NEWSENHANCER

MASTER THESIS IN SOFTWARE ENGINEERING

Department of Computing, Mathematics and Physics,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

Sindre Lilland Nerdal

June 2018

Abstract

This thesis describes a web browser plugin called NewsEnhancer, which has the goal to make it more efficient to read the news.

The plugin receives data from a server application, which continuously aggregates meta information from articles on various news sites. This meta information is then presented to the user through the plugin. This can help the user to decide whether or not to click a headline link, which then can make the reading process more efficient.

The plugin is also community based, and enables users to report headlines, as clickbait or fake news, and submit summaries of articles. The reported headlines will be flagged for other users, as to signal that this has been reported as a clickbait or fake news. The submitted summary will appear alongside the other meta information.

Acknowledgements

I would first and foremost like to thank my supervisors at The Western Norway University of Applied Sciences, Atle Geitung Harald Soleim and Daniel Patel for their continuous guidance and support during this thesis. I would also like to thank Lars Nyre at University Of Bergen for valuable input and suggestions.

Table of Contents

1 Introduction

This chapter starts by introducing the motivation behind this thesis. Then proceeding to describe the thesis goals and how they may be achieved. The last two sections are related work and the thesis outline. The former describes existing solutions, and what this thesis will do differently. The latter explains the structure of this thesis and the content of each chapter.

1.1 Motivation

Newspapers generate income through sale of physical copies, from exposing readers to advertisements on web, by minimizing resources spent on each article and from paid subscriptions. The three first sources of income can be increased by producing headlines and front-pages that create curiosity in readers and by producing low-quality articles. These sales techniques result in reduced reading quality for the news consumer. They were described already in 1941 by Mott and named yellow journalism[1]. Mott defined five characteristics of yellow journalism:

1. Scare headlines in huge print, often of minor news.
2. Lavish use of pictures, or imaginary drawings.
3. Use of faked interviews, misleading headlines, pseudo-science, and a parade of false learning from so-called experts.
4. Emphasis on full-colour Sunday supplements, usually with comic strips.
5. Dramatic sympathy with the "underdog" against the system.

How the newspapers generates income online is very differently with sales from physical copies. Whenever someone wants to read one or more articles they, have to buy the whole physical copy. This copy contains advertisement that the newspaper has already been paid for, so the newspaper expects each customer to buy just one copy. However online, the newspapers still sell all the articles as one package via subscriptions, but unlike the physical copies, the newspaper is not paid a fixed amount for advertisement. Also, a lot of the articles online is "free" in the sense that the user does not pay for it directly, but by including different types of advertisement. This advertisement will generate income by different methods, but most of them will fall under two categories either by "clicks" or by "views". As the name suggests, the former requires a user to interact with the advertisement and click on it, before the newspaper is paid by the advertisement company. The latter only requires the user to “view” the advertisement, so opening a website with a “view” advertisement would generate income. However the generated income from “view” advertisement is most likely in a much smaller scale then by “click”. This leads to news sites trying to get the users to either click on an advertisement or to click on articles that contain advertisement. This in turn can lead to the same results as what Mott[1] called yellow journalism.

Another consequence of yellow journalism online is that the user use unnecessary time to click on what looks to be an interesting headline which leads to an article which does not comply with the expectations. Another factor could be that the user has too little information to go on to decide which article to click and which to ignore.

Arthur S. Brisbane stated in 2011 that “Unlike print, digital news is often updated throughout the day and night, sometimes many times. Versions evolve and sometimes morph into something quite different. Mistakes happen and are fixed.”[2]. The problem is what were the mistakes and what was fixed? There is very little to nothing, besides a timestamp, that marks an article as updated. Some sites, like vg.no, adds a small yellow banner at the top of the page, signalling that the article you are currently reading has been updated. However the actual changes are not highlighted at all. The user would have to read the article once again to find what was updated. In addition, it is quite easy for news sites to just remove content and pretend that the text never did exist. This can also happen with whole articles and not just with some part of the article content.

A prime example of an article where its content is removed, is given by Jennifer Lee in an interview with Arthur in 2012. She describes one article by the NYTimes, which described the account of an Occupy Wall Street confrontation between protesters and police. “In an early version, the police were characterized by The Times as allowing protesters onto the bridge, but then cutting them off and arresting them. In an account 20 minutes later, there was no mention of the police allowing protesters onto the bridge; instead, the protesters were portrayed as moving on their own initiative to occupy the bridge”[3]. The change seemed to shift the blame, and The Time’s got a lot of angry comment about the article and their handling of it.
These problems described above, are addressed in this thesis plugin.

1.2 Thesis Goal

When reading the front page of a news site, one typically scans the headlines and any accompanying photos, before deciding from this information which articles to read. If the decision has been taken to open an article, it is typically to either read carefully, skimmed through, or exited quickly. The reason for exiting is when the reader realized that the article was not interesting or did not match the headline. In many cases, the user has spent time in vain reading an article which the user would not read, if more information had been given up front. A goal for this thesis is to provide extra information to e.g. facilitate the user to decide which articles are worth reading.

There is a lacking feedback mechanism for low quality journalism. When the number of clicks is used as a measure of article quality, this only encourages more use of clickbait headlines, despite that it on the contrary most likely reduces reader satisfaction. In addition, journalists often make sure that public figures are held responsible for their wrongdoings, but they themselves can to a larger degree get away with their own wrongdoings. A goal is to allow users to give feedback on the articles they read by e.g. labelling them as clickbait, and by associating the feedback with article authors. Then one could accumulate statistics on each journalist and present it to the users, to make the journalists more accountable for their work.

To summarize the main two thesis goals:

1) To produce quick overview of news by e.g. helping readers avoid spending time on irrelevant news, and guide readers towards relevant news

2) To make the journalists and newspapers more accountable for their work.

The 1. goal can be achieved by:

a) Showing an article's metadata when the user is reading the article's headline. This is to increase the information base the user has in order to make a decision on whether or not to click that headline.

b) Summarizing the front page of a news site in one single picture by generating a Word cloud from all headlines, and summarizing single articles by generating a Word cloud from the article text.

c) Allow readers to flag articles as clickbait or fake news, and displaying a flag on headlines that have been reported as such.

d) Allow readers to submit one-line summaries of articles, and showing these summaries in the articles metadata when hovering over the the article's headline.

The 2. goal can be achieved by:

a) Enabling information sharing between readers, so they can warn each other regarding clickbait and fake news.

b) Being able to compare the current version of headlines and articles content with their previous version, to track the changes journalists make.

c) Collecting and presenting statistics on journalists regarding e.g. how often their articles are reported as clickbait or fake news.

1.3 Related work

1.3.1 Clickbait and fake news

There already exists several chrome plugins that alerts the user about fake news or clickbait. These plugins rely on methods such as cross referring with lists of URL's known to contain fake news, or by using algorithmic methods that analyse the article content.

Url lists

URL lists are typically either updated by the plugin users by having a reporting mechanism, or using externally moderated lists. Plugins that are supposed to work for any links on the internet are prone to have URL lists with too low coverage for practical use. The browser plugin B.S. detector[4] relies on an open source database called OpenSources[5], headed by Melissa Zimdars of Merrimack College. Here websites are labelled with one or more categories such as Fake News, Satire, Extreme Bias, Conspiracy Theory, Rumor Mill, Junk Science, Clickbait and Credible.

Algorithmic approach

Examples of algorithmic approaches include DownWorthy[6] that reformulates link titles such as "Will Blow Your Mind" to "Might Perhaps Mildly Entertain You For a Moment". The plugin also categorizes a link as Clickbait or not, by checking if it contains certain words or phrases. The plugin TLDR[7], which stands for **T**o **L**ong, **D**id not **R**ead (a phrase used to present a summary of a blog post), automatically creates four summaries of an article in decreasing size by using a content summarization algorithm. The plugin "This is Clickbait"[8] uses a pre-trained artificial neural network building on code publicly available on GitHub[9]. The algorithm assigns a percentage probability of the headline being clickbait. It has been trained using 12,000 headlines taken from different English language news sites, half of which were clickbait articles. No statistics are shown regarding the reliability of the artificial neural network on non-training data, making it is hard to know how well it works on headlines in general.

B.S Detector[4] use an open source database[5] to flag whole sites as satire, fake news, clickabit etc. Flagging whole sites may work for sites that wants to spread misinformation, hate etc. But flagging respected sites that only contains a few clickbait would not be fair.

DownWorthy[6] tries to warn users about clickbait by reformulate obvious clickbait titles, but relies on a wordlist of clickbait phrases and their "not so clickbait anymore" counterpart phrases. For instance, the clickbait phrase "You Wont Believe" will get replaced by "In All Likelihood, You'll Believe". However this requires the developer behind DownWorthy to manually add new phrases which can be very cumbersome. In addition it only works for English phrases for now.

ThisIsClickbait[8] tries to solve clickbait via artificial intelligence and by training the neural network with 6,000 clickbait headlines and 6,000 non clickbait headlines. However all the headlines in the training data is English headlines, so it cannot be used for this thesis which focuses on Norwegian news sites.

1.3.2 Crowdsourced enhancement of articles

There exists a browser plugin, called TL;DR[10] (different from TLDR[7]), which enables users to share comments on links. It is meant for summarizing articles, but there is no control or moderation of the comments. Since the plugin does not focus on any particular websites, it is likely that its coverage will be thinly distributed. Rbutr[11] is a related plugin, where users can write rebuttals on links that they dispute, and they can refer to other articles that support their rebuttal. Links having rebuttals will be visible for all users of the plugin.

TL;DR[10] allows users to right click on any link (`<a href="example.com">Link</a>`) and submit comments. These comments are available for all other users who use the same plugin. This is a great way to let users share information and experience about links to different sites, but there is no form for moderation at all. This means that comments can contain profanity, racism, misleading information and etc. This can make the matter worse for the user.

A similar plugin, with a very similar name, TLDR[7], aggregates article summaries of different length via artificial intelligence. A user can aggregate these summaries by right clicking on links, or by clicking on the plugin icon when an article is opened. However, it does not always produce a result, and the aggregating process can sometimes be very slow.

It would be easier and faster for the user if summaries was pre-generated on a server and then shown with all the other metadata. However using similar techniques as TLDR[7] with artificial intelligence to generate it, would require too much effort into a very small part of the thesis goals. That is why this thesis focus on user submitted summaries. But unlike TL;DR[10] all submission should be reviewed by a moderator who choose which summary fits best with the article. This is to safeguard the integrity of the summaries.

There exists other browser plugins, like CrowdsouRS[12], which use crowdsourcing to aggregates reputation scores for a web page from multiple users. This is score can help other users to determine if the contents of the web page are deceptive.

However, the user would have to be on the web page to see the web page's score. This means that if the user would want to check if the headline leads to an "trusted" article, then the user would have to click on the headline, which defeats the point this thesis is trying to prove by providing information about the headline, before the user decide whether or not to click the headline.

1.3.3 Tracking article changes

NewsDiff

NewsDiffs.org[13] is an open source website and framework, that track changes in online news articles over time by scraping the article's on different news sites front page. It was developed by a journalist and two programmers under a Knight-Mozilla hackaton at M.I.T Michigan USA in June 2012. The site keeps track of changes within news articles for nytimes.com, cnn.com, politico.com, washingtonpost.com, and bbc.co.uk.

NewsDiffs has made it easy to extend the framework and support more sites by creating a parser that's sets the base for what information would be extracted, and where in the website to extract it from. To add support for a site, just create a new parser, which inherit from the base parser, and extract the same type of data that the other parsers do.

NYTdiff

NYTdiff[14] is a twitter bot that use the API at NYTimes to regularly fetch all top articles. Each of these articles is then web scraped to extract the content to check if it is updated from the previous version. If it has, the bot creates an image showing the updated paragraph and highlights the deleted text with a red background and the added text with a green background. This image is also tweeted on twitter under the hashtag `#nyt_diff`.

DiffEngine

DiffEngine[15] is a utility tool developed by a group from Maryland Institute of Technology in its participation with the "Document the Now" organization. It is partly based upon NYTdiff, but use the RSS feed to monitor articles instead. In addition, the article's website is also archived at the "Internet Archive".

All three change tracking tools NYTdiff[14], diffEngine[15] and NewsDiff[13] use web scraping to retrieve the content of an article. The only difference is that NYTdiff and diffEngine only supports nytimes.com, but NewsDiff also supports cnn.com, politico.com, washingtonpost.com, and bbc.co.uk. All of them defines a parser which is either a function or a class that defines what data will be extracted and from where in the HTML document. Since each news site's HTML structure is different, NewsDiff requires multiple parsers. Actually one parser per article layout (a news site may represent articles differently across the site. An example is dn.no which has "normal" articles, but also a magazine type. Their HTML is structured differently and requires two different parser). This means that the person implementing such a parser would need programming experience. In addition, the web application itself must be restarted whenever a new parser is added, which also requires a technical person. The reason why a restart is necessary is because the web application only registers new files or new code in existing files upon start.
It would be far easier if a general parser was used which knew what should be extracted, but received a template with where it should look for the data. Then these templates could be stored in a database and accessed by the application. This would in turn mean that the web application would not have to restart to support additional sites.

In addition the people who would add the templates would not need to be too technical if using CSS selector to define the path to the information. This is because the CSS selector can be found very easily in a Chrome browser by using the “inspector view”.

NYTdiff[14] and diffEngine[15] posts the article diff as an image on twitter. Not the whole article, but just the part that is updated. This means that the user would have to manually scroll back in the twitter post history to find specific articles. The user would also just see partly updates and would have to fit it into the article on their own. NewsDiff[13] has a better solution, where the user can search specific articles by their URL and then view all the changes on NewsDiffs own page. NewsDiff also enables the user to go back in time and watch the changes for each time the article was updated. However all the articles on NewsDiff contains just the article text. It lacks the articles images and site styling which gives the impressions that you are reading a letter instead of an article. In addition, the user would still have to access a different site then the article site to view the changes.
This thesis propose a Chrome plugin where the user can view the changes inline inside the article. That would be more easy and convenient for the user. The user would not have to relate to external sites. In addition, one could also keep track of the images used. The experience would then feel more authentic when reading the diff within the article, because it looks like a normal article.

1.3.4 Fact checking

There exists a unit at Poynter Institute, in Florida USA, called International Fact Checking Network(IFCN)[16] that is dedicated to bring together fact-checkers world wide. The IFCN has a code of conduct which is a collection of principles the members must abide by after they have applied and been vetted which ensures that all fact-checkers upholds the same standard.

A commitment to nonpartisanship and fairness Holding each fact-check to the same standard

A commitment to transparency of sources Provide sources and let readers verify their findings.

A commitment to transparency of funding and organization Ensure that funding from organization has no influence over conclusion in fact-checking.

A commitment to transparency of methodology Explain the methodology used on each step of the fact-checking process.

A commitment to open and honest corrections Publish the corrections to ensure that readers can see the corrected version.

Faktisk.no[17], owned by the Norwegian news sites vg.no, db.no, nrk.no and tv2.no, is an independent organization made up of journalist for fact-checking the social debate in

Norway. Faktisk.no was released in July 2017 and became an IFCN member in October 2017.

Snopes.com[18] is a popular fact-checker. It was founded by David Mikkelson in 1996 to debunk urban legends, but has since then gained a lot of traction in “normal” articles. The team consist of both writers and journalists with backgrounds in media, science, social media and related fields. Snopes is also a member of IFCN.

Both Faktisk.no[17] and Snopes.com[18] requires the user to visit their respective web page to view the fact check. Most sites do not inform the user that an external fact checking site has done an analysis of the article. This means that the user must actively visit these fact checking sites and search for specific articles on their own and then possible find nothing of interest. They both also have dedicated people, often journalists, to investigate whether or not the claims in the article is true or not. This would be really interesting to add in the thesis framework, but neither sites have an API serving these data which means it must be collected via web scraping. This will not be done in this thesis.

1.4 Thesis outline

Chapter 1 - Introduction
This chapter contains the related work, and discusses the goals and motivation behind this thesis.

Chapter 2 - Background
This chapter contains a introduction to the software and theories used in this thesis.

Chapter 3 - Problem description
This chapter contains an analysis of the problem.

Chapter 4 - Design and solution
This chapter contains a description on how the problems investigated in the thesis have been solved.

Chapter 5 - Results and discussion
This chapter contains the obtained results and a discussion around the challenges when trying to reach the thesis goals.

Chapter 6 - Conclusion
This chapter contains a summary of the presented results.

Chapter 7 - Further work
This chapter contains what further work remains.

2 Background

Some basic background knowledge is needed to know how the tools used in this thesis works and how they are implemented. It is central to understand how a website is represented as a HTML document, and how that document is structured. This is explained in the first section of this chapter. The next sections described how to retrieve information from an HTML document. This is done by web scraping, but that is not the only way to retrieve information. Some sites also serve their content directly from a API. This is described next. The last section of this chapter explains how a plugin for the Chrome web browser works.

2.1 Hyper Text Markup Language

HTML is the standard markup language for creating websites. A HTML file, or document, is made up of different elements which usually consist of a start-tag, content and an end-tag. The different variations is listed in the code snippet 1 from line 14 to 20 (on page 11).

Each start-tag can hold multiple attributes. The two most common attributes are id and class. Id contains a number and gives the element an unique id. The class attribute contains one or more CSS classes where each class defines some type of styling on how the element should be rendered in the browser.

```html
<!DOCTYPE html> <!-- Document type declaration -->
<html>
    <!-- Meta-information -->
    <head>
        <!-- Include a style for the document -->
        <link rel="stylesheet" type="text/css" href="theme.css">

        <!-- Include a javascript file -->
        <script src="myScript.js"></script>
    </head>

    <!-- Site content -->
    <body>
        <!-- Element contains another element. -->
        <div>
            <!-- Element with attributes id and class -->
            <h1 id="24" class="text-center"> Text content </h1> <!-- -->
        </div>
        <!-- Element without content  -->
        <img id="2" src="source_image.png" /> <!-- Self-closing tag  -->
    <body>
</html>
```

Listing 1: Structure of a HTML document where some tags have attributes

Whenever a browser requests a site, for instance example.com, the server returns a HTML document back to the client browser, based on the submitted URL. The document undergoes a process called rendering after it is downloaded. This rendering process is what translates a HTML document, as shown in code snippet 1, to the content seen in a browser. During this process the browser only presents the tags content and not the tags itself. Styling is applied on the content according to the rules stated in the CSS.

2.2 Web scraping

Web scraping or web extraction is a technique used to automate the process of retrieving all data or some part of it from websites. An example is to extract the text or the URL from all of the `<a href="urlToASite.com">Text</a>` elements on a web site. This could also be done manually, but it would be far too cumbersome.
Web scraping is done by downloading the HTML document and then scraping the document by finding what data one would want to extract. The finding and extraction can be done in many different ways, one for instance is regex.

Regex
Regex, or Regular Expressions, is a sequence of characters that defines a search pattern. These patterns can be used to find, replace or extract data from a text. However the patterns can be very complex. To match any given HTML tag in a text one could use `<TAGb[^>]*>(.*?)</TAG>`. In addition, this is not human readable at all.

However there exists far easier ways to extract data from a HTML document. Most of them relies on external libraries, which parse the downloaded document into a tree structure, where each tag (`<tag></tag>` is treated as a parent to all its enclosing tags (child's). This tree structure can then be queried to extract data for specific tags. The different types of queries vary from library to library. The library used in this thesis, BS4[19], contains queries by XPath, tag and CSS selectors.

XPath
XPath, or XML Path Language, is a method to identify and navigate through nodes in an XML document. An HTML documents is based upon XML, so it can use XPath to traverse the document. However the query is not really human readable. An example of XPath is `//*[@id="24"]` which represents the path to the tag on line 17 in the code snippet 1(on page 11).

Tag
Searching by tag is done by identifying a specific HTML tag or group of tags. This can be done by specifying the tag itself, but also it attributes. An example to find the same element as in the XPath example (line 17 in code snippet 1) would be by `tree_like_structure.find('div', "id": 24)`).

