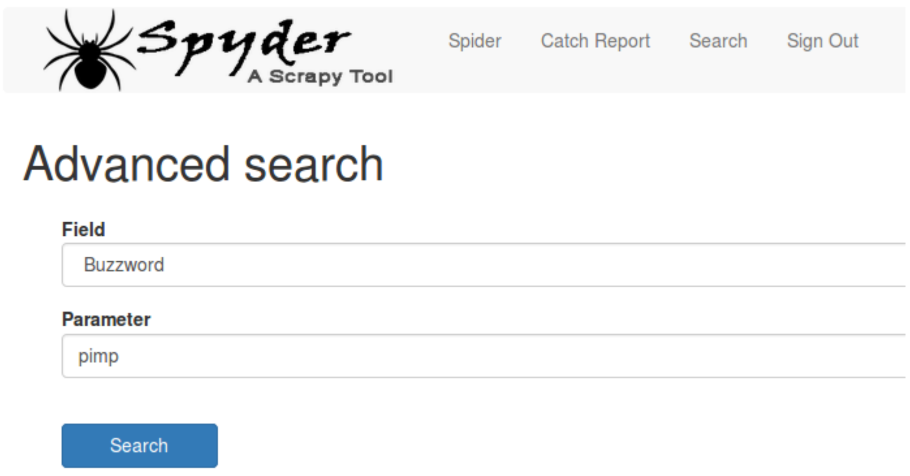
Catch report

Buzzword	Message	Author	Date	Website
stroll	Going on a kiddie stroll	Anonymous 4880988	2018-05-02 22-11-43	http://www.4chan.org
pimp	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-43	http://www.4chan.org
kiddie stroll	Going on a kiddie stroll	Anonymous 4880988	2018-05-02 22-11-43	http://www.4chan.org
trick	Is this some sort of "mind trick"?	Anonymous 4888545	2018-05-02 22-11-43	http://www.4chan.org
romeo pimp	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-42	http://www.4chan.org
bottom bitch	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-42	http://www.4chan.org
bottom	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-42	http://www.4chan.org

The advancedSearch() method allows users to search the database for specific terms. Due to this, the method gets the values in the webform in the advanced.html page, and passes them as parameters into the advanceReport(table,field,param) method.

```
# generate dynamic query
@app.route('/report/search/', methods=['GET', 'POST'])
def advancedSearch():
    if g.user: # check for user session
        if request.method == 'POST':
            if request.form['field']:
                field = request.form['field']
            if request.form['param']:
                param = request.form['param']
                param = param.replace("/", "%2f")
            elif not request.form['param']:
                param = "%"
            return redirect(url_for('advancedReport', table="catches", field=field, param=
                param))
        return render_template('advanced.html')
    return redirect(url_for('signin'))
```

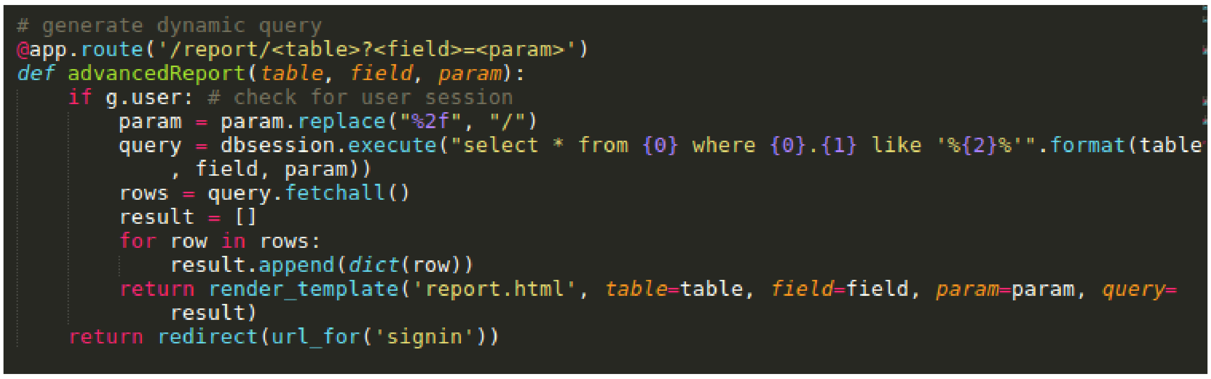
Shown below is the advanced.html page with fields allowing the user to enter data to create custom reports.



Advanced search

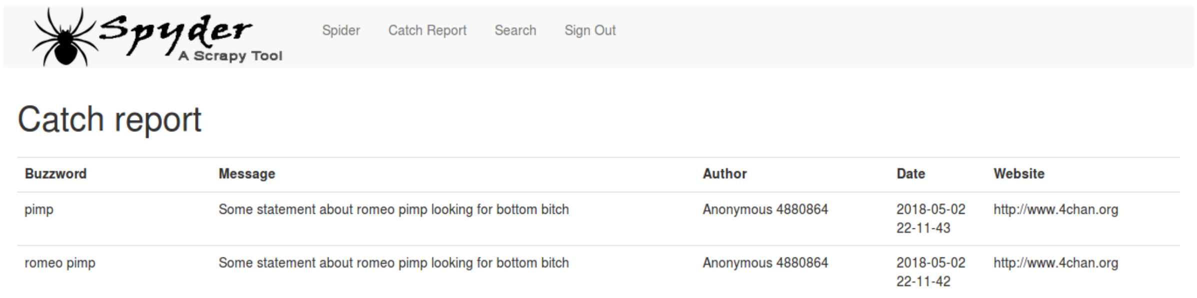
Field	Buzzword
Parameter	pimp

The advancedReport(table,field,param) method uses the parameters received from the advancedSearch() method to generate a custom query. The results of this query are then added to a dictionary, which is then passed into the report.html page for display.



```
# generate dynamic query
@app.route('/report/<table>?<field>=<param>')
def advancedReport(table, field, param):
    if g.user: # check for user session
        param = param.replace("%2f", "/")
        query = dbsession.execute("select * from {0} where {0}.{1} like '%{2}%'".format(table
            , field, param))
        rows = query.fetchall()
        result = []
        for row in rows:
            result.append(dict(row))
        return render_template('report.html', table=table, field=field, param=param, query=
            result)
    return redirect(url_for('signin'))
```

Shown below is the custom report generated in report.html by this method.



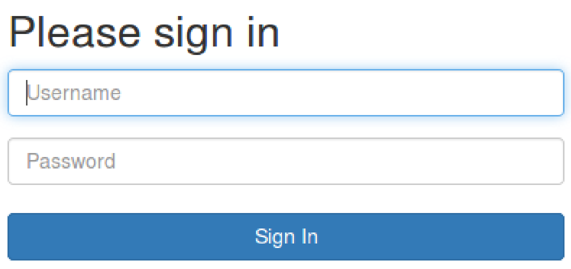
Catch report

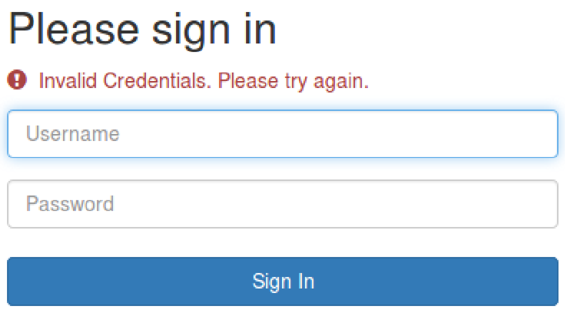
Buzzword	Message	Author	Date	Website
pimp	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-43	http://www.4chan.org
romeo pimp	Some statement about romeo pimp looking for bottom bitch	Anonymous 4880864	2018-05-02 22-11-42	http://www.4chan.org

The signin() method deals with the authentication of a user. The entered username and password is checked for validity in the validate(username,password) method by comparing the entered username to the username stored in the database. To check the password, the check_password(hashed_password,user_password,salt) method is called, which uses a SHA-256 hash function to generate a digest and compare it with the password digest in the database. If the check is unsuccessful, an error message is passed into the signin.html page to be displayed.

```
# compare password with database
def check_password(hashed_password, user_password, salt):
    return hashed_password == hashlib.sha256((user_password.encode()) + salt).hexdigest()

# compare credentials with database
def validate(username, password):
    completion = False
    users = dbsession.query(User)
    for user in users:
        if user.username == username:
            completion = check_password(user.password, password, user.salt)
    return completion

# sign in user
@app.route('/signin/', methods=['GET', 'POST'])
def signin():
    error = None
    if request.method == 'POST':
        session.pop('user', None)
        uname = request.form['username']
        pword = request.form['password']
        completion = validate(uname, pword)
        if completion == True:
            session['user'] = uname # create user session
            return redirect(url_for('main'))
        else:
            error = 'Invalid Credentials. Please try again.'
    return render_template('signin.html', error=error)
```

Shown below is the signin.html page, first with no error message, then with the error message displayed during a failed sign in attempt.



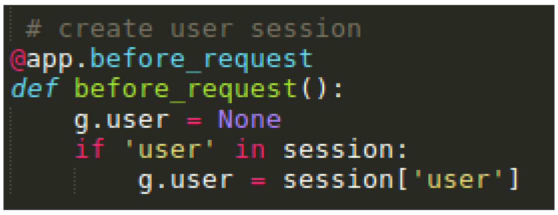


Invalid Credentials. Please try again.

The signout() method deletes the active user session, signing the user out of the application.

```
# delete session
@app.route('/signout/')
def signout():
    session.pop('user', None)
    return redirect(url_for('main'))
```

The before_request() method is a Flask method that is called before each request and checks to see if there is an active session.

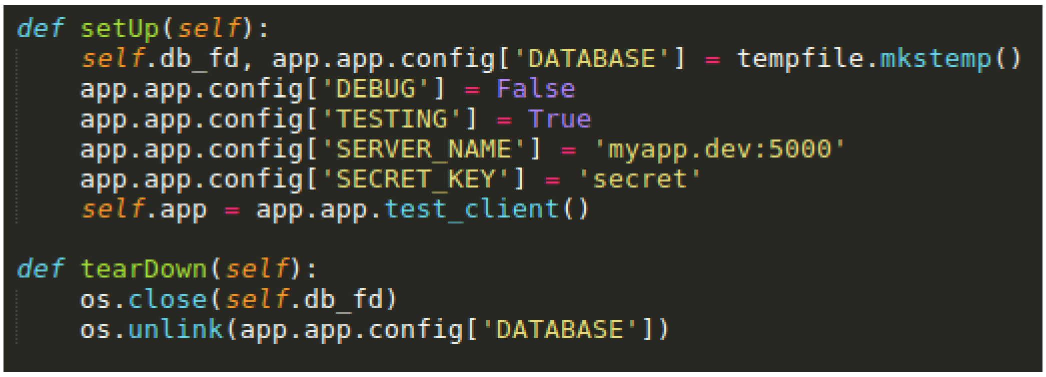


```
# create user session
@app.before_request
def before_request():
    g.user = None
    if 'user' in session:
        g.user = session['user']
```

4.3 Testing

For testing this web application, unit testing was carried out using Python's included unit testing framework.

To begin testing a Flask web application, a setup(self) method and a teardown(self) must be created. The setup(self) method creates the necessary connections to the database and sets configurations for use in automated testing. The teardown(self) method closes these connections and unlinks from the database.

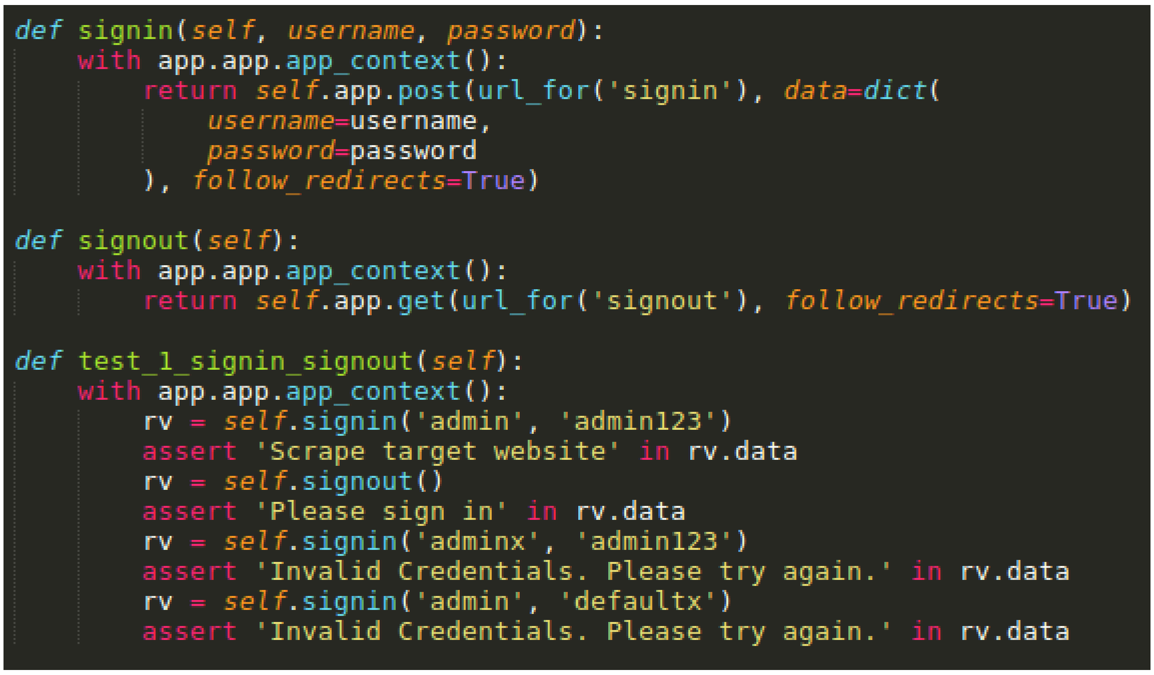


```
def setUp(self):
    self.db_fd, app.app.config['DATABASE'] = tempfile.mkstemp()
    app.app.config['DEBUG'] = False
    app.app.config['TESTING'] = True
    app.app.config['SERVER_NAME'] = 'myapp.dev:5000'
    app.app.config['SECRET_KEY'] = 'secret'
    self.app = app.app.test_client()

def tearDown(self):
    os.close(self.db_fd)
    os.unlink(app.app.config['DATABASE'])
```