CSS selector
Searching by CSSselector is done by identifying a specific HTML tag or group of tags by its attributes. An example to find the same element as in the XPath example (line 17 in code snippet 1) would be by `tree_like_structure.select_one('#24')`

2.2.1 Rate limited services

Web API's often has a form of "rate limiting". It is a way to limit the amount a user can access the Web API's resources. Google use a day by day rate limiting for all its free Web API[20]. This means that the Web API will not return the requested content after the rate is exceeded. This is only for the free version. If using the subscription version, then the content will return when exceeding the rate limit, but each request after has an additional cost.

A web scraper is only limited to what a normal user has access to. This is because the web scraper extracts data from the same HTML document as the user downloads when accessing the web site. This means that web scraper has access tp everything, unless the site limits the user.

There is multiple methods to restrict content access for a web scraper. The restrictions includes, require the user to login before accessing the content, using a Captcha to access the content, or by banning the Internet Protocol address. The first restriction can be easily circumvented by making a login request before accessing the resource and then attach the login result, usually a session id in a cookie, to each subsequent request to identify as “logged in”. Captcha can be pretty though to circumvent as they are designed to keep out robots. Banning Internet Protocol can be solved by Internet Protocol rotation, which is a technique used to switch to a new Internet Protocol addresses whenever the site is not reachable by the current one or switch on regular intervals. The reason for getting the Internet Protocol banned is often because the web scraper requested the resources to frequently. By requesting too frequently, the server can be flooded with requests and it can be mistaken for a Denial of Services attack.

Some web sites may use frameworks like React or AngularJS to generate the entire website from Javascript. These sites have minimal HTML defining the site, but relies instead on Javascript to generate the HTML content. The content of these sites will only appear if it is requested by a browser or by other environments that can executes the Javascript code generated by React/AngularJS, and thereby render the result in the browser. Selenium[21] is one of them. It emulates a browser and allows the Javascript to run.

2.2.2 Services

Web scraping is free unless an externally provided web scraping service is used. These services, like Parsehub[22], may have a free pricing tier, but for features like scheduling and Internet Protocol rotation, a subscription is needed.

2.2.3 Anonymous access

Some Web API’s require a form for ”key” to be sent along with every request such that the site knows who’s making the request and that they are allowed to.

Each HTTP request from a client to the server contains some meta information called headers. One of these headers, User-Agent, is used to identify which browser the user uses and subsequently if is it a browser at all. This information can be modified by the client, because it is created and sent from the client, which means that the web scraper, which is running on the client, can pretend to be a browser and access all content a normal user has access to. Other services like Web API’s often requires a form for key to be sent with every request to the server, to identify and authorize the client. This is not needed for a web scraper since it is accessing the content using the same protocol as a normal user. However if the content is behind a Paywall, then the web scraper must include the login credentials or a form for id to show that it is logged in. This would mean that the server can link each request to a certain client.

2.3 RESTApi

Roy Fielding described in his dissertation in 2000, an "architectural style for distributed hypermedia systems"[23] which he called RESTApi(**R**epresenational **S**tate **T**ransfer API). This has become a standard that most web services have implemented. Especially services that are delivering content to other users. REST lets the user extract data in a uniform way, and may require authentication, but often do not. It consist of 6 architectural properties.

Client-Server

Separating user interface from the data storage will improve the portability and scalability .

Stateless

Keep session state on the client and let each request from client to server contain all the information needed.

Cache

Clients can cache responses. These responses must therefore define themselves as cacheable or not to prevent clients from reusing stale data in response to further requests.

Uniform Interface

Is a way to decouple the architecture and contains 4 constraints.

1. **Resource identification in requests** Identify each resource by a Uniform Resource Locater.
2. **Resource manipulation through representations** Any giving representation of a resource is enough information to modify it.
3. **Self-descriptive messages** Each message includes how to use that message. For instance which parser to use to process the message.
4. **Hypermedia as the engine of application state (HATEOAS)** On each request include dynamically generated hyper-links to other available resources.

Layered System

A client cannot tell if it is directly connected to the end server, or and intermediate between itself and the end server.

Code-On-Demand

Is the only optional constraint in REST, and "allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented."[23]

2.4 Chrome extension

A chrome extension[24] is a small software program that can modify and enhance the functionality of the Chrome browser. The easiest way to get an extension is to download it from the official marked place, Chrome webstore.

There is many different use cases for extensions. Examples range from Google translate which enables users to translate marked words or sentences, to Adblock which blocks advertisement on sites while browsing.

2.4.1 Structure

An overview for the architecture is giving in figure 2.1. A Chrome extension is a Zipped bundle of different files. These files define the functionality of the extension, and can be everything from a Javascript file to an image.

manifest.json

A file containing meta-information about the extensions. Examples of meta-information is the plugins name and description or what permissions the plugin requires.

HTML

At least one HTML file is required in the bundle and is "invisible" for the user. The required HTML file is used as a background page where the background scripts live. Additional HTML files can be used to define the user interface. For instance a popup is shown when the extension icon next to the url bar is clicked. This popup can be defined in its own HTML file.

JavaScript

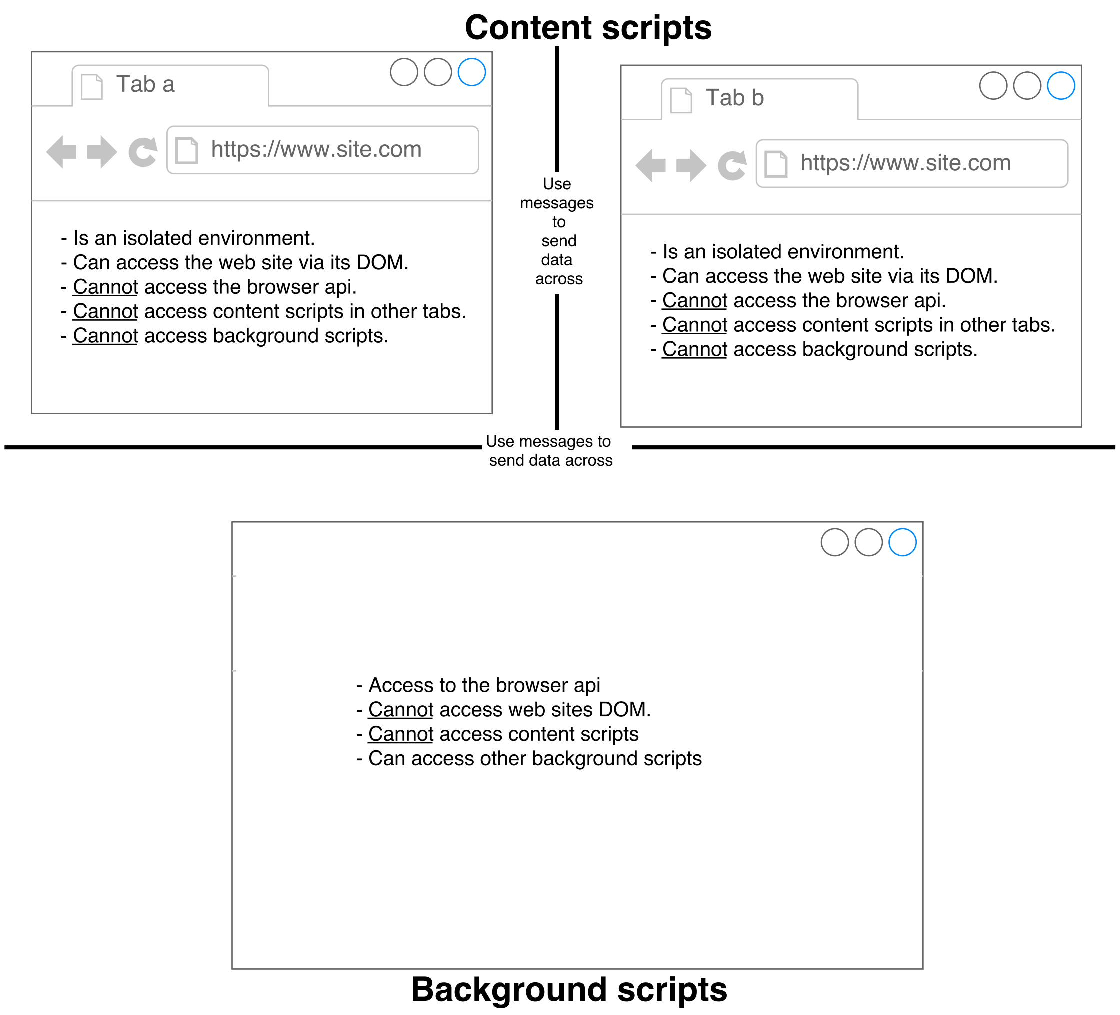
JavaScript files are optional. All javascript files have access to the Chrome api. This API allows the different script to communicate by passing messages. They can also access features like bookmarks, tabs and localstorage within the API. There exists two types of javascript files, background and content scripts.

Background scripts contains code which can control the whole behavior for the extension. This type of code runs continuously from the browser is opened to the browser closes. These scripts cannot interact directly with the different web sites.

Content script is code which is executed in the context of a loaded web site. This means that if the user has two open tabs, then both of them will run the same code files, but independently since these instances share no variables or content. They rely on passing messages to communicate with other content scripts in other tabs or background scripts.

Other files

The bundle can contain other files like CSS to style the different HTML files or images to use as icons, but it is all up to the developer to suit his needs.



Content scripts
Tab a
https://www.site.com
- Is an isolated environment.
- Can access the web site via its DOM.
- Cannot access the browser api.
- Cannot access content scripts in other tabs.
- Cannot access background scripts.
Use messages to send data across
Tab b
https://www.site.com
- Is an isolated environment.
- Can access the web site via its DOM.
- Cannot access the browser api.
- Cannot access content scripts in other tabs.
- Cannot access background scripts.
Use messages to send data across
- Access to the browser api
- Cannot access web sites DOM.
- Cannot access content scripts
- Can access other background scripts
Background scripts

Figure 2.1: Chrome extension architecture

3 Problem description

This chapter defines what the problem is that this thesis is trying to solve and the thesis limitations.

3.1 Tracking changes

Tracking changes allows users to see all changes the journalists have made, both on headlines and article content, such that the correction of erroneous information is visible in case of manipulation of news by deliberately changing content over time. Currently it is hard for a reader to complain on what was an erroneous article if it has been corrected, irrespectively of how long the erroneous article was visible, and irrespectively of how many who have read the erroneous article. Tracking these changes and making them public would also make the journalists more accountable for their work.

3.1.1 Extracting content from articles

Hartly Brody[25] states "companies care way more about maintaining their public-facing visitor website than they do about their structured data feeds". He implies that the most recent data will always be on the web site. Very often sites do not even have data feeds like API or any other means to serve content. It is likely that web scraping (see 2.2) is a good way to retrieve content from sites. This may also be the only way to collect statistics on journalists as stated in the thesis goals 2c.

3.2 Plugin

As mention in the thesis goals (see page 3) a user normally scans for interesting headlines when reading a news site's front page. Each headline has a title, maybe subtitle and an image. That is not much to go on when deciding if this article is interesting or not. In addition, the journalist is sometimes very "creative" with the title or imagery, making it completely misleading. A user could really benefit from an article's metadata and other users feedback, to decide if it is worth a click or not. Instead of relying on the news site itself, which only tries to make money.

3.2.1 Metadata

Piotr, in a blog post on dataedo.com, defines meta data, as "Metadata is simply data about data. It means it is a description and context of the data. It helps to organize, find and understand data"[26]. If metadata can help to understand data, then metadata for an article can help the user to understand the article, which in turn could give the user a larger base to take the decision on whether or not to click the headline.

An articles contains 3 main metadata which can be used to:

Title Compare if the headline and article title deviates in any way.

Timestamps Give an indication on how "recent" the article was published or updated.

Journalist(s) Check if the article is written by a journalist the user dislikes, has as their favourite or any in between.

Additional data, both submitted by other users and aggregated on a server, can also be helpful:

Read time Give an indication on how long it takes to read the article. The user may have limited time at that moment, if in transit on a bus or similar.

Subscription Not everyone has a subscription on each news site. Some only browse the free articles, and not all news sites labels which articles requires subscription.

Summarize User submitted one line summaries. Can give a good indication on what the article is all about.

Word cloud A generated Word cloud of the articles content. Can give the user a quick overview of what the article is all about.

Flagging Let other user label headlines as clickbait or fake news, by submitting reports with a justification as to why that article should be flagged.

These data should be shown to the user for each headline (as explained in the thesis goals 1a).

3.2.2 Feedback

Letting users submit feedback is used to solve the thesis goals 1c, 1d and 2a. This enables users to flag headlines by a certain category or submit summaries of the articles.

Flagging

Flagging articles is a way to warn other users to not read that article. The headline of a flagged article will show when and why this article was flagged and by which category(I.E clickbait or fake news). This will help to solve goal 1c and 2a as flagging could help other users to avoid those headlines.

Summarize

A one line summary is a great way to give the user an understanding of what the article is all about. This is done by other users.

This will help to solve goal 1d and 2a as user submitted summaries can help other users to filter out non relevant headlines.

3.3 Limitations

3.3.1 Supported news sites

This thesis focuses on the Norwegian news sites vg.no, ap.no, db.no, dn.no, nrk.no, bt.no and ba.no (listed in table 3.1). However, the plugin has been designed to be extendable to additional sites, Norwegian or otherwise.

nrk.no is the only site which does not issues a physical copy, but is only online. In addition it is the only state owned news site supported in this thesis. The news sites bt.no and ba.no represents the local news sites, both of which is located in Bergen, Norway. db.no, vg.no, ap.no and dn.no is 4 of the largest national newspapers in Norway, where dn.no relates more to business and ap.no has to some extent local content for Oslo, Norway.

These sites was chosen because they represent both local and national news. The size differ greatly between them(se ap.no vs ba.no in table 3.1) and they are all Norwegian news site, because that is where this thesis is emphasized the most.

Table 3.1: Supported Norwegian news sites.
Online readership figures from 2017[27].
* No data.

News site	Type	Owned by	Circulation
vg.no	National	Schibsted Norge	1 974 000
db.no	National	Aller Media AS	1 166 000
ap.no	National	Schibsted Norge	816 000
dn.no	National	NHST Media Group	334 000
bt.no	Local (Bergen)	Schibsted Norge	174 000
ba.no	Local (Bergen)	A-pressen, Sparebanken Vest, Wimoh Invest	92 000
nrk.no	National	State-owned	*

3.3.2 Client

Google Chrome holds, according to StatCounter[28], the majority of the desktop market by 64.72 % (table A.1), but is not available for mobile or tablet devices (table A.2 and A.3). FireFox is in second place on desktop with 12.21 %, however, FireFox, and also Opera, supports extensions on mobile. According to StatCounter[28] they only hold 5.70 % (Opera) and 0.80 % (FireFox) (table A.2) of the total mobile market share. In the tablet market they are listed under as ”Other” which gives them less den 3.86 % (table A.3) of total market share combined.

The idea behind this thesis extensions is to be used by as many users as possible, and that's why Google Chrome extension was chosen. To support mobile would be very nice, but to reduce 64.72 % coverage down to 12.21 % on desktop just to mobile support from 0 % to 6.05 % is not worth it. An option is to create an extra extension for FireFox or Opera to support mobile and tablet, but it would take too much time maintaining two extensions at the same time and again would not be worth it.

3.3.3 Article coverage

The prototype this thesis created did not cover all articles on all the news sites, which means that the user would not see any information for some headlines and articles. But this do reduce the tracking changes on articles which are not longer on the front page.

Other article formats

Some sites, like dn.no, have multiple formats for their articles. Different formats leads to different layout and HTML structure. This means that each site may need more then one article template to be able to scrape all the articles on that site. This thesis ignores the problem and only use one default template for each news site.

Articles behind a paywall

Articles behind a Paywall are not scraped properly, but are neither ignored. They are set to “requires subscription”, which shows the user that (s)he would need subscription to view this article.

Articles not found on the front page

The server used in this thesis had very little memory and only one CPU, which made the web application really slow when the background task ran the web scraper. Limiting the amount of articles needed to scrape, effectively mitigated the problem. This means that the web scraper only scrapes articles which are currently on the front page.

3.3.4 Word cloud

Each generated Word cloud image has 720p resolution and averages a size of 0.2MB. Upon delivery, the web scraper has scraped more than 62,000 articles. To generate an image for each article would require 12.4GB (0.2 / 1000 * 62,000). Because of limited server space, only the first 5 articles for each news site gets their Word cloud generated. In addition, each time a news site starts to generate its 5 Word clouds, the previous ones are deleted.

4 Design and Solution

This chapter presents the design of the client and server, and how they interact with each other. Further on the chapter explains how the different components on the server is implemented. First, the web scraper is described, and then how the scraper collects data via articles and headlines. Then the "diff" is explained which generates the differences between the different headline and article versions. The next section describes how the background task, which has the responsibility to continuously start the "diff" and web scraper, is implemented. The last section in this chapter explains how the client is implemented.

4.1 Application design

The top level design is a Client Server Model. The server is Django[29] web application which continuously scrapes data from news sites (more on this in 4.1.2) and exposes these data via a RESTApi (chapter 2). The client is implemented as a chrome extensions, and users would have to install it in their browser to use it.

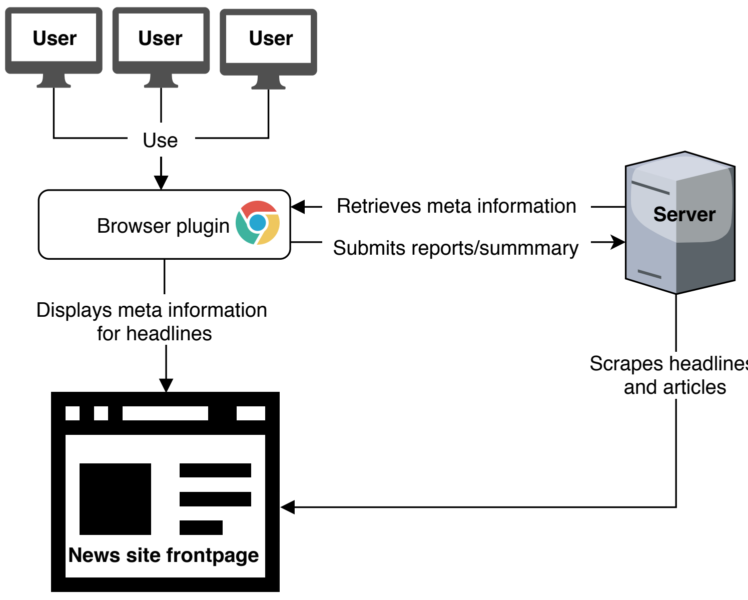

User
User
User
Use
Browser plugin
Retrieves meta information
Submits reports/summmary
Server
Displays meta information for headlines
Scrapes headlines and articles
News site frontpage

Figure 4.1: Architecture for the Client-Server applications

4.1.1 Client design

The client has primarily 8 components split in 2 different layers, where one layer resides in each tab and the other in the background. Storage, ContextMenu, Word Cloud and BackgroundService is operating in the background layer, which means that they cannot talk directly to the other components which lies sand-boxed in each open tab. ArticleInjector, FrontPageInjector, LinkInjector and Modal resides in the tab layer and are called content scripts. These scripts is injected into each tab, making it possible to manipulate the HTML structure of that web site.

Storage
The storage component is in charge of caching all data the plugin receives from the server. The storage itself is a list of storage items. Each storage item has 2 attributes, a unique key to identify it and data which represents its content. The storage data is kept in the browser memory, which means that all the data is lost when the browser is closed. It is possible to persist the storage data to the localstorage, which is still persisted when the browser is closed, but it would contribute to nothing since the data is refreshed so often.

BackgroundService
The backgroundService is responsible for handling all processes done in the background. It is an intermediate (figure 4.2) between the content scripts (scripts running sandboxed in each tab) and either storage or the server, depending on what data is requested. The main responsibilities includes:

1. **Check if current site** is an article, or a front page for a news site.
2. **Submitting** reports and summary to the server.
3. **Retrieve** data from the server.
4. **Distribute** data to other components when requested.
5. **Notifying** the user of events. For instance, if the submission was successful or if some data was inadequate.
6. **Track and cache** the id of the headline last hovered over.
7. **Reloading cache** each x time unit. This is to always have the most recent content updates.