The first test focuses on user authentication. Three sets of usernames and passwords are tested:

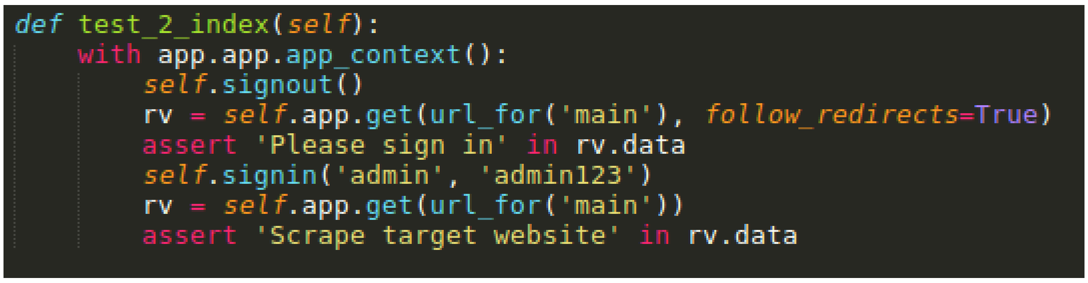
- admin, admin123 (valid, valid)
- adminx, admin123 (invalid, valid)
- admin, default (valid, invalid)

These credentials are then passed into the signin() method to check for validity. The rv.data calls look for text in the rendered webpages.



```python
def signin(self, username, password):
    with app.app.app_context():
        return self.app.post(url_for('signin'), data=dict(
            username=username,
            password=password
        ), follow_redirects=True)

def signout(self):
    with app.app.app_context():
        return self.app.get(url_for('signout'), follow_redirects=True)

def test_1_signin_signout(self):
    with app.app.app_context():
        rv = self.signin('admin', 'admin123')
        assert 'Scrape target website' in rv.data
        rv = self.signout()
        assert 'Please sign in' in rv.data
        rv = self.signin('adminx', 'admin123')
        assert 'Invalid Credentials. Please try again.' in rv.data
        rv = self.signin('admin', 'defaultx')
        assert 'Invalid Credentials. Please try again.' in rv.data
```

The second test signs the user out and attempts to access the main() method, asserting that the signin.html page should open instead. The test then signs the user in with valid credentials and attempts to access the main() method, asserting that text from the index.html page should be returned.

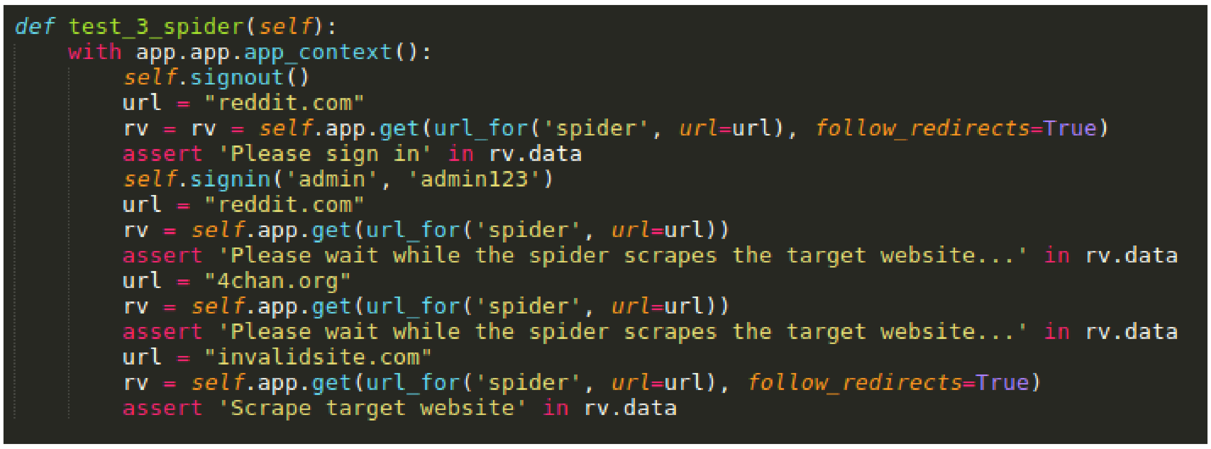


```python
def test_2_index(self):
    with app.app.app_context():
        self.signout()
        rv = self.app.get(url_for('main'), follow_redirects=True)
        assert 'Please sign in' in rv.data
        self.signin('admin', 'admin123')
        rv = self.app.get(url_for('main'))
        assert 'Scrape target website' in rv.data
```

The third test signs the user out and attempts to access the spider, asserting that the signin.html page should open instead. The test then signs the user in with valid credentials and passes three URLs into the spider(url) method:

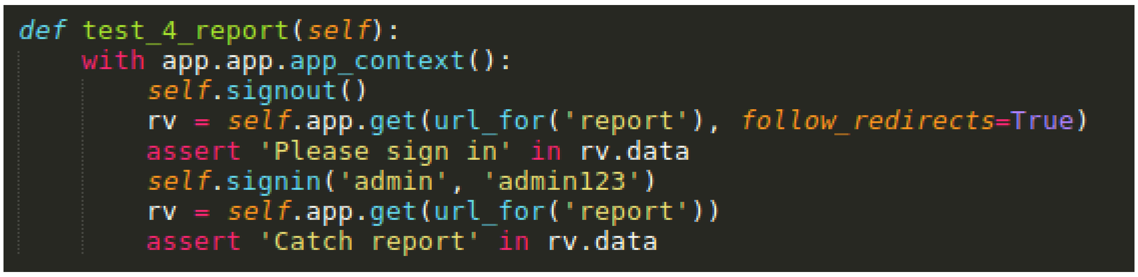
- reddit.com (valid)
- 4chan.org (valid)
- invalidsite.com (invalid)

The first two should execute the corresponding spiders, while the third should not execute any spider and should redirect to the index.html page.



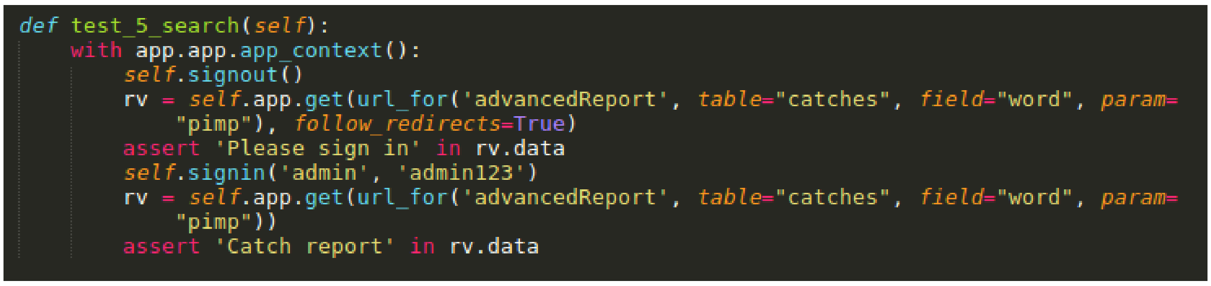
```python
def test_3_spider(self):
    with app.app.app_context():
        self.signout()
        url = "reddit.com"
        rv = rv = self.app.get(url_for('spider', url=url), follow_redirects=True)
        assert 'Please sign in' in rv.data
        self.signin('admin', 'admin123')
        url = "reddit.com"
        rv = self.app.get(url_for('spider', url=url))
        assert 'Please wait while the spider scrapes the target website...' in rv.data
        url = "4chan.org"
        rv = self.app.get(url_for('spider', url=url))
        assert 'Please wait while the spider scrapes the target website...' in rv.data
        url = "invalidsite.com"
        rv = self.app.get(url_for('spider', url=url), follow_redirects=True)
        assert 'Scrape target website' in rv.data
```

The fourth test attempts to access the report() method to generate a report, first while signed out, then while signed in with valid credentials. While signed out, the signin.html page should open, and while signed in, the report.html page should open.



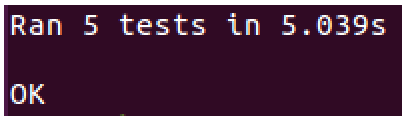
```python
def test_4_report(self):
    with app.app.app_context():
        self.signout()
        rv = self.app.get(url_for('report'), follow_redirects=True)
        assert 'Please sign in' in rv.data
        self.signin('admin', 'admin123')
        rv = self.app.get(url_for('report'))
        assert 'Catch report' in rv.data
```

The fifth test signs the user out and attempts to access the advancedReport(table,field,param) method, and asserts that the signin.html page should open. The test then signs the user in with valid credentials and the report.html page should open.



```python
def test_5_search(self):
    with app.app.app_context():
        self.signout()
        rv = self.app.get(url_for('advancedReport', table="catches", field="word", param=
            "pimp"), follow_redirects=True)
        assert 'Please sign in' in rv.data
        self.signin('admin', 'admin123')
        rv = self.app.get(url_for('advancedReport', table="catches", field="word", param=
            "pimp"))
        assert 'Catch report' in rv.data
```

After running the test script, each test passed in 5.039 seconds with no failed tests.



```
Ran 5 tests in 5.039s

OK
```

These tests show that the developed product works as intended.

5 Evaluation and Reflection

5.1 Evaluation

In chapter 2 of this report, several functional and non-functional requirements were addressed. Reviewing these requirements shows how close the project stayed to the original plan and if it met those requirements. Shown in tables 5.1 and 5.2 are the comparisons with the functional and non-functional requirements respectively.

Table 5.1: Evaluation of functional requirements

Functional Requirement	Requirement Met?	Comment
The system should prompt the user to log in before use for confidentiality, as the data is sensitive.	Yes	Each page of the application checks to see if the user is signed in. if not, the user is redirected to the sign in page.
The system should allow the user to enter a website to scrape.	Yes	The index, or main, page contains a field for the user to enter a target website to scrape.
The system should crawl through and extract data from the desired website.	Yes	The target website is crawled through and information relating to

		human trafficking is extracted.
The system should store data in a database with timestamps.	Yes	The extracted data is stored along with the date and time of capture.
The system should allow users to access and query the database from within the web application.	Yes	Users can perform an advanced search, creating a custom query, which then creates a custom report.

Table 5.2: Evaluation of non-functional requirements

Non-Functional Requirement	Requirement Met?	Comment
The system should be user friendly and easy to navigate.	Yes	The navbar at the top of each page can be used to access all pages within the system.
The system should be easily maintained with minimal maintenance required.	Yes	All components can be modified without shutting down the web application, so there is no downtime.
The system should be scalable and work efficiently under heavy workloads.	Yes	As Flask is a templating engine, more templates or pages can be added very easily and new spiders

		can be created.
The system should be easy to install and set up.	Yes	After installing the dependencies, Python, Flask, Scrapy, etc., the entire system can run from one command; “python app.py”.
The handling of user and scraped data should conform to the Data Protection Act 1998.	Yes	All data is treated as confidential and can only be accessed after login.

Although all test cases passed, and every requirement has been met, the system could still be improved. One such improvement would be to minimise false-positives. The spiders found comments unrelated to human trafficking and extracted them, due to a match with the buzzwords. The matches buzzwords are generic words, such as “stable”, “force, “trick”, “john”, etc. This leads to many false-positives, as the spiders do not rely on the context of the words. This leaves three options for future improvements:

- Remove the generic words
- Let a human read through an assess the findings
- Let an AI (artificial intelligence), that knows context, read through and make decisions on the findings

5.2 Reflection

Over the course of the project, knowledge about web scrapers has been gained, and using this knowledge, a number of things could be done differently if faced with this project again.

Although, at its core, the project did not deviate from the original plan, due to the spiders being specifically targeted to one site each, the choice of websites to target was highly limited. If faced with this problem again, an attempt to create a 'general purpose' spider would be made. This type of spider would need to intelligently find which HTML elements and classes contain the required information. This, however, is beyond the scope of this project.

With more time and added scope to the project, an intelligent system could be created to determine the context of the scraped messages to reduce the number of false-positives in the findings. Due to the sensitive nature of the topic of this project, human trafficking, all scraping had to be performed on local webpages rather than online websites. With the backing of law enforcement agencies, spiders could be made to scrape website on an online domain and be allowed to follow links recursively.

5.3 Conclusions

While this project may not be as sophisticated as web scrapers made by large corporations, there is enough scope in this application to make a significant impact in the world of law enforcement. By utilising a set of buzzwords relating to sex trafficking and a spider targeted towards the right website, many trafficking crimes could be discovered in a matter of seconds, as the spiders in this project crawled through large webpages in under 5 seconds.

6 References

Albert, R., Jeong, H. and Barabási, A. (1999) Internet: Diameter of the world-wide web, *Nature,* 401(6749), pp. 130-131. doi: 10.1038/43601.

Balodis, M. (2017) *Web Scraper.* Available at: http://webscraper.io/ (Accessed: 02/11/17).

Bin, H., Patel, M. and Zhen, Z. (2007) Accessing the Deep Web: a survey, *Communications of the ACM,* 50(5), pp. 94-101.

Boorse, K. (2016) *Spotlight Helps Law Enforcement Identify Victims of Sex Trafficking Faster.* Available at: https://www.wearethorn.org/blog/spotlight-helps-identify-sex-trafficking-victims-faster/ (Accessed: 29/10/17).

Broder, A.Z., Najork, M. and Wiener, J.L. (2003) *Efficient URL caching for world wide web crawling.* , Budapest, Hungary. 20-24 May 2003. New York, NY, USA: ACM, pp. 679.

Castillo, C. (2004) *Effective Web Crawling.* Ph.D. in Computer Science. University of Chile.