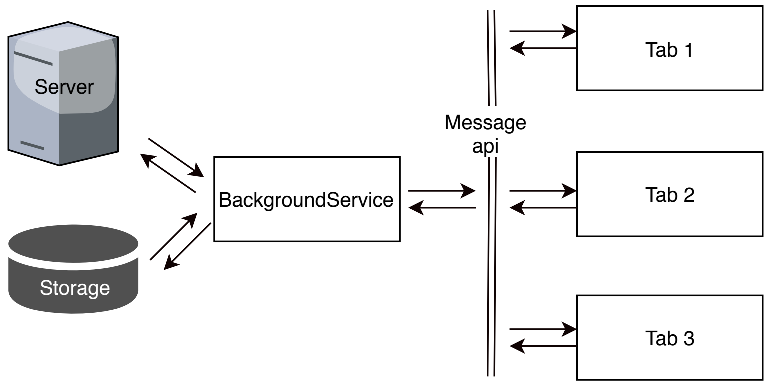


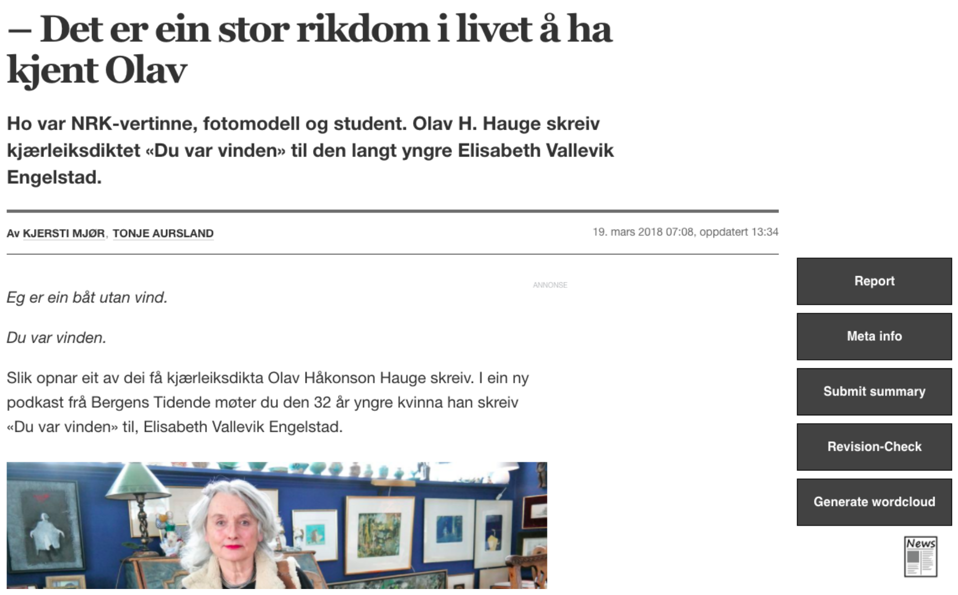
Figure 4.2: BackgroundService data flow

Front page injector
The frontPageInjector has the responsibility to mark each headline on a supported site's front page and show meta information of that headline. The meta information is revealed in the top left corner whenever a supported headline is hovered over. This component is only active if the current url is the front page of a supported site.

LinkInjector
The LinkInjector works like the frontPageInjector, but instead of concentrating on the headlines on a front page, it checks every link (`<a href="example.com"> </a>`) on the site if it is a supported article or not.

Article injector
The articleInjector has the responsibility to show all features and information related to a specific article. If the article is not supported, then it will do nothing. However, if the article is supported, then a large version of the News Enhancer icon is displayed in the bottom right corner (see figure 4.3). The responsibilities are:

1. **Report** Enables the user to report the article under a defined category. This opens the modal component with a form where the user must justify why this article should be reported.
2. **Meta info** Shows all the meta information for this article and its headline.
3. **Submit summary** Enables the user to submit a summary for the current article.
4. **Revision mode** This option is only visible if the article has been updated. It lets the user navigate between the different version of the article. Each version represents the difference between two updates of the article, and marks the text background green if the text is recently added or red if it is removed.
5. **Generate wordcloud** Shows a pre-generated Word cloud of the article text.



– Det er ein stor rikdom i livet å ha kjent Olav

Ho var NRK-vertinne, fotomodell og student. Olav H. Hauge skreiv kjærleiksdiktet «Du var vinden» til den langt yngre Elisabeth Vallevik Engelstad.

Av KJERSTI MJØR, TONJE AURSLAND 19. mars 2018 07:08, oppdatert 13:34

ANNONSE

Eg er ein båt utan vind.

Du var vinden.

Slik opnar eit av dei få kjærleiksdikta Olav Håkonson Hauge skreiv. I ein ny podkast frå Bergens Tidende møter du den 32 år yngre kvinna han skreiv «Du var vinden» til, Elisabeth Vallevik Engelstad.

Figure 4.3: Article feature menu on the right. The article is from www.bt.no

Modal

The modal has the sole responsibility to present information. This can be static information such as Word cloud images or interactive forms for submitting summaries and reports.

Word cloud

The Word cloud component is responsible to generate word clouds and download them. The word cloud is an image that is generated on the server and is presented on the client in a `<img src=" /` via the modal.

Context menu

The context menu is the menu that appears whenever the user right clicks on anything in the browser. Right clicking on links `<a href="example.com"></a>` however, yields an extra menu option to generate a Word cloud. But to actually view the Word cloud the link has to be a supported article. If it is the case, then a Word cloud will show inside the modal, if not the user will be notified that something went wrong.

The context menu class keeps track over which headline was hovered over last. This is because the headline id is included in the report submission to the server for identifying which headline was reported. However, the problem is that the callback function, which is called whenever one of the menu options is clicked, has no way of knowing what headline the user right clicked on. This is solved by sending a message from the content script to the background whenever a headline is hovered over, including the headline's id.

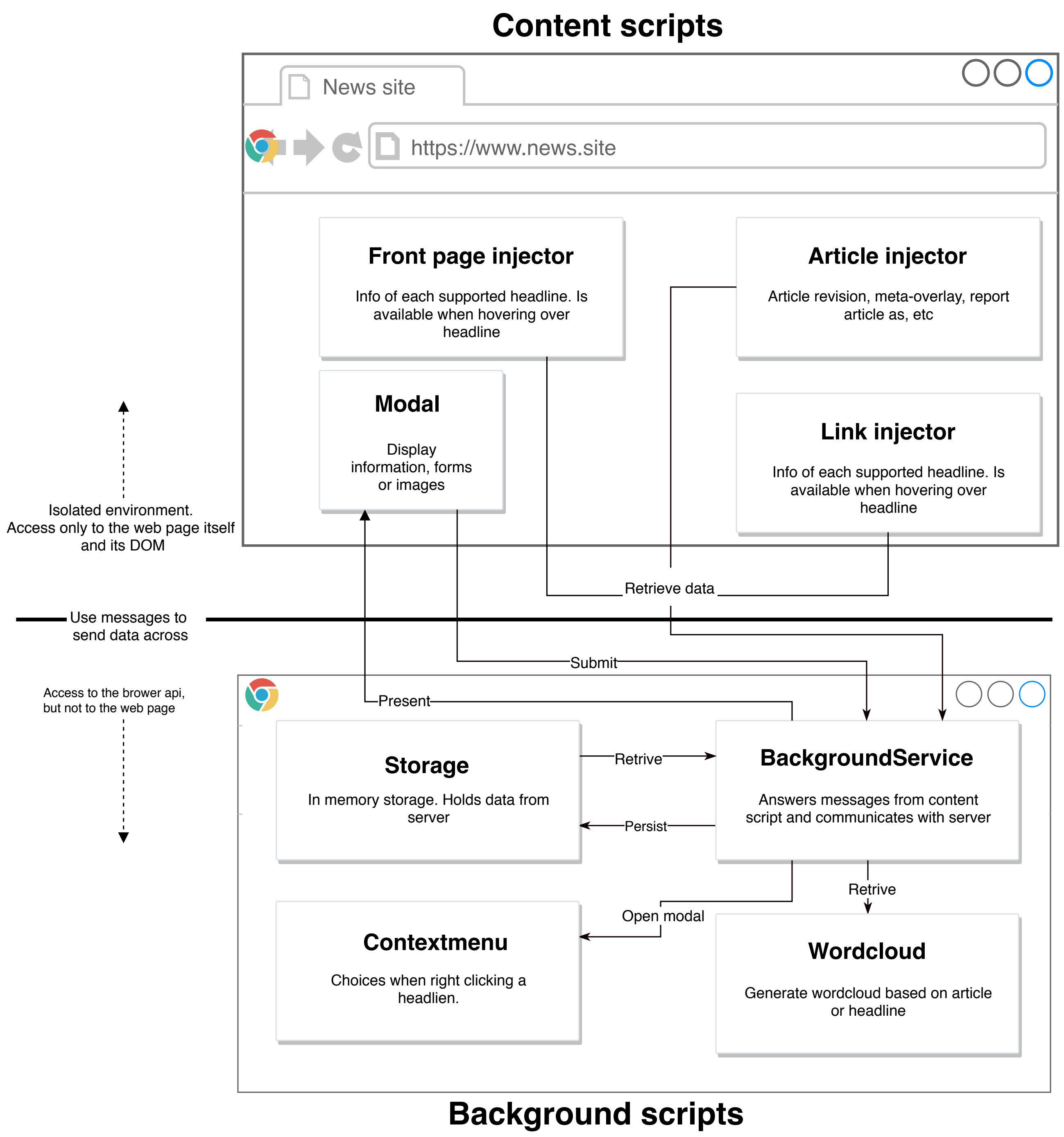

Content scripts
News site
https://www.news.site
Front page injector
Info of each supported headline. Is available when hovering over headline
Article injector
Article revision, meta-overlay, report article as, etc
Modal
Display information, forms or images
Link injector
Info of each supported headline. Is available when hovering over headline
Isolated environment. Access only to the web page itself and its DOM
Retrieve data
Use messages to send data across
Submit
Access to the brower api, but not to the web page
Present
Storage
In memory storage. Holds data from server
Retrive
Persist
BackgroundService
Answers messages from content script and communicates with server
Retrive
Open modal
Contextmenu
Choices when right clicking a headlien.
Wordcloud
Generate wordcloud based on article or headline
Background scripts

Figure 4.4: Client architecture

4.1.2 Server design

The server has primarily 2 responsibilities. The first is to use a web scraping component to retrieve information about headlines and articles. This information is then persisted in the database. The second is to function like a web application that clients can send and retrieve data from. The server design is shown in figure 4.5.

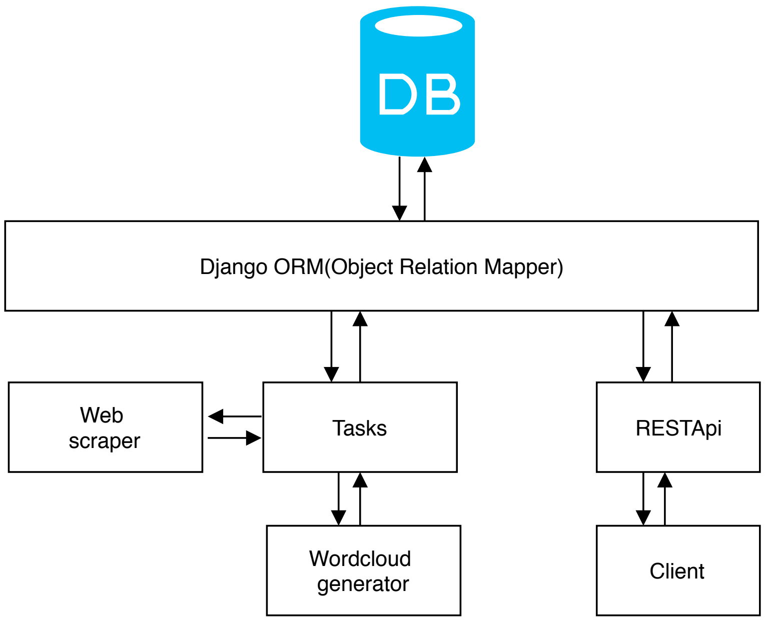


Figure 4.5: Server architecture

Web application

Django web applicaton framework[29] was chosen because it promised rapid development: “Django was designed to help developers take applications from concept to completion as quickly as possible.”[29]. Django also comes with many built in features. A few of them used in this thesis are listed below:

1. **Admin feature** Django[29] has a fully fledge admin site, that let’s you manage users connected to the site and to create, update, edit or delete instances of the models defined in the application. In most frameworks this is something you would have to build yourself, but now you can instead focus 100% on the actual assignment.

2. **ORM** The Django ORM makes it really easy and readable to query the database for data. No need to write specific SQL queries or classes. However, it might not be optimal for more complex queries.

The web-application itself consist of 7 apps where each app has it own set of responsibilities:

1. **Api app** is the only entry point, besides the admin feature, into the application for external connections. It exposes all the data in the database via its RESTApi.

2. **ArticleScraper app** consist of all article related data. It defines the template for scraping an article and holds the actual scraper class which use the template to find the requested elements in an article. In addition, it contains the background task that will run the article scraper.

3. **HeadlineScraper app** consist of all headline related data. It defines the template for scraping a headline and holds the actual scraper class which use the template to find all the headlines on a news site's front page. In addition, it contains the background task that will run the headline scraper.

4. **Scraper app** consist of all news site related data. It defines the base web scraper and initiates the background tasks to scrape a site's headlines and articles.

5. **Differ app** consist of a class which creates HTML file containing the difference between two texts or HTML files.

6. **Submission app** validates all forms the user sends inn.

7. **Word cloud generator app** generates a Word cloud based on an article, a headline or a list of headlines. Also contains the task which creates word clouds for each new or updated article.

Tasks

Celery is, according to their site, "an asynchronous task job queue based on distributed message passing"[30]. This means that Celery can be used to start other scripts in the background. To start a background tasks Celery needs two components (worker and beat), a message broker and a task. The task defines what work should be done and when. It is a function annotated with what type of task it is and how often it should run.

```python
@periodic_task(run_every=timedelta(minutes=20),name='Scrape')
def my_task():
    print("This task is run every 20 minutes")
```

Listing 2: Celery task definition

A task is put into the message broker's task queue whenever it is due. This queue holds the backlog of task, tasks which has not yet been started. Redis[31], an in-memory data structure that can be used as a database, cache or message broker, was used as the message broker because it is really easy to install and use. However, it could easily be replaced by other message brokers like RabbitMQ or AMPQ.

It is the two components that does the heavy lifting. The worker component is, as the name suggest, the one who does the actual work. It invokes the task, however, it is the beat that decides when the task should be invoked. The beat component is a scheduler that appends the task to the message broke's task queue whenever it is due. Available workers is continuously executing tasks in the task queue.

Web scraping

The web scraper has two different instances. One for scraping headlines and one for scraping articles. Both scrapers relies on predefined templates that defines what information to extract, and from where inside the HTML document. These classes are in their respective folders inside the web application, but is not initiated by the web application. The web application has no way of starting regular background tasks, instead Celery[30] is used to start the web scrapers.

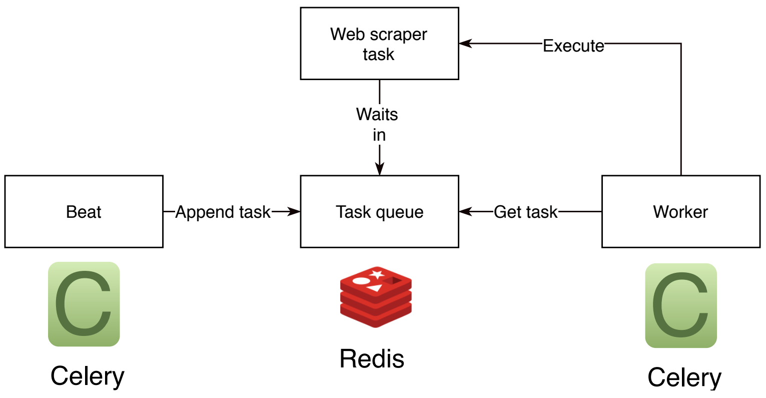


Figure 4.6: Background service architecture

Word cloud

A Word cloud is an image of words shown horizontally and vertically, with words having different sizes according to their importance. The words are densely packed and may be of different colors. A Word cloud is meant to quickly communicate the essence of one or several pieces of natural language text. A Python library called word_cloud[32] was used to generate the Word cloud images. This library takes a list of words and creates an image containing the words from the list. The size of each word depends on the number of occurrences of the word in the text. The more occurrences, the larger the word size. It is also possible to define a list of stop words, which is words the Word cloud generator should ignore and not render in the final image.

In an article there is words that is less important then others. For instance words that does not describe anything if they are isolated like "is", "at", "I". These words are called stop words and is needed to create grammatically correct sentences, but occur very often and do not individually carry much information. The list of stop words used in this thesis was the result by concatenating three Norwegian stop words lists found on the web [33], [34] and [35].

4.2 How the Web scraping extracts data

There are two web scrapers implemented in this thesis. One for extracting headlines from a news site's front page, and one for an article. Both of whom inherits from a "base scraper" that defines methods used in both scrapers. In addition, both scrapers also relies on the use of a general parser that use templates to define where in a HTML document to extract what, instead a hard coded parser for each site.

4.2.1 Base scraper

The base scraper contains 4 methods:

1. **Download** Downloads the requested website and return it as a HTML document.
2. **Scrape** Uses BS4[19] to scrape content from the downloaded web site. It is an empty abstract method that all inherited classes must implement.
3. **Get text** Extracts all text from the current HTML element. Includes text in its child elements.
4. **Get text with no children** Extracts all text from the current HTML element. Excludes text in its child elements.
5. **Parse date** Date is represented very differently across the web, even though international formats exist, see ISO 8601. This method tries to do a catch all and parse every format which the scraper has encountered.

BS4

Scraping the content of a website is pretty easy when using the Python library BS4[19]. The library takes a HTML document as input and parses it into a tree data structure. Information can be extracted from the tree by queries. The different types of queries available is by tag (via `find` and `find_all`) and CSS selector's (via `select_one` and `select`) .

It turns out that CSS selector's was more efficient to use then tags. Not necessarily by speed, but by storage space. `select_one` only requires one argument to specifying a path to an element. `find` however, requires one for tag, two when specifying class for tag or at least three when finding tag with attribute(s). This means that `select_one` requires less space, and fewer fields in the template.

`select_one` is also easier to use when searching for nested tags then `find` . Again this is because all the search parameters fits into a single argument instead of chaining them (as shown below in 3).

```python
# Simple tag only
soup.find('div')
soup.select_one('div')

# With class
soup.find('div', class_='some-class')
soup.select_one('div.some-class')

# With attribute
soup.find('div', {'attribute': 'value'})
soup.select_one('div[attribute="value"]')

# Chaining, finding descendant
soup.find('div').find('span').find('a')
soup.select_one('div > span > a')
```

Listing 3: Select one vs Find syntax

Parsing date

Date and time is represented very differently across the news sites (table 4.1). Some like nrk.no don't specify when the article is published or updated, but gives a general time i.e x hours or minutes ago. Others like db.no, dn.no and ba.no mix text and special characters in with the date and time. db.no and ba.no also use Norwegian words for month, which is understandable since the articles is written in Norwegian, but this means that it has to be translated into English such that the datetime parser can understand which month it is. In addition, all the supported news sites formats the date time this way to make it more human readable, but it also makes it more difficult to support a wide range of formats.

A solution could be to have a date and time template per news site which tells what format to extract the date and time from, or which text and characters to strip away to only remain with the date and time. This would also increase the scraping time, because iterating a string and remove contents or translating words is pretty expansive calls, and each article has a published and maybe an updated time, so it has to do this twice per site. Another solution, though very naive, would be to remove known words like "kl.", "Publisert", etc and trim the month down to the 3 first chars (since almost all months has the same abbreviation in Norwegian and English). Then only test if its "des" (Norwegian abbreviation for December), then it must be replaced with "dec". Then try to parse the date.

Luckily most of the sites includes a datetime attribute on the HTML element displaying the published date (table 4.2). This attribute, at least for the sites in this thesis, is also formatted using the standard ISO 8601. This means it is possible to retrieve date and times in the same format and even with the same timezone (ISO 8601 use UTC as default).

The solution used in this thesis is to add datetime attribute field in the article template, which specifies the name of the attribute that contains the ISO 8601 formatted datetime. If this attribute or datetime field in the template is empty, then run the naive solution described above.