Cordua, J. (2017) *Clarity & Focus in 2017.* Available at: https://www.wearethorn.org/blog/clarity-and-focus-2017/ (Accessed: 29/10/17).

Europol (2017) *SERIOUS AND ORGANISED CRIME THREAT ASSESSMENT Crime in the age of technology*Europol. Available at: https://www.europol.europa.eu/sites/default/files/documents/socta2017_0.pdf (Accessed: 02/11/2017).

Google (2017) *Fighting Human Trafficking & Modern Day Slavery.* Available at: https://www.blog.google/documents/4/Fighting%20Human%20Trafficking%20and%20Modern%20Day%20Slavery.pdf (Accessed: 06/11/17).

IC3 (2017) *2016 Internet Crime Report* Available at: https://pdf.ic3.gov/2016_IC3Report.pdf (Accessed: 19/11/2017).

ILO (2012) *ILO 2012 Global estimate of forced labour Executive summary* Available at: http://www.ilo.org/wcmsp5/groups/public/---ed_norm/---declaration/documents/publication/wcms_181953.pdf (Accessed: 02/11/2017).

Import.io (2017) *Import.io | Extract data from the web.* Available at: https://www.import.io/ (Accessed: 06/11/17).

ITU (2015) *ICT Facts and Figures 2015*
. Geneva: International Telecommunication Union. Available at: http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2015.pdf (Accessed: 07/11/17).

Logan, T.K., Walker, R. and Hunt, G. (2009) Understanding Human Trafficking in the United States, *Trauma, Violence, & Abuse,* 10(1), pp. 3-30. doi: 10.1177/1524838008327262.

Madhusudan, P.A. and Lambhate Poonam, D. (2017) Deep Web Crawling Efficiently using Dynamic Focused Web Crawler, *International Research Journal of Engineering and Technology (IRJET),* 04(06), pp. 3303.

Mattmann, C. (2015) *Search of the Deep and Dark Web via DARPA Memex (abstract).*

McAlister, R. (2015) *Webscraping As an Investigation Tool to Identify Potential Human Trafficking Operations in Romania.* , Oxford, United Kingdom. 28-01 July 2015. New York, NY, USA: ACM, pp. 2.

Mitchell, R. (2015) *Web scraping with Python: collecting data from the modern web.* O'Reilly Media, Inc.

Molinari, S. (2017) *Google's fight against human trafficking.* Available at: https://www.blog.google/topics/public-policy/googles-fight-against-human-trafficking/ (Accessed: 06/11/17).

Palme, J. and Berglund, M. (2004) Anonymity on the Internet, .

Penman, R.B., Baldwin, T. and Martinez, D. (2009) Web Scraping Made Simple with SiteScraper, .

Scrapy (2017) *Architecture overview.* Available at: https://doc.scrapy.org/en/1.5/topics/architecture.html (Accessed: 24/04/18).

Shen, W. (2014) *Memex.* Available at: https://www.darpa.mil/program/memex (Accessed: 01/11/2017).

Thorn (2017) *Spotlight: Human Trafficking Intelligence and Leads.* Available at: https://www.wearethorn.org/spotlight/ (Accessed: 29/10/17).

UNODC (2016) *Detected victims of trafficking in persons, by age and by sex - 2014 or more recent* United Nations publication.

UNODC (2016) *Global Report on Trafficking in Persons 2016*United Nations publication. Available at: http://www.unodc.org/documents/data-and-analysis/glotip/2016_Global_Report_on_Trafficking_in_Persons.pdf (Accessed: 30/10/2017).

Zhao, F., Zhou, J., Nie, C., Huang, H. and Jin, H. (2016) SmartCrawler: A Two-stage Crawler for Efficiently Harvesting Deep-Web Interfaces, *IEEE transactions on services computing,* 9(4), pp. 608-620. doi: 10.1109/TSC.2015.2414931.

7 Appendices

7.1 Appendix A Use Case Descriptions

Sign in	
Actors	User
Description	Allows a user to access the web application.
Data	Username, password
Stimulus	Command issued by student
Response	The user's signin details are checked for authentication.
Comments	

Authenticate signin	
Actors	User
Description	Checks to see if a user's signin details are valid.
Data	Username, password
Stimulus	Signin details
Response	If successful, the user is authenticated. If unsuccessful, the user is rejected.
Comments	

View report	
Actors	User
Description	Messages found by the spider to be a potential occurrence of human trafficking is displayed with related information from the scraped website.

Data	Buzzword, message, author, date and time, website
Stimulus	Command issued by user, advanced search query
Response	Report is displayed.
Comments	

Advanced search	
Actors	User
Description	Allows a user to search for a specific term, author, date or website stored in the database.
Data	Buzzword, author, date and time, website
Stimulus	Command issued by user
Response	Custom report is displayed.
Comments	

Select target website	
Actors	User
Description	Allows a user to enter a webpage to send to the spider.
Data	Website
Stimulus	Command issued by user
Response	Webpage is sent to the spider.
Comments	

Scrape website	
Actors	Spider

Description	Webpage is crawled through by the spider and information is scraped if a possible occurrence of human trafficking has been detected.
Data	Website, buzzwords
Stimulus	Select target website
Response	Spider crawls and scrapes the target website.
Comments	

Store possible occurrences	
Actors	Spider
Description	If a message is scraped by the spider, it is stored in the database.
Data	Buzzword, message, author, date and time, website
Stimulus	Scrape website
Response	Database is updated.
Comments	

7.2 Appendix B Code

dbsetup.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

import sys

from sqlalchemy import Column, ForeignKey, Integer, String, Text, LargeBinary,
Boolean, Date

from sqlalchemy.ext.declarative import declarative_base

from sqlalchemy.orm import relationship

from sqlalchemy import create_engine


Base = declarative_base()


class Buzzwords(Base):

    __tablename__ = 'buzzwords'


    word = Column(Text, primary_key = True)


    @property

    def serialise(self):

        return {

            'word': self.word

        }




class Catches(Base):

    __tablename__ = 'catches'
    catch_id = Column(Integer, primary_key = True)
    message = Column(Text)
    author = Column(Text)
    word = Column(Text, ForeignKey('buzzwords.word'))
    buzzwords = relationship(Buzzwords)
    catchdate = Column(Text)
    website = Column(Text)