Table 4.1: Datetime representation in text

News site	Rendered	ISO 8601	Timezone
Nrk.no	Publisert idag, for 3 timer siden	No	No
db.no	25. JANUAR 2018 KL. 13.43	No	UTC+1
aftenposten.no	24.jan.2018 19:15	No	UTC+1
bt.no	25. jan. 2018 13:22	No	UTC+1
ba.no	25. januar 2018, kl. 13:54	No	UTC+1
vg.no	25.01.18 13:21	No	UTC+1
dn.no	Publisert: 25.01.2018 10:00	No	UTC+1

Table 4.2: Datetime representation in html tag attributes

News site	Rendered	ISO 8601	Timezone
nrk.no	2018-01-25T11:32:02+01:00	Yes	UTC+1
db.no	2018-01-25T12:43:11.000Z	Yes	UTC
aftenposten.no	2018-01-24T18:15:00Z	Yes	UTC
bt.no	2018-01-25T12:22:37Z	Yes	UTC
ba.no	2018-01-25T13:54:13.000+0100	Yes	UTC+1
vg.no	-	-	-
dn.no	-	-	-

4.2.2 Headline scraper

All of the supported site's headlines follows the same type of markup, as shown in figure 4. They have a specific tag or CSS class that encapsulates a headline and the headline itself contains one `<a href="newsSite.com"><a/>` with the article's URL, a title and maybe a subtitle. The CSS selector path to each of these elements is defined in the headline template for each site.

```html
<article id="article_x" class="preview columns lg-md-sm-12" role="article">
    <a href="https://articleUrl">
            <img data-srcset="https://imgUrl" src="https://imgUrl">
        <div class="article-preview-text">
            <h1 class="headline">
                Headline title
            </h1>
        </div>
    </a>
</article>
```

Listing 4: Simplified headline markup from db.no

The scraping flow, as presented in figure 4.7, works by:

1. **Init scraper** Download and parse the front page of a news site.
2. **Find headlines** Find all headlines on a front page by the CSS selector that encapsulates each headline (`"article [role='article']"` would be the CSS selector for code example 4).
3. **Remove unwanted content** Some sites pose advertisement as headlines. That is unwanted content. The headline template contains a list of CSS selector which identifies unwanted content. All of these are removed in this step.
4. **Extract data** Tries to extract title, article URL and, if it exists, the subtitle from the headline. The headline is ignored if no title or URL is found.
5. **Generate additional objects** In addition, to the data extracted in the latest step, the scraper aggregates new data.

   Headline Contains which news site the headline belongs to and the URL to the article.

   Rank Represents the headlines position on the front page as a digit. The lower digit, the closer to the top was the headline (I.E a headline with rank 1 is the top headline on the front page). This actually allows us to track the headlines life-cycle on the front page.

   Revision Contains the data that can be updated by the news site which in this case is the title, subtitle and which version this revision is (I.E if it is not updated, then the version is 1. But for each time it is updated the version is incremented by one.).

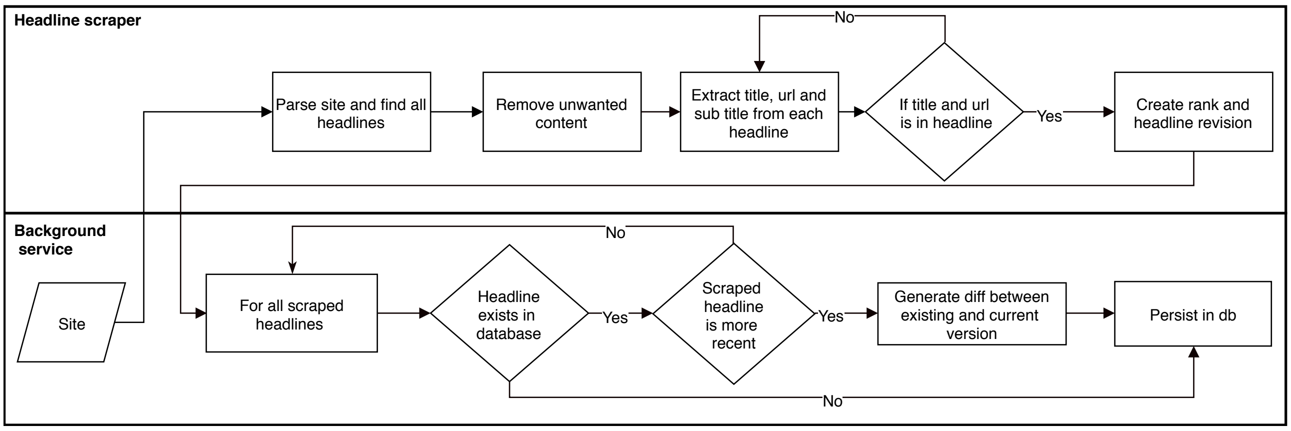

Headline scraper
Parse site and find all headlines
Remove unwanted content
Extract title, url and sub title from each headline
If title and url is in headline
No
Yes
Create rank and headline revision
Background service
Site
For all scraped headlines
Headline exists in database
Yes
No
Scraped headline is more recent
No
Yes
Generate diff between existing and current version
Persist in db

Figure 4.7: Flowchart for the headline scraper

4.2.3 Article scraper

An article's website is encapsulated by a tag or a CSS in the same way as each headline on a news site's front page. The article scraper use an article template, which contains multiple CSS selectors, to define where in that article to extract what.

The flow of the article scraper is:

1. **Init scraper** Download and parse the article.
2. **Extract metadata** Tries to extracts the title, journalist(s), published datetime and updated datetime.
3. **Words** Counts all the words in the article.
4. **Images** Retrieves all the images in the article.
5. **Subscription** All of the supported site's subscription articles has a large "block", with information on how to buy a subscription, that seems to hide all article content

except the first 4 lines. The CSS selector to this “block” is defined in the article template, and if the “block” is found when scraping, then a subscription flag is set to true, to indicate that users need subscription for this article.

7. **Category** Decides the category for the article. The categories are:
   - **Article** An actual article.
   - **Video** The article is actually just a video.
   - **External** The Article resides on a website outside the domain of the news site, which often Native ads does (I.E the newsite is `example.com`, but article link to `notOnExample.com`).
   - **Feed** A type of article, where multiple articles is listed vertically on a time-line.

8. **Generate additional objects** In addition, to the data extracted in the latest step, the scraper aggregates some data and persist it alongside the article.
   - **Article** Contains which news site the headline belongs to and the URL to the article.
   - **Revision** Contains the data that can be updated by the news site, which is the title and subtitle. The revision also contains the revision version as the headline revision.
   - **Journalist** Process all journalists to a new list of Journalists entities, which contains the first name, last name and news site affiliation.
   - **Images** Process all images to a new list of Article Image entities which contains the caption, photograph and link to the image.

4.3 Diff generation

An author at Atlassian’s blog, Giancarlo Lionetti, states that the purpose of a diff “is to take two versions of a file as input, and output a hunk of text that tells you how to change the first file into the second file”[36]. This is very much used in version control system, as he later describes, but can be used for any types of files or texts. The result of the diff between two files is the union between them, but with each line that differ from each other marked. The marking indicates whether or not the line has been edited between the oldest version and the recent one. The unified diff format, as Giancarlo mentions, uses `---` to mark text that has been deleted and `+++` to mark text that has been inserted.

4.3.1 Algorithm

Google built, in 2006 to power Google Docs, a library called “Diff Match Patch” which offers “robust algorithms to perform the operations required for synchronizing plain text”[37]. The library contains 3 features (diff, match, patch) where the diff feature is used in this thesis.

Diff Compare two blocks of plain text and efficiently return a list of differences.

Match Given a search string, find its best fuzzy match in a block of plain text.

Patch Apply a list of patches onto plain text.

The list of differences is a list of tuples (operation, length) where the operation is '-' for removed text, '+' for added text and '=' for text that have not changed, and the length is how many of the next characters is in the operation. An example is when computing the diff between `Hello stranger. How are you?` and `Hello. How old are you?` would yield the result list
`[('=', 5), ('-', 9), ('=', 6), ('+', 4), ('=', 8)]`. The algorithm shows that both text is equal up to the fifth character, but then the next 9 characters is removed. Again the next six characters is equal, before 4 new one is added, and then the rest of the string is equal.

The library used in this thesis is a Python wrapper[38] for Google's C++ version. The reason why the wrapper was used and not Google's own Python implementation is that C++ runs much faster then Python code, at least according to the author behind it.

Implemented Differ class

Differ is the implemented diff class, and it has two main functions that either finds the diff between two texts(`create_diff_of_text`)
or two HTML files(`create_diff_of_html`). Both functions function the same way by using the list of differences and wraps each modified section of the text inside `<span class="inserted_text">changes</span>` when the text is inserted or `<span class="deleted_text">changes</span>` if it has been deleted. This enables the whole modified text section to be highlighted by a color instead of having a prefix. This is to increase the readability. The highlighted color for inserted text is green and red for deleted text. In addition, all diffs are persisted as a `.html` file on disc, with the path to the different types of diffs defined in the web application's setting file.

How to generate a headline diff

Generating diff for the headline title and headline subtitle was done by using the differ's `create_diff_of_text` function. This functions function exactly as described in the `Hello stranger` example.

How to generate an Article diff

Unlike the headline, the article diff compares 2 different HTML nodes with the article content. This means that it is not only text that is inserted into the diff_math_patch algorithm, but also HTML nodes encapsulating the text. The diff_match_patch[37] does not work on HTML with respect to the text only. It will mark changes for the HTML attributes and in the document structure. This is not good enough when one is only after text differences. This is solve by using `create_diff_of_html` method, which functions exactly like `create_diff_of_text` but with every inserted or removed operation the differ checks whether or not it is inside a HTML tag (`<tag attribute="is inside"/>` or `<tag attribute="is inside">not inside</tag>`). The function used to check if the current change is inside a tag or not is `is_in_tag_definition` and its implementation is very naive, but works in this scenario. It functions by iterating backwards in the text from where the changes were made and checking each character if it is a `>` or a `<`. The changes is not inside a tag if a `>` is found before `<`.

4.4 How background tasks work

Each 20 minute the Celery[30] beat appends a background task into the message broker's task queue. This is quickly picked up by an available worker, which then executes this task. The task itself is a function that fetches a list of all the active supported site's and executes 3 other tasks (HeadlineTask, ArticleTask and WordCloudTask) with each news site as input.

4.4.1 Headline task

The headline task is divided into 3 parts (scrape front page, process scraping result and generate diff) and is run for each news site.

Scraping the front page

Scrape headlines from the front page of a news site by using its headline template (as described in 4.2.2 on page 31).

Processing scraping result

Process the result from scraping the front page in 4 steps. The result is a triple (simple data structure with 3 parts) that contain a list of headlines, revision and rank.

1. **Find headline id** The article URL cannot be used as an unique id because the URL is not always a fixed string. Therefore a part of the article URL is used as the id. The first processing step is to extract this from the URL (This is discussed more in detail in 4.5.1 on page 38).

2. **Diff** If the headline exists, but the title or subtitle has been updated, then a diff is generated and persisted.

3. **Database** Persist all scraper generated objects in the database (HeadlineRevision, Headline).

4. **Rank** Updates the headlines rank (the new headline location on the front page).

Generate headline diff

If the processing yielded any result and some of the headlines had updated their title or subtitle, then a diff between the current and most recent version is generated (as described in 4.3.1 on page 35).

4.4.2 Article task

The article task is divided into 3 parts (scrape article, process scraping result and generate diff) and is run for each news site:

Scraping

Scrape an article on a news site's using its article template (as described in 4.2.3 on page 33).

Processing scraped article

Process the article in 5 steps by creating new objects for some of the metadata, generating diff and persisting the result.

1. **Result** Check if the scraping yielded any result.
2. **Database** Check if the article is update and if so create and persist a new revision for the updates.
3. **File** Persist the website as a file on disk.
4. **Images** Persist the image link with photographer and caption.
5. **Journalists** Persist the journalist and associate (s)he with a news site.

Generate article diff

Generates a diff between the current and the most recent version (as described in 4.3.1 on page 35), but only if the article was updated. There exists a method to check if 2 article versions is equal or not, but it always returns true. In reality it should extract all text from both versions and compare it, but for this prototype it is decided to only check if the article's updated datetime was more recent then the previous one. This was because comparing large texts can be very slow, and the background task used enough time and process as it is.

4.4.3 Word cloud task

The Word cloud task generates two different Word clouds. One for the front page of a news sites and one for each of its articles.

Front page

The headlines which currently resides on the front page for a news site is used to generate a front page Word cloud. The title for each headline is fetched from the database and appended to a list of headline titles. These titles is what the Word cloud is generated from.

Article
The article Word cloud is generated from the articles text content. All article websites is persisted on disk when the articles scraper request the website. This document is fed into the article scraper, which then extracts all the text content from it.

Persist files
All Word clouds are persisted on disc in the Django[29] web applications media folder. This is because this folder is accessible from outside the server. Each news site has their own directory with their Word clouds, and that path is defined in the web applications settings file.

4.5 Plugin

The Chrome plugin works on any website which contains links to the website of previously scraped articles. But the main focus has been on the front page of news sites and their articles. The plugin works by identifying headlines and articles and finding them on the server, however, this is not as straight forward as it seems.

4.5.1 Identifying headlines and articles

There is a need to identify headlines and articles in the client plugin on several occasions.

Front page Match all the headlines with the downloaded ones.

In an article Matching the correct article with information in the database.

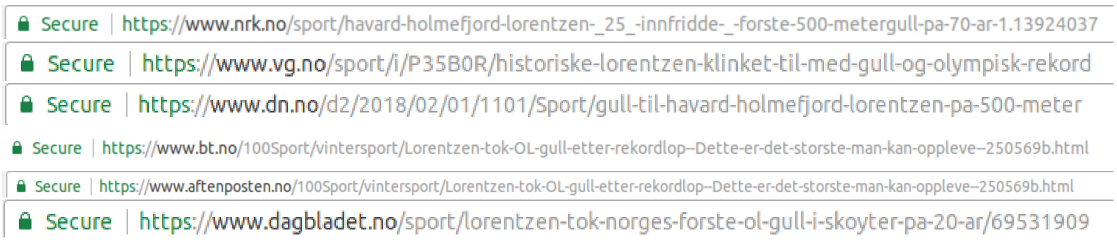
Every page Trying to match all `<a href="https://example.com"></a>` elements on the page with headlines.

Identifying the headlines and articles should be pretty simple. Find something unique for that article or headline and use it. The part of an article which is most unique is the URL, and using it as an identifier, works really great the first time an article is scraped. But what if a later scrape reveals that the article's URL is slightly altered, but still redirects back to the original article? Then the scraped article is seen as a completely new article, even though earlier revision of it resides in the database. This will not effect the metadata displayed by the plugin, because they reflects the latest revision, but Revision time-line mode is severely affected. It will not get the earlier revisions, and also the "previous article version" will no longer get a match in the client plugin because that URL is no longer in use at the news site.

The reason why the URL can be slight altered is because most of the sites includes the title in the URL to make it more readable, both to humans and to robots (see figure 4.8). This means that whenever the article title gets updated, then the updated version is created as a "new" article in the database. This is also why CrowdsouRS[12] would not work properly with articles. Because votes meant for the same headline would be registered on different URL's, if the title were updated. This would be registered as different web page, when in reality were same web page.

Instead it is possible to use a portion of the URL as the identifier. For most of the sites, as long as the id is correct, the title can be edited and even omitted, and it would still redirect to the correct article (This does not include DN.no, but all of the other sites used in this thesis). Which means that the id part of the URL is much more safe to use as an identifier then the whole URL. Below is an example of 3 different variations the author has come across during this thesis.

1. https://example.com/optional/../**id**/title-with-dash
2. https://example.com/optional/../title-with-dash/**id**
3. https://example.com/optional/../title-with-dash-**id**



Secure | https://www.nrk.no/sport/havard-holmefjord-lorentzen-_25_-innfridde-_-forste-500-metergull-pa-70-ar-1.13924037

Secure | https://www.vg.no/sport/i/P35B0R/historiske-lorentzen-klinket-til-med-gull-og-olympisk-rekord

Secure | https://www.dn.no/d2/2018/02/01/1101/Sport/gull-til-havard-holmefjord-lorentzen-pa-500-meter

Secure | https://www.bt.no/100Sport/vintersport/Lorentzen-tok-OL-gull-etter-rekordlop--Dette-er-det-storste-man-kan-oppleve--250569b.html

Secure | https://www.aftenposten.no/100Sport/vintersport/Lorentzen-tok-OL-gull-etter-rekordlop--Dette-er-det-storste-man-kan-oppleve--250569b.html

Secure | https://www.dagbladet.no/sport/lorentzen-tok-norges-forste-ol-gull-i-skoyter-pa-20-ar/69531909

Figure 4.8: Example of url variation across the sites on the same news story.

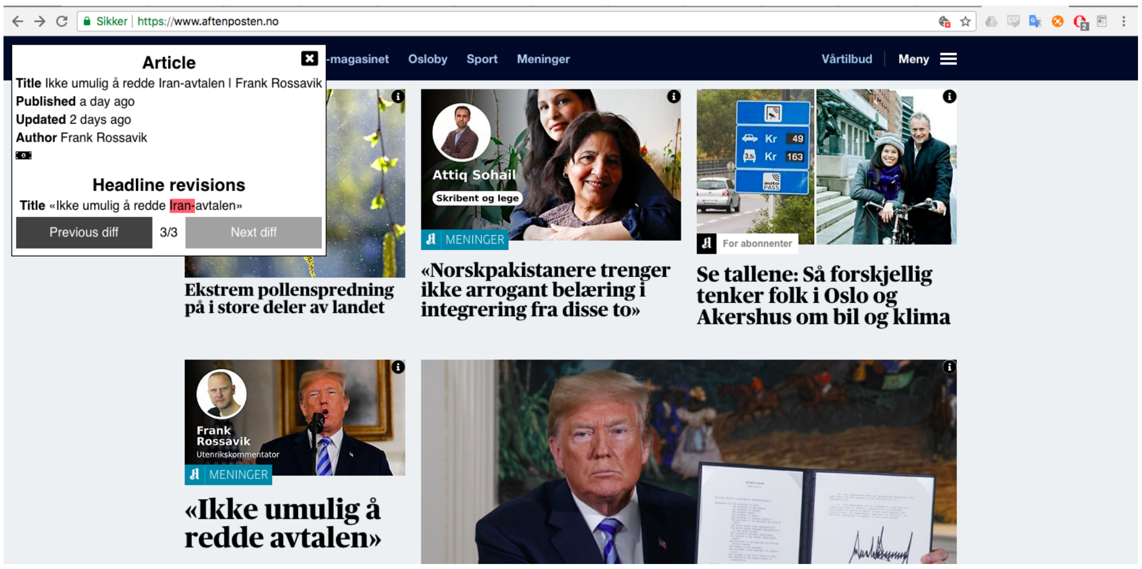
The only drawback is that it has to be specified for each site where in the URL, to find the id section, and some sites may use multiple schemes. Like bt.no and aftenposten.no, they both use variation 1 and 3. This means that each site should be able to include multiple url-templates to specify where in the URL to extract the id. Now the url position refers to a position in a list when the url is splitted on `/` (`ex.no/op/t-w-d` → `["ex.no", "op", "t-w-d"]`), and counting backwards. However, the URL "ex.no/op/t-w-d-**id**" would be valid for variation 1 and 3 in figure 4.8. The difference being only one of the returned words could be used as an id. The other could be very common word, and would lead to duplicates. One could set a fixed length on the id, since each news site seems to use a fixed length, but it could still lead to duplicates. This thesis always tries variation 3 first, and if the id portion of the url is not of a specific length (is specified in the url template), then it tries with another template. This could lead to duplicates, but it is rare to find words of length between 6 and 8 (which are the id length used by the sites in this thesis) as the last word in a sentence.

This thesis uses the id part of the URL to identify each article and headline.

4.5.2 Front page

Two features are provided by the client to help the user to get a quick overview of the headlines. The first feature is an information box which contains a lot of information about the article, both collected by web scraping and submitted by other users. In addition, all headlines has a “view Word cloud” option when the user right clicks the headline which will display a Word cloud.

These features are primary intended for the front page of a news site, but works for all websites that has at least one link to a supported article, and has an article that is previously scraped by the web scraper (see 4.2.3 on page 33). These are marked by a black circle with an “i” in the center, which resides at the right most side of the link (as shown in fig 4.9).



Article

Title Ikke umulig å redde Iran-avtalen | Frank Rossavik
Published a day ago
Updated 2 days ago
Author Frank Rossavik

Headline revisions

Title «Ikke umulig å redde Iran-avtalen»

Previous diff 3/3 Next diff

Sikker | https://www.aftenposten.no
-magasinet
Osloby
Sport
Meninger
Vårtilbud
Meny
Attiq Sohail
Skribent og lege
MENINGER
For abonnenter
Ekstrem pollenspredning på i store deler av landet
«Norskpakistanere trenger ikke arrogant belæring i integrering fra disse to»
Se tallene: Så forskjellig tenker folk i Oslo og Akershus om bil og klima
Frank Rossavik
Utenrikskommentator
MENINGER
«Ikke umulig å redde avtalen»

Figure 4.9: Front page of aftenposten.no. Supported headlines is marked in the top right corner by a black circle. A headline’s information box is shown in the left corner.

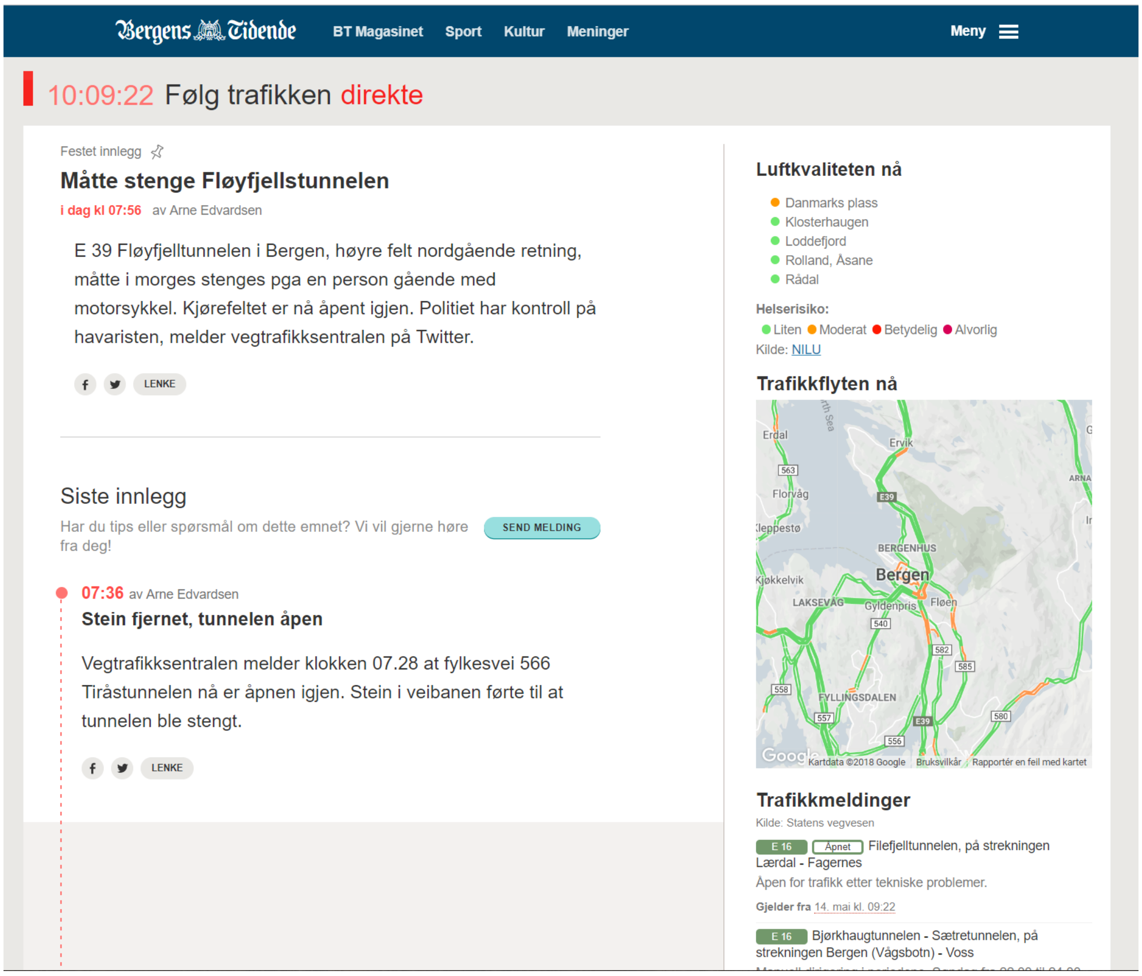
Information box

The information box is a rectangular box, as shown in fig 4.9, which appears in the browser’s top left corner whenever the user hovers the mouse over a supported headline. There exists 4 different categories for an article (video, feed, article or external). The data displayed in the information box depends of the article category:

1. **Video** Refers to an article with only a video. The information box will then have only a title with the word “Video”.

2. **Feed** Refers to a type of article where there is multiple articles listed vertically along time-line (as shown in figure 4.10). The information box will then have only a title with the word “Not an article”.

3. **External** Refers to an article which is not on the news sites domain. I.E if the news site is at `newsSite.com`, then the article is on `example.com/article`. “Native



10:09:22 Følg trafikken direkte

Festet innlegg

Måtte stenge Fløyfjellstunnelen

i dag kl 07:56 av Arne Edvardsen

E 39 Fløyfjelltunnelen i Bergen, høyre felt nordgående retning, måtte i morges stenges pga en person gående med motorsykkel. Kjørefeltet er nå åpent igjen. Politiet har kontroll på havaristen, melder vegtrafikksentralen på Twitter.

LENKE

Siste innlegg

Har du tips eller spørsmål om dette emnet? Vi vil gjerne høre fra deg!

SEND MELDING

07:36 av Arne Edvardsen

Stein fjernet, tunnelen åpen

Vegtrafikksentralen melder klokken 07.28 at fylkesvei 566 Tiråstunnelen nå er åpnen igjen. Stein i veibanen førte til at tunnelen ble stengt.

LENKE

Luftkvaliteten nå

- Danmarks plass
- Klosterhaugen
- Loddefjord
- Rolland, Åsane
- Rådal

Helserisiko:
● Liten ● Moderat ● Betydelig ● Alvorlig
Kilde: NILU

Trafikkflyten nå

Trafikkmeldinger

Kilde: Statens vegvesen

E 16 Åpnet Filefjelltunnelen, på strekningen Lærdal - Fagernes

Åpen for trafikk etter tekniske problemer.

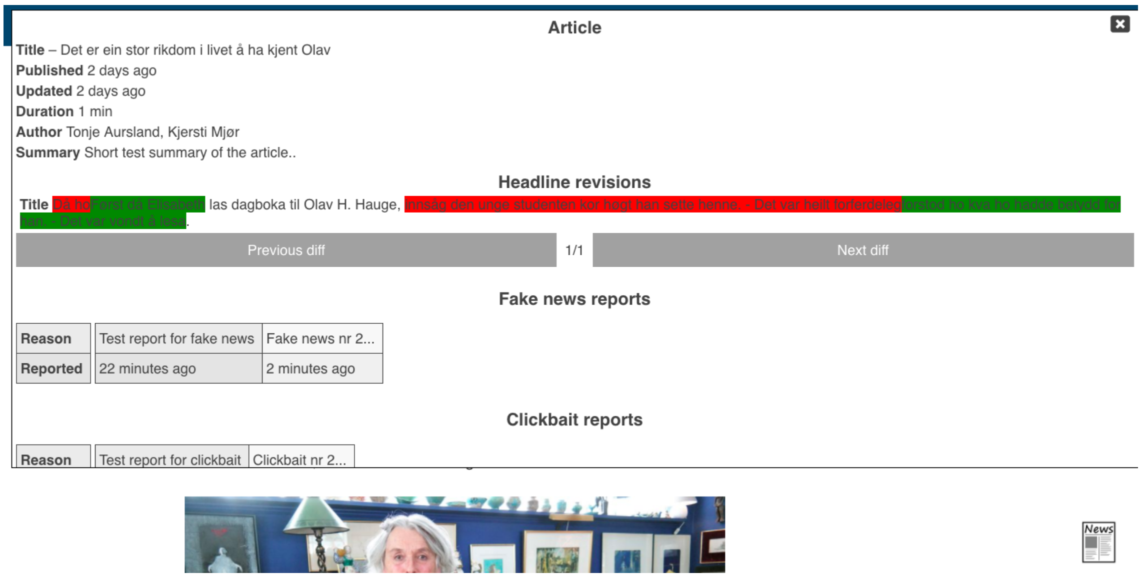
Gjelder fra 14. mai kl. 09:22

E 16 Bjørkhaugtunnelen - Sætretunnelen, på strekningen Bergen (Vågsbotn) - Voss

Figure 4.10: Example of the article type "Feed" from bt.no.

ads" is often marked as external. The information box will then display the domain name to show which website the link is to.

4. **Article** An actual article. The information box can contain 4 different parts of data for these exists:
   1. **Article metadata** Data that is extracted and aggregated by scraping the article (see information box in fig 4.9 under "Article").
   2. **Summary** User submitted one line summary of the content.
   3. **Flagging** User submitted reports which "flags" the headline as clickbait or fake news. Is a horizontal list of user reports which contain when the user reported it, and why (as shown in figure 4.11).



Article

Title – Det er ein stor rikdom i livet å ha kjent Olav
Published 2 days ago
Updated 2 days ago
Duration 1 min
Author Tonje Aursland, Kjersti Mjør
Summary Short test summary of the article..

Headline revisions

Title Då hoFørst då Elisabeth las dagboka til Olav H. Hauge, innsåg den unge studenten kor høgt han sette henne. - Det var heilt forferdelegforstod ho kva ho hadde betydd for han. - Det var vondt å lesa.

Previous diff 1/1 Next diff

Fake news reports

Reason	Test report for fake news	Fake news nr 2...
Reported	22 minutes ago	2 minutes ago

Clickbait reports

Reason	Test report for clickbait	Clickbait nr 2...

Figure 4.11: User interface for the information box containing metadata

These reports also changes the supported headline indication based on the number of reports. It operates in 4 levels where each level can only be reported by a specific amount, before the article enters the next level. These levels is adjustable in the database via the Django[29] admin feature. The levels used under development is in parentheses.

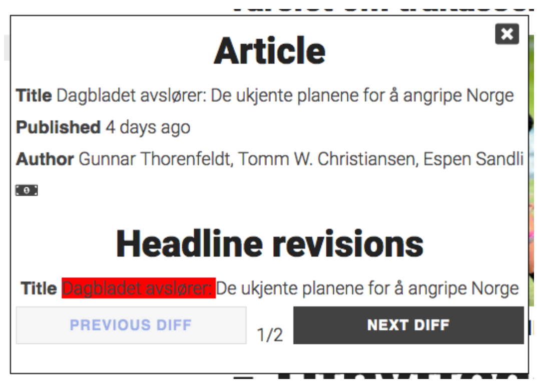
Black None have reported the article (0).

Yellow Low amount of reports (1).

Orange Medium amount of reports (2).

Red High amount of reports (more than 3).

4. **Headline revision** Enables the user to view all the updates done to the headline title and subtitle. The user can navigate forward and backwards between the different versions. All inserted text between the versions is marked with a green background and the removed text with a red one.


Article

Title Dagbladet avslører: De ukjente planene for å angripe Norge

Published 4 days ago

Author Gunnar Thorenfeldt, Tomm W. Christiansen, Espen Sandli

Headline revisions

Title Dagbladet avslører: De ukjente planene for å angripe Norge

Figure 4.12: User interface for navigating and viewing headline revision.

Word cloud

The user can right click any link and click on a custom option, which comes from the plugin. This option is called "Word cloud" and is marked by the NewsEnhancer icon (as shown in figure 4.13). The "Word cloud" itself appears in a modal, but only if the user right clicks a supported headline.

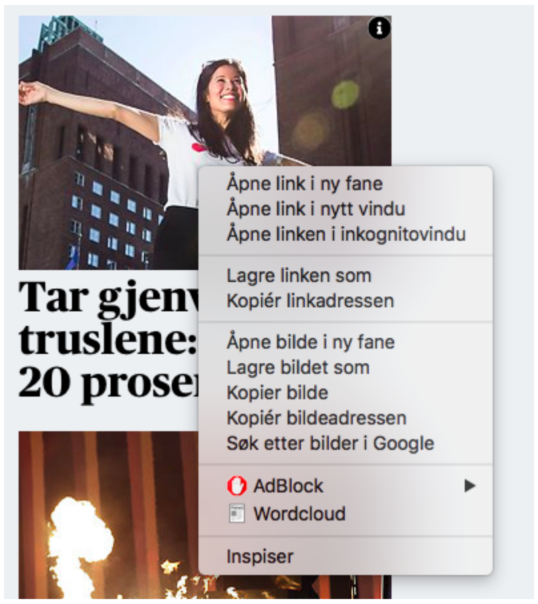


Figure 4.13: Word Cloud context menu option.

However, to generate a Word cloud for a news site's front page, the user would have to click on the NewsEnhancer icon next to the browsers URL field (as shown in figure 4.14).

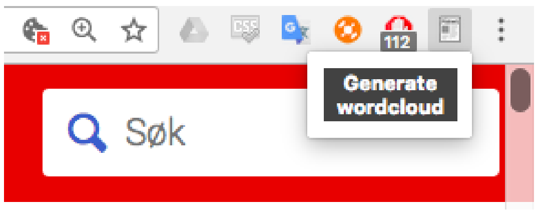
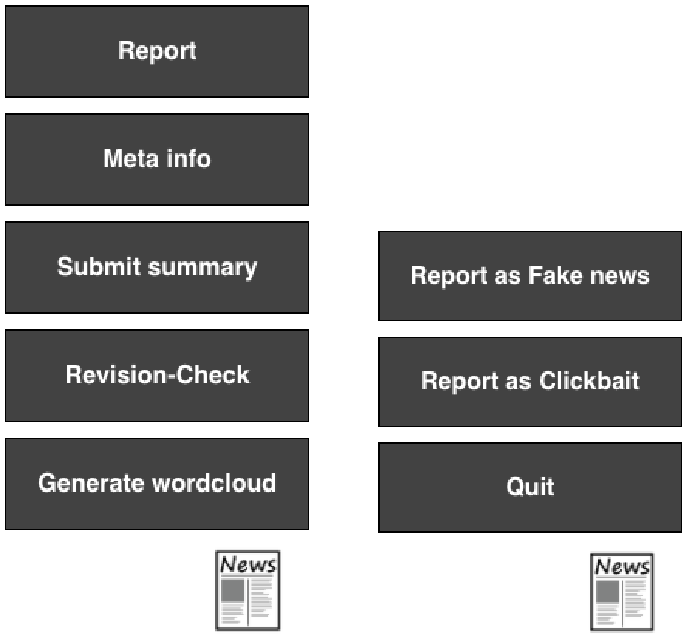


Figure 4.14: Generate Word Cloud from a front page menu.

4.5.3 Article

The article menu is located in the bottom right corner, and is a larger version of the NewsEnhancer icon (see figure 4.3). The menu includes the options listed in figure 4.15a. options:



(a) User interface for the article menu. Appears if the article is supported and if the user clicks on the news paper icon.

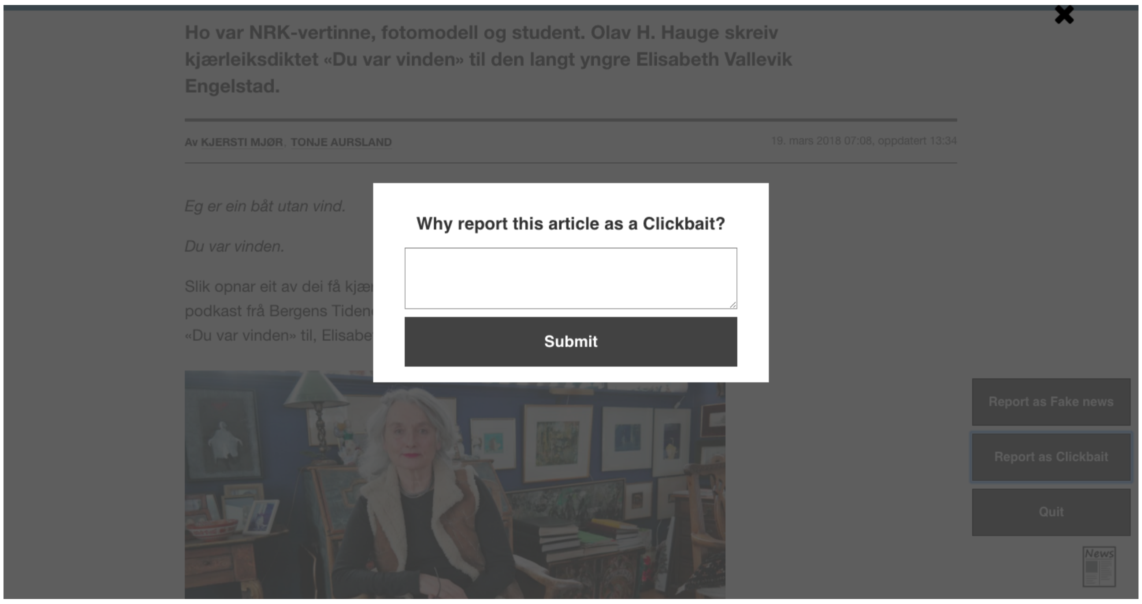
(b) User interface for the report sub menu, which appears after the user have clicked on “Report as” in figure 4.15a

Figure 4.15: User interface menus for an article

Report as

A submenu appears when choosing the “Report as” option from the article menu (see figure 4.15b). This enables the user to report an article by a category. The user is required to justify why the article must be reported (as seen in figure 4.16) before submitting.

This thesis only use two categories, Clickbait or Fake news. These categories is stored in the database, and is dynamically created in the plugin. This means that whenever a report category is added or removed on the server, the plugin will mirror that change.



Ho var NRK-vertinne, fotomodell og student. Olav H. Hauge skreiv kjærleiksdiktet «Du var vinden» til den langt yngre Elisabeth Vallevik Engelstad.

Av KJERSTI MJØR, TONJE AURSLAND 19. mars 2018 07:08, oppdatert 13:34

Eg er ein båt utan vind.

Du var vinden.

Slik opnar eit av dei få kjær
podkast frå Bergens Tiden
«Du var vinden» til, Elisabe

Why report this article as a Clickbait?

Figure 4.16: User interface for reporting an article

Submit summary

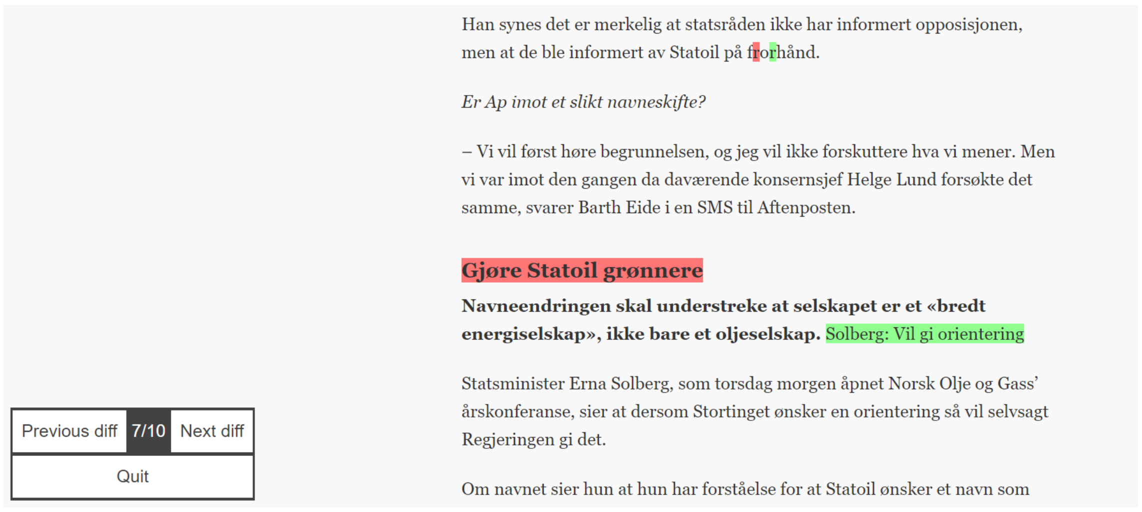
Enables the user to submit a one-line summary of the article. The submitting process is identical to submitting a report (see figure 4.16).