    @property
    def serialise(self):
        return {
            'catch_id': self.catch_id,
            'message': self.message,
            'author': self.author,
            'word': self.word,
            'catchdate': self.catchdate,
            'website': self.website
        }


class User(Base):
    __tablename__ = 'user'

    username = Column(String(20), primary_key = True)
    password = Column(String(20), nullable = False)
    salt = Column(LargeBinary)

engine = create_engine('sqlite:///scrape.db', encoding='utf-8')
```

```
Base.metadata.create_all(engine)
```

dbinsert.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

from sqlalchemy import create_engine

from sqlalchemy.orm import sessionmaker

from dbsetup import *

import hashlib

from os import urandom

import datetime



engine = create_engine('sqlite:///scrape.db')

Base.metadata.bind = engine

DBSession = sessionmaker(bind=engine)

session = DBSession()



words = ['automatic', 'bottom', 'bottom bitch', 'branding', 'caught a case', 'choosing up', 'circuit', 'coercion', 'commercial sex act', 'cousin-in-law', 'cousin in law', 'daddy', 'date', 'exit fee', 'facilitaor', 'family', 'folks', 'finesse pimp', 'romeo pimp', 'force', 'fraud', 'gorilla pimp', 'guerilla pimp', 'head cut', 'human smuggling', 'human traffick', 'in-pocket', 'in pocket', 'john', 'trick', 'kiddie stroll', 'loose bitch', 'lot lizard', 'madam', 'out of pocket', 'pimp', 'pimp circle', 'pimp partner', 'quota', 'eyeballing', 'renegade', 'seasoning', 'serving a pimp', 'squaring up', 'stable', 'the game', 'the life', 'track', 'stroll', 'blade', 'trade up', 'trade down', 'traficker', 'turn out', 'the wire', 'wifey', 'wife-in-law', 'wife in law', 'sister wife']



for word in words:

       buzz = Buzzwords(word=word)

       session.add(buzz)

       session.commit()
```

```
password = b"admin123"

salt = urandom(8)

hash_object = hashlib.sha256(password.encode() + salt)

admin = User(username=u"admin", password=hash_object.hexdigest(), salt=salt)

session.add(admin)

session.commit()
```

reddit.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

import scrapy

from sqlalchemy import create_engine, and_, text

from sqlalchemy.orm import sessionmaker, exc

from dbsetup import *

import datetime


engine = create_engine('sqlite:///scrape.db')

Base.metadata.bind = engine

DBSession = sessionmaker(bind = engine)

dbsession = DBSession()


class Spider(scrapy.Spider):

	name = "reddit"


	def start_requests(self):

		url = 'file:///home/evan/Documents/reddit1.html'

		yield scrapy.Request(url=url, callback=self.parse)



# overall -> <div class="thing ... noncollapsed comment">

# catch_id -> automatic

# message -> <div class="md"><p>

# author -> <a class="author may-blank ...">"

# catchdate -> datetime.now

# website -> htpps://www.reddit.com
```

```
    def parse(self, response):

        count = 0


        comments = response.css(".thing.noncollapsed.comment")

        words = [w.word for w in dbsession.query(Buzzwords.word)] # get list
of buzzwords

        for word in words:

            for comment in comments:

                message =
comment.css(".md").xpath('./p/text()').extract_first()

                if word in message.lower(): # if buzzword is found

                    author =
comment.css("a.author::text").extract_first()

                    # add to database

                    new_catch = Catches(message=message,
author=author, word=word, catchdate=datetime.datetime.now().strftime("%Y-%m-%d %H-
%M-%S"), website=u"https://www.reddit.com")

                    dbsession.add(new_catch)

                    dbsession.commit()

                    count+=1


        self.log('Finished scraping. Found {0} possible
occurrences'.format(count))
```

4chan.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

import scrapy

from sqlalchemy import create_engine, and_, text

from sqlalchemy.orm import sessionmaker, exc

from dbsetup import *

import datetime


engine = create_engine('sqlite:///scrape.db')

Base.metadata.bind = engine

DBSession = sessionmaker(bind = engine)

dbsession = DBSession()


class Spider(scrapy.Spider):

	name = "4chan"


	def start_requests(self):

		url = 'file:///home/evan/Documents/4chan.html'

		yield scrapy.Request(url=url, callback=self.parse)




# overall -> <div class="postContainer">

# catch_id -> automatic

# message -> <blockquote>

# author -> <span class="name">"

# catchdate -> datetime.now

# website -> htpp://www.4chan.org
```

```
    def parse(self, response):

        count = 0


        comments = response.css(".postContainer") # gets both OP and replies

        words = [w.word for w in dbsession.query(Buzzwords.word)] # get list
of buzzwords

        for word in words:

            for comment in comments:

                message =
comment.css(".post").xpath('./blockquote/text()').extract_first() # extract
comment

                if message == None:

                    message = ""

                if word in message.lower(): # if buzzword is found

                    authorName =
comment.css("span.name::text").extract_first()

                    authorNo = comment.css('a[title="Reply to this
post"]::text').extract_first()

                    author = authorName+" "+authorNo

                    # add to database

                    new_catch = Catches(message=message,
author=author, word=word, catchdate=datetime.datetime.now().strftime("%Y-%m-%d %H-
%M-%S"), website=u"http://www.4chan.org")

                    dbsession.add(new_catch)

                    dbsession.commit()

                    count+=1


        self.log('Finished scraping. Found {0} possible
occurrences'.format(count))
```

app.py

```
from flask import Flask, render_template, request, redirect, url_for, jsonify,
json, Response, session, g, flash, request, make_response

from sqlalchemy import create_engine, and_, text

from sqlalchemy.orm import sessionmaker, exc

from dbsetup import *

from werkzeug.exceptions import abort

import hashlib

import codecs

import re

from spiders import reddit

from twisted.internet import reactor

import scrapy

from scrapy.crawler import CrawlerRunner

import os


app = Flask(__name__)


# initialise the database

engine = create_engine('sqlite:///scrape.db')

Base.metadata.bind = engine

DBSession = sessionmaker(bind = engine)

dbsession = DBSession()




# open the index page

@app.route('/', methods=['GET', 'POST'])

def main():
    if g.user: # check for user session
        if request.method == 'POST': # webform submission
            url = request.form['site']
            url = url.replace("/", "%2f")
            return redirect(url_for('spider', url=url)) # run the spider
        return render_template('index.html')
    return redirect(url_for('signin'))

# run the spider
@app.route('/run-spider/<url>')
def spider(url):
    if g.user: # check for user session
        if 'reddit' in url:
            os.system('scrapy crawl reddit') # run command
        elif '4chan' in url:
            os.system('scrapy crawl 4chan')
        else:
            return redirect(url_for('main'))
        # ...and so on...
        return render_template('spider.html')
    return redirect(url_for('signin'))

# display report
@app.route('/report/')
def report():
    if g.user: # check for user session
        query = dbsession.query(Catches)
```

```
            return   render_template('report.html',   query=([catch.serialise   for
catch in query]))

        return redirect(url_for('signin'))


# generate dynamic query

@app.route('/report/search/', methods=['GET', 'POST'])

def advancedSearch():

        if g.user: # check for user session

                if request.method == 'POST':

                        if request.form['field']:

                                field = request.form['field']

                        if request.form['param']:

                                param = request.form['param']

                                param = param.replace("/", "%2f")

                        elif not request.form['param']:

                                param = "%"

                        return    redirect(url_for('advancedReport',    table="catches",
field=field, param=param))

                return render_template('advanced.html')

        return redirect(url_for('signin'))


# generate dynamic query

@app.route('/report/<table>?<field>=<param>')

def advancedReport(table, field, param):

        if g.user: # check for user session

                param = param.replace("%2f", "/")

                query  =  dbsession.execute("select  *  from  {0}  where  {0}.{1}  like
'%{2}%'".format(table, field, param))

                rows = query.fetchall()
```

```
             result = []

             for row in rows:

                    result.append(dict(row))

             return  render_template('report.html',  table=table,  field=field,
param=param, query=result)

      return redirect(url_for('signin'))


# compare password with database

def check_password(hashed_password, user_password, salt):

    return  hashed_password  ==  hashlib.sha256((user_password.encode())  +
salt).hexdigest()


# compare credentials with database

def validate(username, password):

    completion = False

    users = dbsession.query(User)

    for user in users:

        if user.username == username:

            completion = check_password(user.password, password, user.salt)

    return completion


# sign in user

@app.route('/signin/', methods=['GET', 'POST'])

def signin():

    error = None

    if request.method == 'POST':

        session.pop('user', None)

        uname = request.form['username']

        pword = request.form['password']
        completion = validate(uname, pword)

        if completion == True:

            session['user'] = uname # create user session

            return redirect(url_for('main'))

        else:

            error = 'Invalid Credentials. Please try again.'

    return render_template('signin.html', error=error)


# delete session

@app.route('/signout/')

def signout():

    session.pop('user', None)

    return redirect(url_for('main'))


 # create user session

@app.before_request

def before_request():

    g.user = None

    if 'user' in session:

        g.user = session['user']




if __name__ == '__main__':

    app.secret_key                                                              =
"\xc2\x0f\xdc\x9d0\x10A\xfa:DO\xcf\xa8%\xf0\x8e\xc1\xcb=\xf8$\xaa\xc8\xfb"

    app.debug = True

    app.run(host = '0.0.0.0', port = 5000)
```