Metadata

The metadata options shows the same information box as when hovering over the articles headline (see page 40).

Revision mode

The different updates, or diffs, an article has, is presented in-line in the HTML, and lets the user navigate between the different versions. The navigation is located in the bottom left corner and enables the user to either go forward or backwards in the revision history, and also view what the current version number is, as shown in figure 4.17. To view all the revisions, the user must click the “revision check” option in the article menu at the bottom right corner.


Han synes det er merkelig at statsråden ikke har informert opposisjonen, men at de ble informert av Statoil på frorhånd.

Er Ap imot et slikt navneskifte?

– Vi vil først høre begrunnelsen, og jeg vil ikke forskuttere hva vi mener. Men vi var imot den gangen da daværende konsernsjef Helge Lund forsøkte det samme, svarer Barth Eide i en SMS til Aftenposten.

Gjøre Statoil grønnere

Navneendringen skal understreke at selskapet er et «bredt energiselskap», ikke bare et oljeselskap. Solberg: Vil gi orientering

Statsminister Erna Solberg, som torsdag morgen åpnet Norsk Olje og Gass' årskonferanse, sier at dersom Stortinget ønsker en orientering så vil selvsagt Regjeringen gi det.

Om navnet sier hun at hun har forståelse for at Statoil ønsker et navn som

Figure 4.17: Client user interface for navigating between the different article revisions. Green indicates inserted text and red removed text.

Word cloud

Retrieves a pre-generated Word cloud for the article and displays it in a modal. Is the same Word cloud one would get when generating a Word cloud by right clicking on a headline (see page 43).

5 Results and discussion

This chapter discusses the results obtained during this thesis, but also the challenges along the way. The last section of this chapter discusses how the plugin or web scraper can be blocked or rendered useless by the news sites.

5.1 Web scraper

The web scraper manages to extract data from the different sites even though they are structured very differently. This is mostly because all the sites contains the same kind of data, but the data location in the HTML document is different. This means that by using only one parser to define what data should be extracted, and have templates for each news site that defines where the data should be extracted from, works really well. However, not all of the news sites represents the data in the same way, see identifying headlines and articles on page 38, which poses a challenge.

Layout change

One of the downsides by using a specific parser to extract data from a website, is that the parser would be useless if the web site updates its HTML structure. Websites don't do that very often, but it means that when the website changes its layout then a programmer would have to write a new parser. This thesis has had its focus on writing a general parser that knows what data is to be extracted, but receives the location where the data should be extracted from within the HTML document in the form of templates. These templates are stored in the database and can be edited via the Django[29] admin interface in a browser. There is no need to restart the web application to register new or edited parsers.

vg.no changed their layout during this thesis which "broke" the web scraper for vg.no. That was because the CSS selectors defined in the different templates (for headline and article) did not reflect any valid path in the "new" website's HTML document. However, this was easily fixed by updating the templates to reflect actual valid CSS selectors. The time it took to update the CSS selector paths was no more than 2 minutes. Writing a new parser and registering it in the web application would take more time. It would also require personnel with access directly into the server, which may expose unnecessary security risk. However, a user account on the Django admin can easily be managed to have only access to edit templates and nothing more.

Duplicate headline revisions

dn.no have some headlines with more then 200 revision, but with only 2-4 different unique ones. This is clearly a bug, but it is also only happening on dn.no. This could either be some form for a/b testing where dn.no rotates two or more titles for each incoming request, or there exist duplicate headlines on the website but with different titles. Both cases would mean that each time the front page is scraped then at least 2 revisions is created. This could give at least 144 new revisions per day (the web scraper scrapes a site each 20 minute which is roughly 72 times a day). The revision count can be much higher if the headlines is kept on the front page longer.

The a/b case could only be fixed by removing all duplicate revisions on the server. However, this could lead to removing headline revisions where the journalist regrets a change and revert it back to the way it was before. Which is a perfect legit action. But this case is not very likely.

The duplicate headlines case could be fixed by removing duplicates during the scraping process. However, deciding which “duplicate” should be kept could be troublesome. One could check with all the previous revisions per headline, but that would increase the time to scrape a web site by a large amount.

5.1.1 Collecting data

There is a large quantity of data collected by the web scraper. These comes primarily from two types of sources (either a headline or an article), but is collected for each news site:

1. **Headline** Title, subtitle and position of headline on the front page.

2. **Article** Title, image(s), journalist(s), timestamps and the article itself.

Separating photographs from images

Each image on a news site also contains the photographer(s) and a caption. Many of the supported news sites did not separate the photographer(s) and caption into separate HTML tags, which made it harder to extract this information as two separate data. However, all of the sites which did not do this separation used a “prefix” for the photograph
(I.E `image caption PREFIX photographer`). This “prefix” did not just vary for each news site, but it did vary within the news sites as well. This may be because the news site gets their images from multiple sources, where each source may also supply the image text.

The solution used was to register a list of photograph “prefix” in the article template, and then try to split the image text on each of the “prefix” list items. The correct “prefix” for an image is found when the split operation yielded 2 results (an image caption and the photographer). However, this could lead to a false positive, where the image caption actually contains one of the “prefixs”, which would not extract the caption or photographer(s) correctly.

Headline statistics

The headline data, and especially the position, enables us to track a headlines movement, or lifecycle, on the front page. It could be really interesting to see which headlines stay in the top 5 the longest vs which headlines is the shortest time in the top. However, these type of statistics is not implemented during this thesis due to them not being a priority.

Article statistics

The data collected from an article can be connected to at least one or more journalists. That is because those journalists is the ones who wrote and signed the article with their own names. These data can help to achieve the second goal (on page 4) which is to make the journalists more accountable for their work, by making the journalists dependable on their reputation (which they to some degree already are, but there is no searchable overview for journalists or tools that allows a user to submit report an journalist or their work). This can be done by presenting a list of statistics per news site or overall for all the journalists. One of these statistics could be the ratio of flagged articles (clickbait or fake news) vs non flagged articles. Then present the list of journalist in the plugin when they are visiting a news site or having a separate website that displays this statistics. In addition one could also mark the headline in the information box, if the journalist who wrote the article had been flagged many times, to show that this journalist often use clickbait headlines etc. This would let the user immediately know when they hover over a headline that this journalist's work might not be legit. To make it even more interesting one could use the reports from fact checking sites like faktisk.no[17] or snopes.com[18], if any, and mark the headline in the same way as with flagging, but this time to show that the content of the article is either true or false. Of course, a combination of the two would be even stronger.

This have not been implemented for this thesis because:

1. Fact checking sites, especially for Norwegian news, does not have an API to service these reports, which means that one would have to scrape these data and that has not been a priority.

2. Because of the thesis time limit, this did not have the highest priority.

5.1.2 Legal aspect

The content of a website is owned by a company, but their website is public and everybody can access their content. Is it allowed to take data from these websites? Or to store these data on your own server and use it? How about accessing the site as a web scraper at all. Is that legal? The following sections will shed light on these questions.

Robots.txt

Most sites define a resource file called robots.txt. This file defines what content is allowed to be accessed by robots. It is not a legal document, but more of a advisory file on how to behave on that site. The worst that can happen when ignoring this rules, is to get your

Internet Protocol blocked, and therefore not getting access to the resource. However, a web scraper is treated as a robot, so if the robots.txt disallows robots to access the site, then they will most likely not have any web scrapers there either. But again this has to be respected from the one who scrapes the site. There is not really anything the web site can do except to block the Internet Protocol. This blocking can be circumvented by Internet Protocol rotation.

Terms of Service
Most sites have a terms of service on what applies, when using their sites and accessing their contents. Most of the supported sites have a terms of service for what applies to the articles, text and images as shown in table 5.1. These terms of service states that the site has copyright for all content, and the use of their content on other sites, mostly in a commercial context, is prohibited. This would most likely mean that it is allowed to take these data and store it on a server, like buying a newspaper and keep it in an archive at home. However, if these data is being used in any way, for commercial use etc, then it is not allowed without any deals with that particular site.

Table 5.1: Terms of service.

News site	Terms	Where	External sites disclaimer
nrk.no	Yes	Footer	-
db.no	-	-	-
dn.no	Yes	Dedicate page	-
vg.no	Yes	Yes	Yes
ap.no	Yes	Footer	Yes
bt.no	-	-	-
ba.no	Yes	Footer	Yes

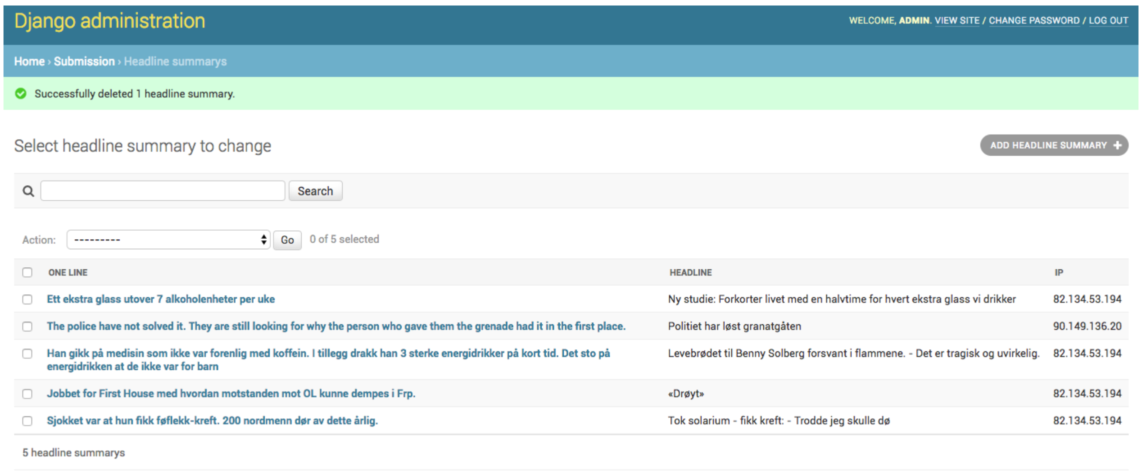
The matter of legality is very tricky for this thesis, because data from these news site is required in order to give more information to the user. The only crowdsourced, and 100% legal data, is the one-line summaries. But that might not be enough information for the user to decide whether or not to click the headline, and certainly not enough to collect statistics on journalists. In addition, showing one-line summaries of articles behind pay-walls is probably debatable.

5.2 Web application

5.2.1 Moderation

Browser extensions like TL;DR[10] lets users submit comments on any links, but without moderation. It can be very dangerous to allow users submit content anonymously, and let other users read it. Without any form for moderation, the users would be free to send in whatever they like. Profanity, racism, fake content, etc. This thesis enables users to submit one-line summaries of articles, and proposes to use moderators who review all submitted content and picks out the best summary. However, even moderators can be bias and subjective, which can make this solution not an optimal one.

Figure 5.1 shows the web user interface (Django admin) into the Django[29] database, where anyone who have been given access, may see the reported data and possibly remove erroneous reporting. The right column shows headlines, and the left column shows the summaries of the articles made by plugin users. Giving moderators direct access to the database opens up for vulnerability in the system, but the Django admin also makes it really easy to manage users and what resources they have access to.


Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Submission › Headline summarys
ADD HEADLINE SUMMARY +
Search
Action: ---------
Go

Successfully deleted 1 headline summary.

Select headline summary to change

0 of 5 selected

ONE LINE	HEADLINE	IP
Ett ekstra glass utover 7 alkoholenheter per uke	Ny studie: Forkorter livet med en halvtime for hvert ekstra glass vi drikker	82.134.53.194
The police have not solved it. They are still looking for why the person who gave them the grenade had it in the first place.	Politiet har løst granatgåten	90.149.136.20
Han gikk på medisin som ikke var forenlig med koffein. I tillegg drakk han 3 sterke energidrikker på kort tid. Det sto på energidrikken at de ikke var for barn	Levebrødet til Benny Solberg forsvant i flammene. - Det er tragisk og uvirkelig.	82.134.53.194
Jobbet for First House med hvordan motstanden mot OL kunne dempes i Frp.	«Drøyt»	82.134.53.194
Sjokket var at hun fikk føflekk-kreft. 200 nordmenn dør av dette årlig.	Tok solarium - fikk kreft: - Trodde jeg skulle dø	82.134.53.194

5 headline summarys

Figure 5.1: Moderation of incoming summaries (some empty space in the original screenshot of the GUI has been removed)

Artificial intelligence could decide more objectively, than moderators, which summary to use. But comparing different summaries with an article and finding which summary stays most true to the article message, is just a to large task. That is why artificial intelligence was discarded for this thesis. For simplicity during development, this thesis automatically accepted the first submitted summary for each headline.

5.3 Background tasks

The time it took to scrape all the supported news site under development was 11 minutes. When deploying to the first server (shown in table 5.2), the background task was set to run each 20 minute to allow some wiggle room (no need to scrape the same sites at the same time). However, the server used only 6.37 minutes to do the full background task. The servers system was incredible slow during the background task. The system was so slow that each incoming request to the RESTApi timed out, rendering the plugin useless since the plugin did not get any data. However, these issues were dismissed by upgrading the server to one with two CPU's and more memory (see table 5.3). The average duration for one background task stayed roughly the same. This worked really well under during development, but if the web scraping framework should be used in production, then the background task frequency should be increased. That is because the more articles the scraper scraped, the more updates one could get, which means that tracking changes and journalists would be more accurate. However, this would also require a more powerful server.

Table 5.2: Specifications of the first server

Memory	1GB
Virtual CPU	1
SSD	25GB
Operating System	Ubuntu 16.04.3

Table 5.3: Specifications of the first server

Memory	4GB
Virtual CPU	2
SSD	80GB
Operating System	Ubuntu 16.04.3

5.4 Client

The client worked really well by showing an articles metadata and user submitted summaries. However, there was some issues with the different CSS rules across the news sites as the sites has very different layouts.

Marking

All supported headlines are marked in their top right corner by a circle with an “i” inside. The color of this circle is by default black, but when users reports that headline/article for being clickbait or fake news, the color changes from yellow to more and more red (as shown in figure 5.2). This marking lets the user know if the headline is supported, or not, but may not be presented in the same way across all the news sites because the CSS specifications is different. One example is nrk.no where all the headline markings is grouped together in the top right corner of the website and not spread across on each headline. This is decided by the website’s CSS. This makes it “hard” for the user to know if a headline is supported, or not, by just looking at it. The user would have to hover over each headline to check if it is supported. However, this could be fixed by overriding the website’s CSS and set all headline elements to `position: relative;`. This is not recommended since it can drastically change the website’s layout and how it is presented in the browser.

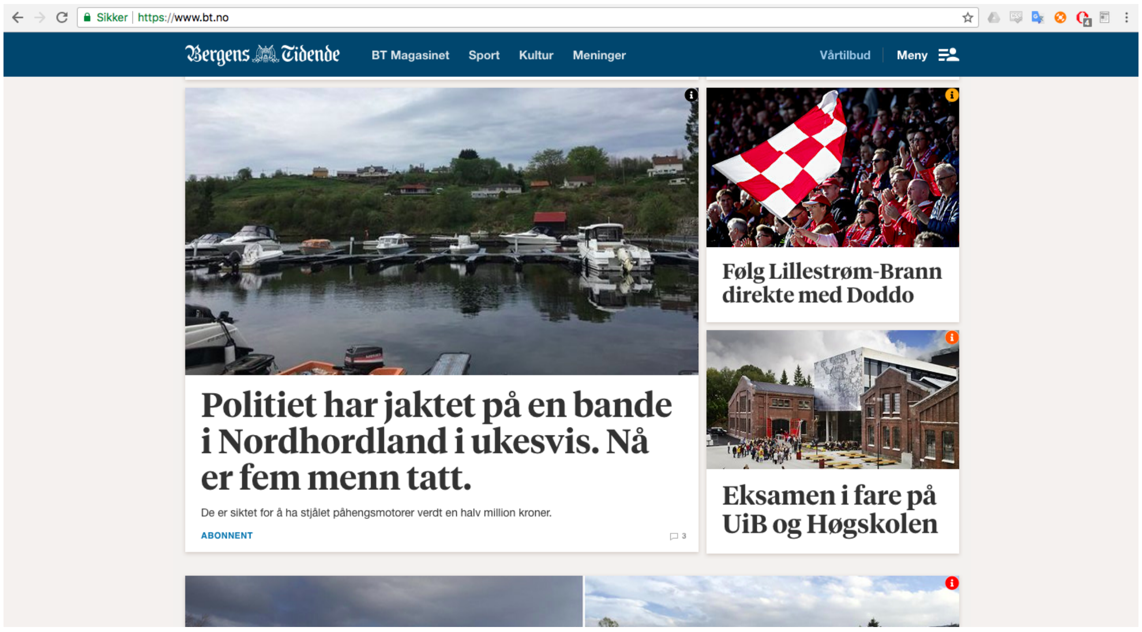
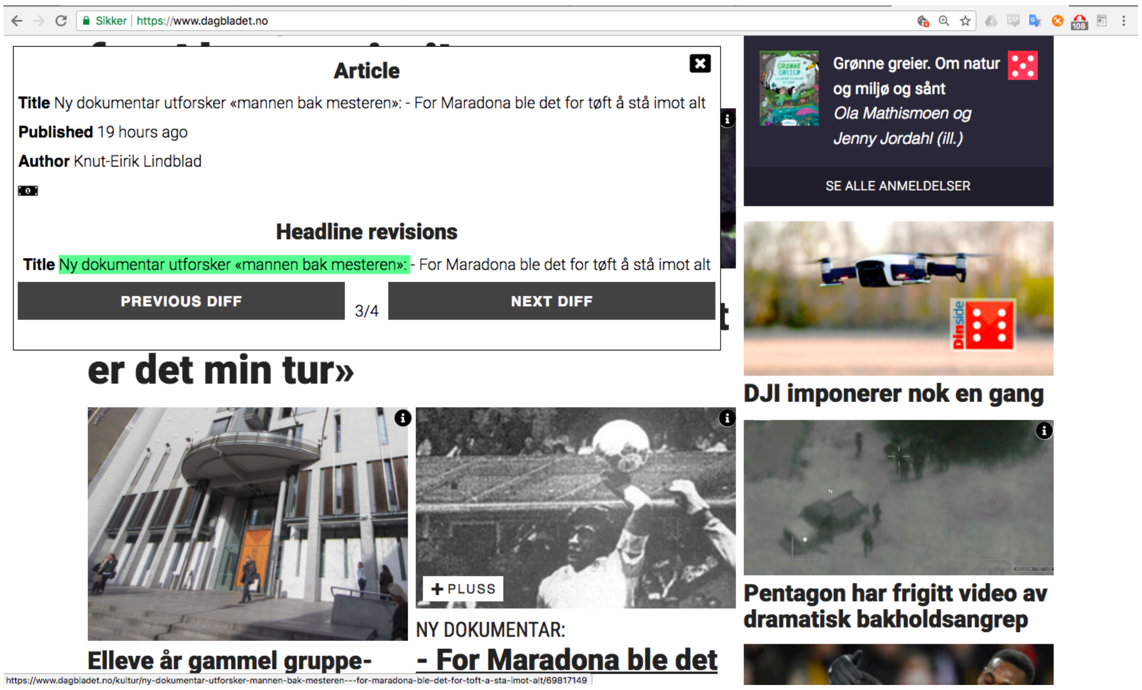

Sikker | https://www.bt.no
Bergens Tidende
BT Magasinet
Sport
Kultur
Meninger
Vårtilbud
Meny
Politiet har jaktet på en bande i Nordhordland i ukesvis. Nå er fem menn tatt.
De er siktet for å ha stjålet påhengsmotorer verdt en halv million kroner.
ABONNENT
Følg Lillestrøm-Brann direkte med Doddo
Eksamen i fare på UiB og Høgskolen

Figure 5.2: The different report levels. These values is adjustable in the database (values used under development in parentheses).
Black = No reports.
Yellow = Low number of reports (1).
Orange = Medium number of reports (2).
Red = High number of reports (¿2).