layout.html

```
<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>{% block title %}{% endblock %} - Spyder</title>

    <link  rel="shortcut  icon"  type="image/x-icon"  href="{{  url_for('static', filename='img/favicon.png') }}" />

    <link  rel="stylesheet"  type="text/css" href="https://fonts.googleapis.com/css?family=Open+Sans">

    <link  rel="stylesheet"  type="text/css"  href="{{  url_for('static', filename='css/bootstrap.min.css') }}">

    <link  rel="stylesheet"  type="text/css"  href="{{  url_for('static', filename='css/style.css') }}">

    <script  src="{{  url_for('static',  filename='js/jquery-2.2.4.min.js') }}"></script>

    <script  src="{{  url_for('static',  filename='js/bootstrap.min.js') }}"></script>

</head>

<body>

    <div class="container-fluid">

        <header class="row">

            <div class="col-xs-12 col-head">

                <nav class="navbar navbar-default">

                    <div class="container-fluid">

                        <div class="navbar-header col-lg-3">

                            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"  data-target="#bs-example-navbar-collapse-1"  aria-expanded="false">

                                <span class="sr-only">Toggle navigation</span>
                                <span class="icon-bar"></span>
                                <span class="icon-bar"></span>
                                <span class="icon-bar"></span>
                            </button>
                            <a class="navbar-left" href="{{ url_for('main') }}">
                                <img class="img-responsive navimg" src="{{
url_for('static', filename='img/spyder-logo.png') }}" alt="Spyder | A Scrapy Tool"
title="Spyder | A Scrapy Tool">
                            </a>
                        </div>
                        <div class="collapse navbar-collapse" id="bs-example-
navbar-collapse-1">
                            <ul class="nav navbar-nav col-lg-9">
                                <li><a href="{{ url_for('main')
}}">Spider</a></li>
                                <li><a id="db" href="{{ url_for('report')
}}">Catch Report</a></li>
                                <li><a href="{{ url_for('advancedSearch')
}}">Search</a></li>
                                {% if g.user %}
                                <li class="signin"><a href="{{ url_for('signout')
}}">Sign Out</a></li>
                                {% else %}
                                <li class="signin"><a href="{{ url_for('signin')
}}">Sign In</a></li>
                                {% endif %}
                            </ul>
                        </div>
                    </div>
                </nav>
            </div>
        </header>
        <div class="col-xs-12">

            {% block content %}{% endblock %}

        </div>

    </div>

    {% block script %}{% endblock %}

    <script>

        $("#menu").click(function(e) {

            $("#nav").addClass("open");

            e.stopPropagation();

        });

    </script>

</body>

</html>
```

index.html

```
{% extends "layout.html" %}

{% block title %}Spider{% endblock %}

{% block content %}

<div class="row">

    <div class="col-md-9 col-xs-12 col-head">

        <span class="h1">Scrape target website</span>

        <div class="row">

            <div class="col-xs-12 col-data">

                <form id="spiderform" method="POST">

                    <div class="col-xs-12">

                        <div class="col-xs-12 form-group col-req">

                            <label for="site">Please enter a webite you wish to
scrape</label>

                            <select class="form-control" name="site" id="site">

<option>file:///home/evan/Documents/reddit1.html</option>

<option>file:///home/evan/Documents/4chan.html</option>

                            </select>

                        </div>

                        <div class="col-md-2 col-xs-12 form-group submit">

                            <input type="submit" value="Submit" class="btn btn-md
btn-primary btn-block submitBtn">

                        </div>

                    </div>

                </form>

            </div>

        </div>

    </div>
```

```
</div>

{% endblock %}

{% block script %}

{% endblock %}
```

spider.html

```
{% extends "layout.html" %}

{% block title %}Scraping website...{% endblock %}

{% block content %}

<div class="row">

    <div class="col-xs-12 col-head">

        <span  class="h1">Please  wait  while  the  spider  scrapes  the  target
website...</span>

    </div>

</div>

{% endblock %}

{% block script %}

<script>

    setTimeout(function() {

        $('#db')[0].click();

    }, 5000);

</script>

{% endblock %}
```

report.html

```
{% extends "layout.html" %}

{% block title %}Catch report{% endblock %}

{% block content %}

<div class="row">

    <div class="col-xs-12 col-head">

        <span class="h1">Catch report</span>

        <div class="row">

            <div class="col-xs-12 col-data table-overflow">

                <table class="table" id="report"></table>

            </div>

        </div>

    </div>

</div>

{% endblock %}

{% block script %}

<script>

    var catches = {{ query|tojson|safe }};


    $(function(){

        var table = $("#report");

        var row = $("<tr>");

        row.append("<th>Buzzword",   "<th>Message",   "<th>Author",   "<th>Date",
"<th>Website");

        table.append(row);

        for (i = catches.length-1; i >= 0; i--) {

            var row = $("<tr>");

            row.append("<td  class='col-xs-2'>"+catches[i].word,  "<td  class='col-
xs-5'>"+catches[i].message,   "<td   class='col-xs-2'>"+catches[i].author,   "<td
class='col-xs-1'>"+catches[i].catchdate,             "<td             class='col-xs-
2'>"+catches[i].website);

            table.append(row);

        }

    });

</script>

{% endblock %}
```

advanced.html

```
{% extends "layout.html" %}

{% block title %}Advanced search{% endblock %}

{% block content %}

<div class="row">

    <div class="col-md-9 col-xs-12 col-head">

        <span class="h1">Advanced search</span>

        <div class="row">

            <div class="col-xs-12 col-data">

                <form  id="reportForm"  action="{{  url_for('advancedSearch')  }}"
method="POST">

                    <div class="col-xs-12">

                        <div class="col-xs-12 form-group col-req">

                            <span><strong>Field</strong></span>

                            <select class="form-control" name="field" id="field">

                                <option value="word">Buzzword</option>

                                <option value="author">Author</option>

                                <option value="catchdate">Date</option>

                                <option value="website">Website</option>

                            </select>

                        </div>

                        <div class="col-xs-12 form-group">

                            <span><strong>Parameter</strong></span>

                            <input  class="form-control"  type="text"  name="param"
id="param">

                        </div>

                        <div class="col-md-2 col-xs-12 form-group submit">

                            <input type="submit" value="Search" class="btn btn-md
btn-primary btn-block submitBtn">
```

```
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

signin.html

```
{% extends "layout.html" %}

{% block title %}Sign in{% endblock %}

{% block content %}

<div class="col-centered col-xs-12 col-md-6 col-lg-4">

<form class="form-signin" id="signin" autocomplete="off" method="POST">

    <h2 class="form-signin-heading">Please sign in</h2>

    <p  class="col-hidden  error"  id="error"><span  class="glyphicon  glyphicon-exclamation-sign" aria-hidden="true"></span>&nbsp;&nbsp;{{ error }}</p>

    <div class="form-group col-req">

        <input  type="text"  name="username"  id="username"  class="form-control" placeholder="Username" required autofocus>

    </div>

    <div class="form-group col-req">

        <input type="password" name="password" id="password" class="form-control" placeholder="Password" required>

    </div>

    <div class="form-group submit">

        <input type="submit" value="Sign In" class="btn btn-md btn-primary btn-block submitBtn">

    </div>

</form>

</div>

{% endblock %}

{% block script %}

<script    src="{{    url_for('static',    filename='js/jquery.validate.min.js') }}"></script>

<script>

    $("#loggedUser").hide();

    $(".btn-link").hide();
    $(".dropdown").hide();

    $("footer").hide();


    $(function() {

        if ($("#error").text() !== "\xa0\xa0None") {

            $("#error").removeClass("col-hidden");

        }

    });


    $("#signin").validate({

        highlight: function(element) {

            $(element).closest('.col-req').addClass('has-error');

        },

        unhighlight: function(element) {

            $(element).closest('.col-req').removeClass('has-error');

        },

        errorElement: 'span',

        errorClass: 'help-block',

        errorPlacement: function(error, element) {

            if(element.parent('.input-group').length) {

                error.insertAfter(element.parent());

            } else {

                error.insertAfter(element);

            }

        }

    });

</script>

{% endblock %}
```

style.css

```
.col-hidden {

    display: none;;

}


.table-overflow {

    overflow-x: auto;

}


.col-icon {

    margin-top: 20px;

    margin-left: 20px;

}


.col-data {

    margin-top: 20px;

}


.submit {

    margin-top: 20px;

}


.col-centered {

    float: none;

    margin: 0 auto;

}


.error {
    color: #a94442;
}

.navbar-default {
    border-color: transparent;
}

@media screen and (min-width: 992px) {
    .col-line {
       border-left: 1px solid gray;
       float:left;
    }

    .col-pc {
        display: initial;
    }

    .col-mobile {
        display: none;
    }

    .col-head {
        padding-top: 15px;
    }
}

@media screen and (max-width: 991px) {
    .col-line {
        border-top: 1px solid gray;

        padding-top: 5px;

    }


    .col-pc {

        display: none;

    }


    .col-mobile {

        display: initial;

    }


    .col-head {

        padding-top: 5px;

    }


    .navimg {

        max-width: 195px;

    }


    .nav {

        display: flex;

        flex-direction: column;

    }

}


@media screen and (max-width: 1199px) {

    .col-search {
        padding-top: 15px;
    }
}
```

7.3 Appendix C Test Suite

test.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

import os

import app

import unittest

import tempfile

from flask import url_for


class AppTestCase(unittest.TestCase):


    def setUp(self):

        self.db_fd, app.app.config['DATABASE'] = tempfile.mkstemp()

        app.app.config['DEBUG'] = False

        app.app.config['TESTING'] = True

        app.app.config['SERVER_NAME'] = 'myapp.dev:5000'

        app.app.config['SECRET_KEY'] = 'secret'

        self.app = app.app.test_client()


    def tearDown(self):

        os.close(self.db_fd)

        os.unlink(app.app.config['DATABASE'])




    def signin(self, username, password):

        with app.app.app_context():

            return self.app.post(url_for('signin'), data=dict(
                username=username,
                password=password
            ), follow_redirects=True)

    def signout(self):
        with app.app.app_context():
            return self.app.get(url_for('signout'), follow_redirects=True)

    def test_1_signin_signout(self):
        with app.app.app_context():
            rv = self.signin('admin', 'admin123')
            assert 'Scrape target website' in rv.data
            rv = self.signout()
            assert 'Please sign in' in rv.data
            rv = self.signin('adminx', 'admin123')
            assert 'Invalid Credentials. Please try again.' in rv.data
            rv = self.signin('admin', 'defaultx')
            assert 'Invalid Credentials. Please try again.' in rv.data

    def test_2_index(self):
        with app.app.app_context():
            self.signout()
            rv = self.app.get(url_for('main'), follow_redirects=True)
            assert 'Please sign in' in rv.data
            self.signin('admin', 'admin123')
            rv = self.app.get(url_for('main'))
            assert 'Scrape target website' in rv.data
    def test_3_spider(self):

        with app.app.app_context():

            self.signout()

            url = "reddit.com"

            rv      =      rv      =      self.app.get(url_for('spider',     url=url),
follow_redirects=True)

            assert 'Please sign in' in rv.data

            self.signin('admin', 'admin123')

            url = "reddit.com"

            rv = self.app.get(url_for('spider', url=url))

            assert 'Please wait while the spider scrapes the target website...' in
rv.data

            url = "4chan.org"

            rv = self.app.get(url_for('spider', url=url))

            assert 'Please wait while the spider scrapes the target website...' in
rv.data

            url = "invalidsite.com"

            rv = self.app.get(url_for('spider', url=url), follow_redirects=True)

            assert 'Scrape target website' in rv.data


    def test_4_report(self):

        with app.app.app_context():

            self.signout()

            rv = self.app.get(url_for('report'), follow_redirects=True)

            assert 'Please sign in' in rv.data

            self.signin('admin', 'admin123')

            rv = self.app.get(url_for('report'))

            assert 'Catch report' in rv.data
    def test_5_search(self):

        with app.app.app_context():

            self.signout()

            rv = self.app.get(url_for('advancedReport', table="catches",
field="word", param="pimp"), follow_redirects=True)

            assert 'Please sign in' in rv.data

            self.signin('admin', 'admin123')

            rv = self.app.get(url_for('advancedReport', table="catches",
field="word", param="pimp"))

            assert 'Catch report' in rv.data




if __name__ == '__main__':

    unittest.main()
```