Inherits site CSS

All elements that is created and injected on the website by the plugin, inherits the web site's CSS. This means that the website defines how everything should look by default. This can be overridden by creating own specific CSS rules, but even with a specific font size, the overlay is presented differently across the news sites. This in turn have led to some sites having very large information boxes because of the font size, when the font size is actually the same across all news sites. This gets really annoying because then the plugin lacks consistency across the news sites, but also because a larger information box blocks a larger part of the screen (as shown in figure 5.3).



Article

Title Ny dokumentar utforsker «mannen bak mesteren»: - For Maradona ble det for tøft å stå imot alt

Published 19 hours ago

Author Knut-Eirik Lindblad

Headline revisions

Title Ny dokumentar utforsker «mannen bak mesteren»: - For Maradona ble det for tøft å stå imot alt

PREVIOUS DIFF 3/4 NEXT DIFF

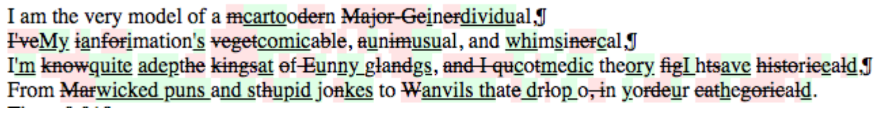
Sikker | https://www.dagbladet.no
Grønne greier. Om natur og miljø og sånt
Ola Mathismoen og Jenny Jordahl (ill.)
SE ALLE ANMELDELSER
er det min tur»
DJI imponerer nok en gang
+PLUSS
NY DOKUMENTAR:
- For Maradona ble det
Pentagon har frigitt video av dramatisk bakholdsangrep
Elleve år gammel gruppe-
https://www.dagbladet.no/kultur/ny-dokumentar-utforsker-mannen-bak-mesteren---for-maradona-ble-det-for-toft-a-sta-imot-alt/69817149

Figure 5.3: A large information box that blocks a very large surface of the screen.

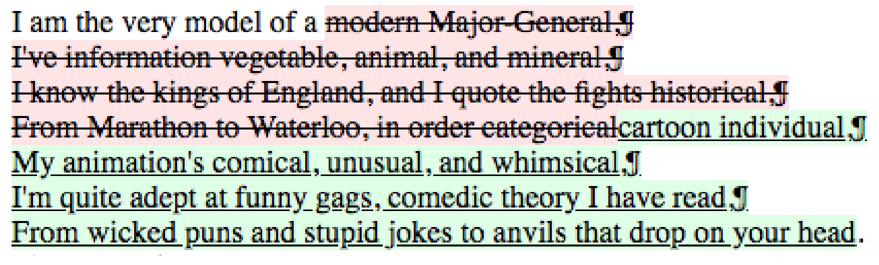
Information box
Some news sites have a very large information box. This is due to the websites CSS, as mention in the previous paragraph, but also when many users have reported the headline (see figure 4.11) or if the different texts (title, subtitle, revision etc.) is very long. This can be solved by setting a fixed width on the information box and then either let it scroll horizontal or line break on words that is past information box border. However, when using horizontal scroll, the user may sometime see only a part of the text, and not all together as with line breaks.

5.4.1 Revisions

The diffs worked very well by highlighting the changes between two versions of the text. However, the changes should be more readable then they are as shown in figure 5.4. The readability also improved drastically, when using lighter highlights (see figure 4.12 for a very dark highlight and figure 4.17 for a lighter one).


I am the very model of a ~~m~~cartoo~~dern~~ ~~Major-Ge~~i~~ner~~dividual,¶
~~I've~~My ~~i~~an~~for~~imation's ~~veget~~comica~~b~~le, ~~a~~un~~im~~usual, and whimsi~~ner~~cal,¶
I'm ~~know~~quite adep~~the~~ ~~kings~~at ~~of E~~unny g~~l~~a~~nd~~gs, ~~and I qu~~co~~t~~medic theory ~~fig~~I h~~ts~~ave ~~historie~~ca~~l~~d,¶
From ~~Mar~~wicked puns and st~~h~~upid jo~~n~~kes to ~~W~~anvils that~~e~~ dr~~l~~op o~~, i~~n yo~~rde~~ur ~~eat~~h~~egorie~~a~~l~~d.

(a) Normal diff, not very readable.


I am the very model of a ~~modern Major-General,¶~~
~~I've information vegetable, animal, and mineral,¶~~
~~I know the kings of England, and I quote the fights historical,¶~~
~~From Marathon to Waterloo, in order categorical~~cartoon individual,¶
My animation's comical, unusual, and whimsical,¶
I'm quite adept at funny gags, comedic theory I have read,¶
From wicked puns and stupid jokes to anvils that drop on your head.

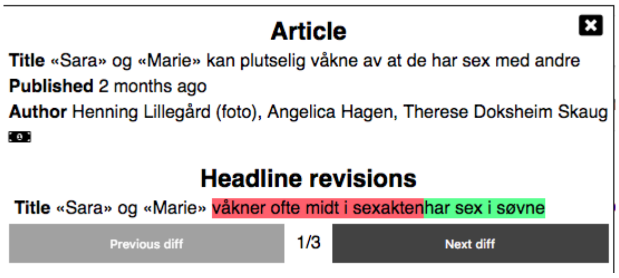
(b) Human readable diff.

Figure 5.4: Example of a human readable diff vs a normal diff (source from diff_match_patch author Neil Fraser diff demo).

Headline

The headline diffs were very interesting because they showed how a news site tries to increasingly make the headline title more attractive or clickable. Below is listed three examples where first shows how a headline can be more attractive by talking about sex. The second example shows how the news sites tones down the a headline. The third and final example shows how the news site plays around with different titles, where some of them is taken completely out of context and making the title to have no relation to the actual article.

1. An article about people with sexsomnia (who has sex when they are asleep with their partner). The original title was “Sara and Marie often wakes during sex”. This is then changed to “Sara and Marie have sex in their sleep” (see figure 5.5a). Which is true, but just sounds more sexy that they have sex in their sleep with their partners. This is later changed to “The orgasm is more intense” (as seen in figure 5.5b), which the article barley mentions that some people with sexsomnia can feel a more intense orgasm, but is completely taken out of context. This is later changed to the actual title of the article, “Sara and Marie can suddenly wake up during intercourse” which is actually more fitting.



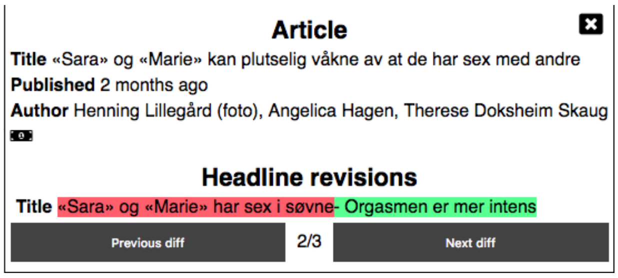
Article

Title «Sara» og «Marie» kan plutselig våkne av at de har sex med andre
Published 2 months ago
Author Henning Lillegård (foto), Angelica Hagen, Therese Doksheim Skaug

Headline revisions

Title «Sara» og «Marie» våkner ofte midt i sexaktenhar sex i søvne

(a)



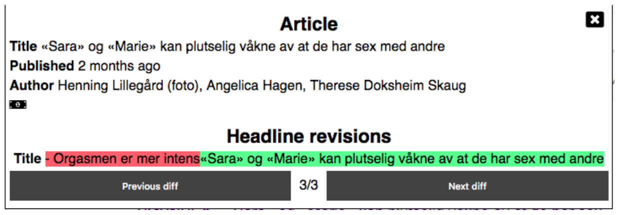
Article

Title «Sara» og «Marie» kan plutselig våkne av at de har sex med andre
Published 2 months ago
Author Henning Lillegård (foto), Angelica Hagen, Therese Doksheim Skaug

Headline revisions

Title «Sara» og «Marie» har sex i søvne- Orgasmen er mer intens

(b)



Article

Title «Sara» og «Marie» kan plutselig våkne av at de har sex med andre
Published 2 months ago
Author Henning Lillegård (foto), Angelica Hagen, Therese Doksheim Skaug

Headline revisions

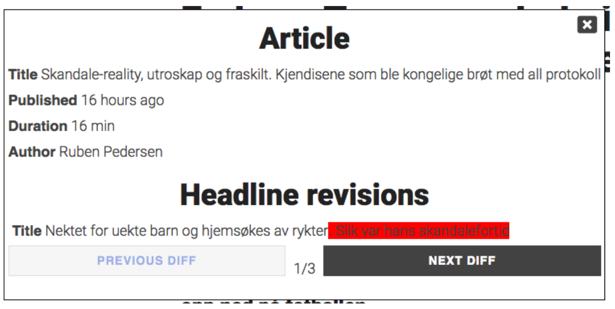
Title - Orgasmen er mer intens«Sara» og «Marie» kan plutselig våkne av at de har sex med andre

(c)

Figure 5.5: The images a, b, and c shows how each update in the headline title tries to make the headline more appealing and also walking farther from the truth. The article is about people who has sex when they are asleep with their partners.

2. This is about the scandalous life of Prince Albert of Monaco, where there is rumors that he has a child outside marriage. The original title was, as shown below in figure 5.6a, “Refuses for illegitimate children and is hunted by rumors. This was his scandal past”. “This was his scandal past” was later removed before also removing “and is hunted by rumors”, as seen in figure 5.6b. The last edit was changing illegitimate children with paternity.

   In this case, the headline starts tabloid-like, but is moderated for each revision. It would be interesting to know the motivation behind this development from a sensational towards a more factual headline.



Article

Title Skandale-reality, utroskap og fraskilt. Kjendisene som ble kongelige brøt med all protokoll

Published 16 hours ago

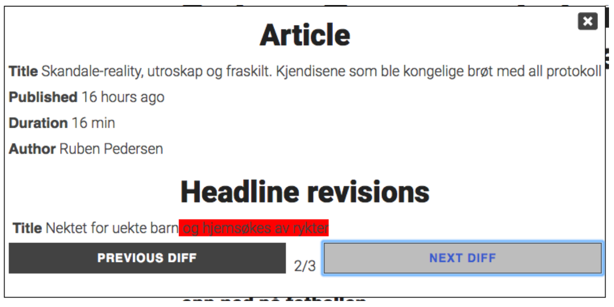
Duration 16 min

Author Ruben Pedersen

Headline revisions

Title Nektet for uekte barn og hjemsøkes av rykter. Slik var hans skandalefortid

(a)



Article

Title Skandale-reality, utroskap og fraskilt. Kjendisene som ble kongelige brøt med all protokoll

Published 16 hours ago

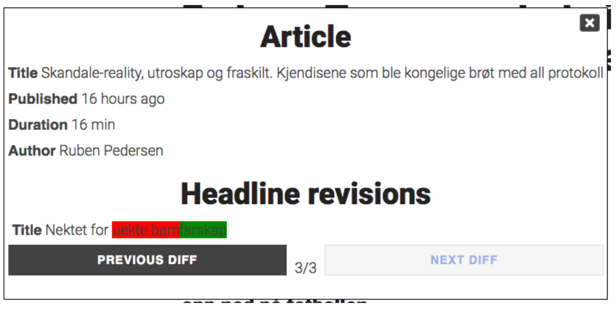
Duration 16 min

Author Ruben Pedersen

Headline revisions

Title Nektet for uekte barn og hjemsøkes av rykter

(b)



Article

Title Skandale-reality, utroskap og fraskilt. Kjendisene som ble kongelige brøt med all protokoll

Published 16 hours ago

Duration 16 min

Author Ruben Pedersen

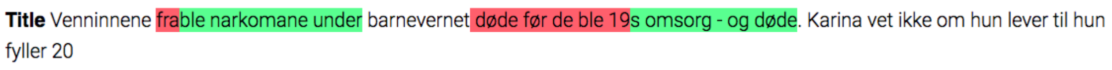
Headline revisions

Title Nektet for uekte barnfarskap

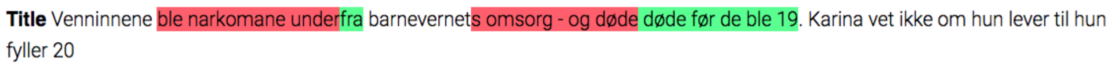
(c)

Figure 5.6: The images a, b, and c shows how each update in the headline title is more and more toned down over the course of 4 different versions (3 updates). The article is about a celebrity who rumors to have child outside the marriage.

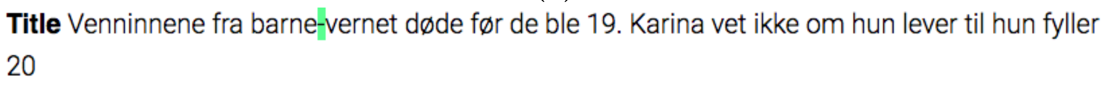
3. A tragic story about three young girls under state child welfare, where two of them became drug addicts and died. The article title is "Barnevernets Engler" which translates roughly to "The Child Welfare Angels". The two first headlines are informative, then the following headlines are more clickbait-like and do not contain context information. They also become increasingly more dark. It is also interesting to see how the journalist chooses to change between clickbait-like headlines as to see what attracts most people.


Title Venninnene frable narkomane under barnevernet døde før de ble 19s omsorg - og døde. Karina vet ikke om hun lever til hun fyller 20

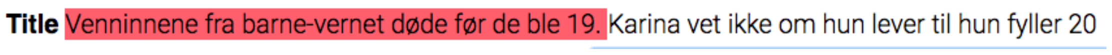
(a)


Title Venninnene ble narkomane underfra barnevernets omsorg - og døde døde før de ble 19. Karina vet ikke om hun lever til hun fyller 20

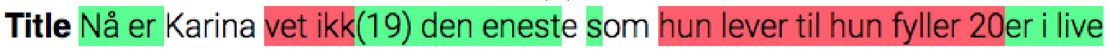
(b)


Title Venninnene fra barne-vernet døde før de ble 19. Karina vet ikke om hun lever til hun fyller 20

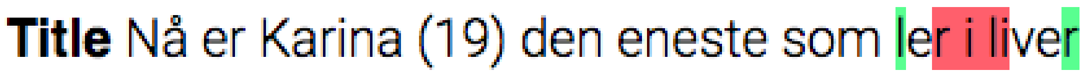
(c)


Title Venninnene fra barne-vernet døde før de ble 19. Karina vet ikke om hun lever til hun fyller 20

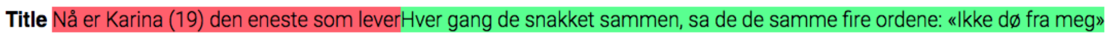
(d)


Title Nå er Karina vet ikk(19) den eneste som hun lever til hun fyller 20er i live

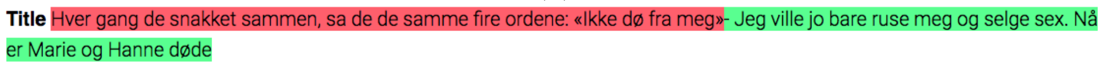
(e)


Title Nå er Karina (19) den eneste som ler i liver

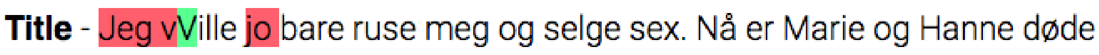
(f)


Title Nå er Karina (19) den eneste som leverHver gang de snakket sammen, sa de de samme fire ordene: «Ikke dø fra meg»

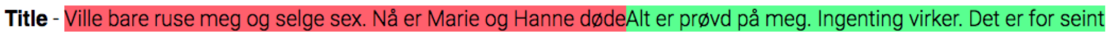
(g)


Title Hver gang de snakket sammen, sa de de samme fire ordene: «Ikke dø fra meg»- Jeg ville jo bare ruse meg og selge sex. Nå er Marie og Hanne døde

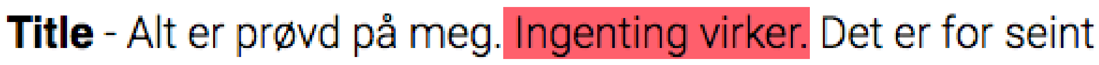
(h)


Title - Jeg vVille jo bare ruse meg og selge sex. Nå er Marie og Hanne døde

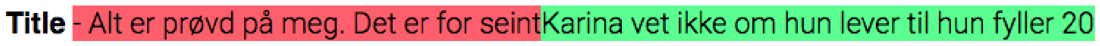
(i)


Title - Ville bare ruse meg og selge sex. Nå er Marie og Hanne dødeAlt er prøvd på meg. Ingenting virker. Det er for seint

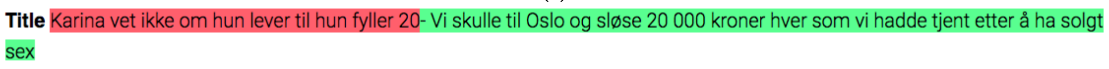
(j)


Title - Alt er prøvd på meg. Ingenting virker. Det er for seint

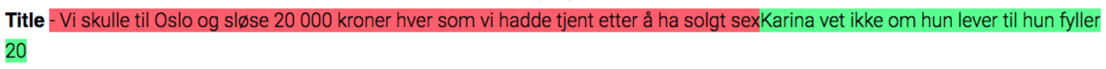
(k)


Title - Alt er prøvd på meg. Det er for seintKarina vet ikke om hun lever til hun fyller 20

(l)


Title Karina vet ikke om hun lever til hun fyller 20- Vi skulle til Oslo og sløse 20 000 kroner hver som vi hadde tjent etter å ha solgt sex

(m)


Title - Vi skulle til Oslo og sløse 20 000 kroner hver som vi hadde tjent etter å ha solgt sexKarina vet ikke om hun lever til hun fyller 20

(n)

Figure 5.7: Headlines taken out of context

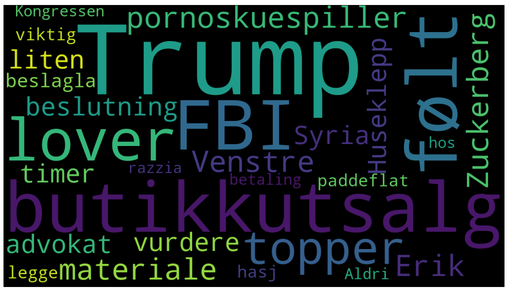
Article

The article diffs works really well, but have not yielded any specific results as the headline diffs. An example of an article revision is shown in Figure 4.17. Such revisions show for instance how information is added to an article, and how typographic errors are corrected.

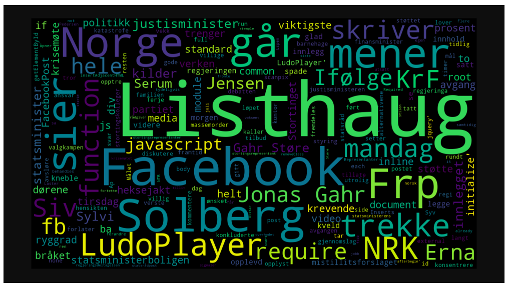
5.4.2 Word Cloud

The Word clouds give a quick overview of the content of an article or the most mentioned headlines on a front page. However, the list of stop words could be extended even more. In addition, in figure 5.8b, there is mentioned the words "LudoPlayer" and "require". These words are keywords in the programming language Javascript. They should also be excluded. To strip an article of any Javascript code, comments and other non article text would be even better solution, but that was not done in this thesis.

Figure 5.8a, shows a Word cloud generated from a newspaper's frontpage. The most prominent headlines on that day was Trumps involvement with a porn actress, Zuckerbergs hearing in congress and chemical attacks in Syria. Figure 5.8b shows the Word cloud generated from a news article about the politician Listhaug resigning from position due to posting controversial comments on Facebook.



(a) Wordcloud generated from the front page a bt.no (11 April 2018).



(b) Wordcloud generated from an article at nrk.no (19 April 2018).

Figure 5.8: Different Word Clouds

Mixed content error

The Word cloud is an image which is represented by an `<img src="urlToImage" />`. This image tag is injected into the DOM of the modal to display the image. However, this would result in an error when using the plugin on the the supported sites. The error is called "Mixed Content"[39], which refers to resources loaded over HTTP when the initial connection to the website is by HTTPS (would apply to all images, CSS file, Javascript files or other resources injected via the plugin). Chrome blocks these "unsafe" connections because they are a security hole. Luckily this can be solved by using HTTPS on the server (which is free via Let's Encrypt). But it is also possible to download the image in a background script, where the protocol can be either HTTP or HTTPS and convert it to a base64 string. That is because the `<img />` can display images by using a valid URL in the "src" attribute, but also with a base64 string representation of an image. However, when trying to reproduce this error later in the thesis, the `<img />` automatically replaces HTTP with HTTPS, resulting in a 404 error message from the server instead. Luckily the base64 solution still applies.

5.5 Restriction

The news sites cannot restrict access to their website without doing so for all the users, except blocking specific Internet Protocol addresses, which would not benefit the news sites at all. But they can make the web scraping process very hard if not impossible and they can also selectively block the content on their website for user with the plugin installed.

5.5.1 Web scraping

The web scraper need the HTML of the website structured in the same way on each request. Even some small deviation in the HTML structure could potentially break the web scraper. There is a couple scenarios where this could happen. The first and most realistic scenario is when the website changes its layout.

Changing layout

Websites can change their site layout whenever they want. They may do it to update the look and feel of the website, or to restructure content etc. This will always effect the HTML structure of the website. Which in turn, could break the web scraper, which only have one way to extract data from that site. This would also apply to just change the CSS names since the web scraper relies on CSS selector. These kind of changes is easily fixed by updating the different scraping templates with the new CSS selector. However, this could be very cumbersome to do very often, but would still be necessary if the web scraper should continue extracting data.

Dynamic CSS

Some sites obfuscates the CSS classes from readable names like `<button class="btn-lg btn-red" />` to `<button class="fsew qwer" />`. This is done by obfuscates the .css files and replace the new class name in other files (.html, .js etc..). This is often done to reduce the file size of static files (CSS and Javascript), which can make the website load faster (less resources to download), but can also make it more challenging to identify elements and their CSS selector. This process, called minification, is usually done once (typically when building the application or releasing it) and then cache the resulting files to make the site's load time faster. However, it could also be done on each request, but would slow the site's load time significantly. It would also make it hard if not impossible to scrape the website with CSS selectors, since the resource path would change on each request.

5.5.2 Plugin

The plugin resides on the client, which means that the news sites cannot do anything when a website is requested. However, they may check, via Javascript, after the web site has been downloaded if the DOM contains specific classes or id's and then block content or similar. It is pretty easily to stop Javascript from web sites to run in the browser by disabling it, but that would also render the plugin, which relies heavily on Javascript, to be useless.

6 Conclusion

The introduction of this thesis described two goals (see page 4):

1. To produce quick overview of news by e.g. helping readers avoid spending time on irrelevant news and guide readers towards relevant news.
This goal have 4 subgoals to achieving the primary goal:

a) Showing an article's metadata when the user is reading the article's headline. This is to increase the information base the user has in order to make a decision on whether or not to click that headline.

b) Summarizing the front page of a news site in one single picture by generating a Word cloud from all headlines, and summarizing single articles by generating a Word cloud from the article text.

c) Allow readers to flag articles as clickbait or fake news, and displaying a flag on headlines that have been reported as such.

d) Allow readers to submit one-line summaries of articles, and showing these summaries in the articles metadata when hovering over the the article's headline.

Subgoal 1a helped to produce a quick overview of news by showing an article's metadata in an information box via the plugin. The small test group found the article title to be the most interesting metadata. Viewing the article and headline title next to each other, makes it really effortless to spot any deviations between the two (as shown in figure 4.9 where there is little to no deviation in the message between the two), which can make it easier to detect clickbait headlines. However, it could be argued how effective it is to know when an article is published and by whom, when deciding what article to click.

Subgoal 1b did not always help as much as 1a to contribute to produce a quick overview. The goal itself was implemented by showing a Word cloud, see figure 5.8a for an example, for each article via the "Word cloud" option in the context menu. These Word clouds made much sense after having read the news, when the words were known, but it can be argued that they had less effect otherwise. Also, the list of stop words needs to be improved and the article text needs to be cleaned up for programming code and comments, before generating the Word cloud.

Both subgoal 1c and 1d, used crowdsourcing to help other users to produce a quick overview of the news. This was done by either submitting a summary of an article

(1d) or flag a headline (1c) as either clickbait or fake news. Both was presented in the information box, as shown in figure 4.11.

The summary helped greatly to let other know what the article is all about. Especially if the headline was overly creative. However, it would be very easy for users to submit false information, but this is solved by using a moderator that chooses which summary to use.

The flagging feature made it really easy to see why one should not read that specific article. This is because each flag required the one submitting it, to justify why this headline should be flagged.

However, crowdsourcing requires users and the test group used under development was too small, and reported rarely, which made this very hard to test properly.

All subgoals, for the first goal, was fully reached and worked well. The user gets a lot more information about each headline and its corresponding article, which makes it easier to filter out articles that is actually relevant.

2. To make the journalists and news sites more accountable for their work.
This goal had 3 subgoals to achieving the primary goal:

a) Enabling information sharing between readers, so they can warn each other regarding clickbait and fake news.

b) Being able to compare the current version of headlines and articles content with their previous version, to track the changes journalists make.

c) Collecting and presenting statistics on journalists regarding e.g. how often their articles are reported as clickbait or fake news.

Subgoal 2a enables information sharing between users by letting them submit summaries for articles, or flagging headlines as clickbait or fake news (see figure 4.15b and 4.16). This can help to make journalists more accountable for their work by using these submissions to present statistics (2c) on flagged/not flagged ratio for journalist and news site. However, that would require more active users to keep flagging headlines. The presenting statistic part was also not included in this thesis. This was because of the thesis time limit and too little data basis due to a small test group.

Subgoal 2b could help to make journalists more accountable for their work by archiving all changes done to an article or a headline. This also enables users to navigate between and compare different versions, or revisions, of an article or headline (shown in figure 4.17 for article diffs and in figure 5.6 for headline diffs). The revisions revealed more interesting information than expected, particularly for headlines. The headline revisions showed how a journalists tried to make the title more attractive and clickable by taking words and sentences out of context (see figure 5.7).

Both the server and plugin is created to make it possible to reach all subgoals, however due to a too small test group, subgoal 2c was not implement. But the first goal (1) and most of the subgoals for the second goal(2) were reached, which makes the goals for this thesis largely achieved.

7 Future work

7.0.1 Collecting and presenting statistics

Presenting statistics for journalists has not been fully achieved as mentioned in the conclusion on page 62. All statistics should be presented in the plugin. Then the user would only need to relate to one tool for everything. The plugin should be able to filter statistics for journalists on each news site or overall. In addition, the information box should have different markings based on results from fact checking sites like faktisk.no[17] or snopes.com[18]. But also if the journalist is “known” for clickbait headlines or fake news articles as described on page 49.

7.0.2 Related articles

The plugin TLDR[7] has an option to show related articles based on keywords from the text. This is a great feature. However, instead of getting the results in the same tab, TLDR opens a new tab and uses Google to search for the keywords. This thesis would implement something similar, but make it easier for the user. The suggested feature would be pre-generated on the server and available from the article menu (see figure 4.15a). The result, a list of the most relevant articles, would also show up in-line inside the article.

List of most relevant articles

It would be easier for the user if the background service, on the server, fetched results from Google’s search API whenever it scraped articles and persisted the results. Then serve the 5-10 most relevant ones with the article. In addition, the Google Search API is also capable to include description text, title and images in the search result, making the links more exiting and descriptive.

Translating word for word

It would also be very interesting if all the stop words are removed from the article text, and translate it word for word to English. Then it would be possible to also search for related articles in English, but also to disclose if the journalist “stole” the article by copying and translating it without referring to the original one.

7.0.3 Full article scraping coverage

As mention in the limitation section, see 3.3.3, not all of the available articles was scraped. However, all articles could be scraped by adding some modifications to the web application.

Support for multiple article templates per site

All news sites has only one article template in this thesis. However, many news sites have different article layouts. Some, like dn.no, has a magazine format where the articles has a totally different HTML structure, rendering the default article template useless. So each news site should have a list of article templates, instead of just one, to have 100% article scraping coverage. The only problem is how to identify which article should use which template. dn.no's magazine format is called D2 and all articles in this magazine starts their URL with `DN.no/d2/`, however, not necessary all sites use this format. Another way could be to just try all article templates and use the the one which yielded the most result. This method however, would be terrible slow. Scraping the same article x times for each article template, and doing so for all headlines on the news site's front page would waste a lot of time on unnecessary scrapes.

Subscription articles

In an article on website The IT stuff Rishabh Kandari describes a session id to be a unique identifier that is issued from a site whenever a user enters correct username and password combination. He later explains that this id is then stored in the browser in what's called a Cookie. This Cookie is included on each request to the site to verify that this user is logged in[40] . Articles behind a Paywall could be supported by using frameworks like Selenium[21], to enter user-name and password into the the news sites login form and then fetching the session id from the cookies. This would only be necessary to do one time, then cache the session id and include it on each article request. However, there is a possibility that the sites has set the session Cookie to HTTPOnly which means that Selenium cannot extract information from it. Instead the scraper would have to use Selenium to log in and hope to get redirected back to the article.

Articles not on the front page

To support scraping articles that is not currently on the front page one would simply have to edit the `ArticleManager` to return **all** articles for that site, instead of only the ones on the front page.

Literature

[1] F. L. Mott, *American Journalism.* Routledge, 1941, ISBN: 9780415228947.

[2] A. S. Brisbane. (Jun. 2011). 'On nytimes.com, now you see it, now you don't', [Online]. Available: https://www.nytimes.com/2011/06/26/opinion/sunday/26pubed.html (accessed April 24, 2018).

[3] C. M. Al Baker and S. M. Nir. (October 2011). 'How to make money with a news website', [Online]. Available: https://cityroom.blogs.nytimes.com/2011/10/01/police-arresting-protesters-on-brooklyn-bridge/ (accessed April 24, 2018).

[4] H. Brody. (2017). 'B.s. detector homepage', [Online]. Available: http://bsdetector.tech/ (accessed November 7, 2017).

[5] OpenSources. (2017). 'Homepage', [Online]. Available: http://www.opensources.co/ (accessed November 7, 2017).

[6] snipe. (2017). 'Downworthy homepage', [Online]. Available: http://downworthy.snipe.net/ (accessed November 7, 2017).

[7] TLDR. (2018). 'Homepage', [Online]. Available: http://www.recognant.com/demos/tldr/ (accessed April 23, 2018).

[8] R. Kapoor. (2017). 'Homepage', [Online]. Available: https://github.com/rahulkapoor90/This-is-Clickbait (accessed April 23, 2018).

[9] S. Mathur. (2017). 'Homepage', [Online]. Available: https://github.com/saurabhmathur96/clickbait-detector (accessed April 23, 2018).

[10] TL;DR. (2017). 'Homepage', [Online]. Available: http://tldr.guru/ (accessed November 7, 2017).

[11] Rbutr. (2018). 'Homepage', [Online]. Available: http://rbutr.com/ (accessed April 23, 2018).

[12] M. R. Siddiki, M. A. Talha, F. Chowdhury, and M. S. Ferdous, "Crowdsours: A crowdsourced reputation system for identifying deceptive online contents", *2017 20th International Conference of Computer and Information Technology (ICCIT)*, pp. 1–6, 2017.

[13] NewsDiff. (2011). 'About newsdiff', [Online]. Available: http://www.newsdiffs.org/about/ (accessed January 29, 2018).

[14] J. E. D. (2016). 'Nytdiff github', [Online]. Available: https://github.com/j-e-d/NYTdiff (accessed April 28, 2018).

[15] Incdroid. (January 2017). 'Tracking changes with diffengine', [Online]. Available: https://inkdroid.org/2017/01/13/diffengine/ (accessed April 28, 2018).

[16] Poynter. (2018). 'Homepage', [Online]. Available: https://www.poynter.org/international-fact-checking-network-fact-checkers-code-principles (accessed April 23, 2018).

[17] Faktisk.no. (2017). 'Homepage', [Online]. Available: https://www.faktisk.no/ (accessed April 23, 2018).

[18] Snopes. (2017). 'Homepage', [Online]. Available: https://www.snopes.com/ (accessed April 23, 2018).

[19] BS4. (2017). 'Beautiful soup documentation', [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (accessed April 28, 2018).

[20] Google. (Jul. 2017). 'Custom search json/atom api', [Online]. Available: https://developers.google.com/custom-search/json-api/v1/overview (accessed November 21, 2017).

[21] Selenium. (2017). 'Homepage', [Online]. Available: https://www.seleniumhq.org/ (accessed April 28, 2018).

[22] Parsehub. (2017). 'Homepage', [Online]. Available: https://www.parsehub.com/ (accessed November 21, 2017).

[23] R. T. Fielding. (2000). 'Architectural styles and the design of network-based software architectures', [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/abstract.htm (accessed April 25, 2018).

[24] Google. (2017). 'About chrome extension', [Online]. Available: https://developer.chrome.com/extensions (accessed November 7, 2017).

[25] H. Brody. (2017). 'I don't need no stinking api: Web scraping for fun and profit', [Online]. Available: https://blog.hartleybrody.com/web-scraping/ (accessed November 7, 2017).

[26] P. Kononow. (2017). 'What is metadata', [Online]. Available: https://dataedo.com/blog/what-is-metadata-examples (accessed May 4, 2018).

[27] MedieNorge. (2018). 'Lesertall for norske nettaviser - resultat', [Online]. Available: http://www.medienorge.uib.no/statistikk/medium/avis/253 (accessed May 10, 2018).

[28] StatCount. (December 2017). 'Browser market share worldwide', [Online]. Available: http://gs.statcounter.com/browser-market-share (accessed January 23, 2018).

[29] Django. (). 'Homepage', [Online]. Available: https://www.djangoproject.com/ (accessed January 17, 2018).

[30] F. Ximenes. (October 2017). 'Celery: An overview of the architecture and how it works', [Online]. Available: https://www.vinta.com.br/blog/2017/celery-overview-archtecture-and-how-it-works/ (accessed February 26, 2018).

[31] Redis. (2018). 'Redis homepage', [Online]. Available: https://redis.io/ (accessed May 13, 2018).

[32] A. Mueller. (2013). 'Word cloud github page', [Online]. Available: https://github.com/amueller/word_cloud (accessed May 14, 2018).

[33] Skrivesenteret. (2018). 'Skrivesenteret', [Online]. Available: www.skrivesenteret.no/uploads/files/Bindeord_avsnitt.docx (accessed May 14, 2018).

[34] S. ISO. (2016). 'Norwegian stopwords', [Online]. Available: https://github.com/stopwords-iso/stopwords-no/blob/master/stopwords-no.txt (accessed May 14, 2018).

[35] K. Melvær. (2014). 'Norske stoppord', [Online]. Available: https://gist.github.com/kmelve/8869818 (accessed May 14, 2018).

[36] G. Lionetti. (Sep. 2012). 'What is version control: Diffs and patches', [Online]. Available: https://www.atlassian.com/blog/archives/version-control-diffs-patches (accessed May 8, 2018).

[37] C. OKeefe. (February 2018). 'Modern web automation with python and selenium', [Online]. Available: https://realpython.com/modern-web-automation-with-python-and-selenium/ (accessed April 26, 2018).

[38] J. Tauberer. (2011). 'Diff$_{m}atch_{p}athgithubpage$', [Online]. Available: https://github.com/JoshData/diff_match_patch-python (accessed May 13, 2018).

[39] J.-e. van Bergen. (February 2018). 'What is mixed content', [Online]. Available: https://developers.google.com/web/fundamentals/security/prevent-mixed-content/what-is-mixed-content.

[40] R. Kandari. (December 2017). 'Sessions and cookies – how does user-login work?', [Online]. Available: http://www.theitstuff.com/sessions-cookies-user-login-work (accessed May 6, 2018).

List of Tables

List of Figures

List of source codes

Glossary

a/b testing A controlled experiment with two variants A and B to test for instance what button color makes the button more attriactive to click. Which of the variants the user sees is random. 48

AngularJS A Javascript framework by Google to create frontend applications. 13

API Application Program Interface is a set of functions, procedures or definitions to interact with the program. 7, 9, 10, 14, 15, 17, 49, 65

base64 A binary-to-text scheme that represents binary data as ASCII (American Standard Code for Information Interchange) string. 59, 60

BS4 Beautiful Soup is a Python library for pulling data out of HTML and XML files. 12, 29

Captcha Completely Automated Public Turing is a test to check if the one accessing the web page is a human or a robot. 13

Client Server Model A server is providing resources and services to one or more clients. 21

CPU A Central Processing Unit is the electronic circuitry within a computer that carries out the instructions of a computer program. 20, 51

CSS Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language. 10–12, 15, 31, 33, 52–54, 59, 60

CSS selector Using a string of HTML tags, attributes and CSS classes to describe the path to the HTML element. 8, 29, 31–34, 47, 60

Denial of Services Distributed Denial of Services is a cyber-attack where the incoming request from a user to a server is higher then it can process. This leads to the server being unavailable. 13

DOM The HTML DOM (Document Object Model) is an application programming interface (API) for valid HTML. 59, 61

GB The gigabyte is a multiple of the unit byte for digital information. One GB is 1,000,000,000B. 20

HTML Hypertext Markup Language is the standard markup language for creating web pages and web applications. 7, 10–13, 15, 20, 22, 27–30, 35, 36, 45, 47, 48, 60, 66, 72

HTTP Hypertext Transfer Protocol is a protocol distributed hypermedia information systems. Is the communication foundation of the internet. 13, 59, 60

HTTPS Hypertext Transfer Protocol Secure is an extension of HTTP where the data between the client and server is encrypted. 59, 60

Internet Protocol Internet Protocol address is a numerical address assigned to each device connected to a network. 13, 50, 60

ISO 8601 International standard for exchange of data and time related data (yyyy-mm-dd for date and hh:mm:ss for time). 29–31

Javascript A high-level interpreted programming language. 13, 15, 61

MB The megabyte is a multiple of the unit byte for digital information. One MB is 1,000,000B. 20

Native ads Advertisement that looks like a normal headline. 34, 40

ORM Object Relation Mapper is a technique for converting data between objects of incompatible types systems. 26

Paywall A virtual wall between the user and the article, which only goes away if the user has a subscription for that site. 13, 20, 66

Python A interpreted high-level programming language. 28, 29

React A Javascript library by facebook for creating user interfaces. 13

robots A term for all non human interaction with a website. Includes amount other web scrapers, crawlers and spiders. 38, 49

RSS feed Rich Site Summary feed is a type of web feed which enables the user to access and get updates of content in a standardized format. 7

SQL Structured Query Language is a language designed for managing data in relational databases. 26

stop words Common words of little to no value to describe the sentence. 28, 59, 65

supported site A site is supported if the database has a record of it. 23, 31, 36

union The union of two texts a and b is the resulting content of merging both, but removing duplicates. 34

URL Uniform Resource Locator. Term for the address of a website, i.e https://www.example.com. 5, 8, 11, 31, 32, 34, 36, 38, 39, 43, 60, 66

UTC Is the primary time standard by which the world regulates clocks and time. 30

Web API Is a HTTP endpoint for accessing resources. 12, 13

Word cloud A visual representation of words from a text where the size of the word represent number of occurrences in the text. 4, 18, 20, 23, 24, 27, 28, 37, 38, 40, 43, 46, 59, 62

XML Extensible Markup Language defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. 12

Zipped ZIP is a file format for archiving one or more files into a single file. 15

Appendices

A Browsers market share Dec 2017

Table A.1: Browser distribution on desktop worldwide as of Dec 2017

Browser	Market share	Extension
Chrome	64.72 %	Yes
FireFox	12.21 %	Yes
Internet Explore	07.71 %	Yes
Safari	06.29 %	Yes
Other	09.07 %	No data

Table A.2: Browser distribution on mobile worldwide as of Dec 2017

Browser	Market share	Extension
Google	49.72 %	No
Safari	18.34 %	No
UC Browser	15.76 %	No
Opera	05.70 %	Yes
Firefox	00.80 %	Yes
Other	10.48 %	No data

Table A.3: Browser distribution on tablet worldwide as of Dec 2017

Browser	Market share	Extension
Safari	59.48 %	No
Google	24.01 %	No
Android	12.65 %	No
Other	03.86 %	No data