System for downloading up to date webpages for offline viewing

Final Year Report

Cardiff University

School of Computer Science & Informatics

CM3203 ONE SEMESTER INDIVIDUAL PROJECT

Markuss Baumgarts
C1646173

Supervised by: Padraig Corcoran
Moderated by: Luis Espinosa-Anke

Abstract

The need for constant access to the web, forces us to pay mobile providers for data plans when Wi-Fi networks are not available. The project aims to to develop a cross-platform desktop solution that enables its users to save any public website on their local machine for later offline viewing. The goal of the project is to provide a solution that improves upon the already available similar products on the market. It explores the necessary methods of successfully downloading and displaying a live webpage locally on the users' machine, whilst also explains the limitations constraining similar projects reaching their optimal performance. As a result, a minimum viable product was achieved which was able to successfully download and capture most of the tested webpages, with some exceptions. Satisfying the set out goal of the project whilst also leaving room of improvement for further development.

Table of Contents

Introduction

It is no secret that arguably we have become dependent on the connection and the resources that the internet provides for us. That's why nowadays, we have Wi-Fi networks covering practically the whole country. However, at times when we are commuting or are out in the countryside we may not have access to these networks. And consumers have shown desire for this constant access to these resources by paying mobile network providers a considerable sum for data plans. "In about 95 percent of emerging markets, people rely almost entirely on expensive prepaid mobile data, and many can only afford 250MB of data per month" [16]

To tackle this problem, a few years ago, almost everyone stored all their media content locally on their mobile devices, which they transferred through USB from their home computer. But as data is moving towards the cloud, this local storage method is slowly dissipating. Now, we have Spotify, Netflix and other content providers that give us instant access to their content. But even these services recognize the need to download their content for local access. That's why both of these platforms provide an offline mode, where you can download and access their content on your device.

However, I have not seen a user-friendly solution for website viewing offline. Recognizing the demand for constant access to data, this project will detail the process and evaluation of creating such a program.

The overall aim of this project is to develop a cross-platform desktop solution that enables its users to save any public website on their local machine for later offline viewing. The goal is to provide a solution that improves upon the already available similar products on the market, which after researching said products it was discovered that their performance was left wanting.

Although web scraping is a relatively well known and simple concept, this project does not aim to extract a single piece of information from a webpage. But to clone the whole page and display it as faithfully as possible to the original version. The possible reason why the already available solutions are not ideal, is because to actually capture a website and display it locally as it would be shown on its server is no small undertaking. There are multiple issues you have to consider to access all the rich content on the website. Even if one tries to accommodate all the various different media variations and structures that you can encounter on the web. There is no one full proof way on doing so and most likely all the possible different cases will have to be coded explicitly. For example, not all of the images are embedded in the <img> tag, as some of them are displayed alternatively, and the source of the file either defined in the JavaScript or in the styling of the website, requiring tedious resource scrapping to access the image. Additionally, there are multiple other issues such as: dynamically loaded data which is only provided once a user has triggered a specific event e.g. scrolling to a certain section of the page. And some websites also employ client side origin protection which aims to limit the origin from where the website can be accessed and launched from.

Initially the scope of the project was to provide a robust solution to all available public websites including such edge cases as social media platforms e.g. Facebook, Twitter, etc. But after gaining a better understanding of the task and problems at hand. The scope of the project was altered. In the end, the scope was readjusted to properly be able to download and display the bbc.co.uk website which due to its complex and rich content, would serve as a good benchmark for most of the applications one will encounter.

Background

2.1 Justification of the chosen target platform

There were three possible target platforms that were considered and would be suited for this kind of a project:

- Mobile application
- Chrome extension
- Desktop application

Due to the plethora of possible use cases that such an application might have on mobile - the project initially was heading towards the possibility of it being created for the mobile environment. Not only would users be able to access the websites when a connection is not available, but the application would enable them to save their mobile bandwidth on commonly accessed websites.

However, because of the projects nature, it would have to use a variety of native mobile elements e.g. write access to local storage. Meaning that a hybrid solution like the Ionic Framework would not be preferable in this scenario, because hybrid applications have a limited local storage space of 10MB. [1] Creating a native mobile application, given the time constraints and the scope of the project, would have been difficult to accomplish.

Google Chrome extension was another possibility. As such a solution would be conveniently "bundled up" in the browser application, much like Safaris reading list. But, disregarding the potential difficulties for local file management from a browser application – if the project had headed that route, the application would have only been exclusive to the Google Chrome users, thus alienating a lot of potential users.

Accounting for the ease of development, target audience, the chance of successfully finishing the application in time and the fact that multiple well known applications such as Spotify, Skype, Slack, WhatsApp, etc. use the electron framework for their desktop versions - it was decided to create a cross-platform, hybrid desktop application utilising the Node.js based Electron framework.

2.2 Existing solutions

Naturally the idea of saving websites locally for later use, isn't a completely novel proposition. Paid 3rd party solutions notwithstanding, the user already has multiple free and available solutions to save locally websites for later use.

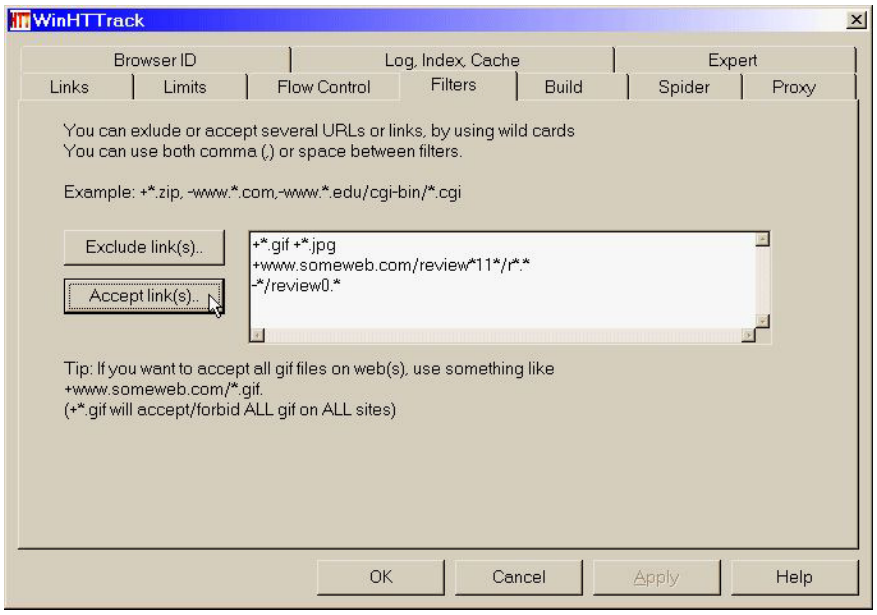
2.2.1 Built-in browser "Save-As" feature

Although most modern day browsers already provide a built in "Save Page as" feature which allows the user to download the index html and its associate files for any given website. Specifically, Safari users having the "reading list" option. It was discovered that there are number of areas that this feature falls short and where the projects proposed solution “fills in the gaps”.

a) Apart from safari's reading list, the browsers *save-as* feature does not provide a way to neatly organise your download content from one source. Instead the user is given an html file and an assets folder along with it, leaving the organization and structuring of the content up to the user.
b) It does not deal with dynamic content e.g. *lazy loading.* Which means that most content (specifically images) only load when the user has scrolled to that specific section of the page. If you were to take just a static html file, you would not be able to see the images that have not loaded yet. (refer to section **4.6.1**)
c) Some websites like http://www.reddit.com also employ CORS protection (refer to section **4.6.2**). Which doesn’t allow their websites to be display from a *file:///* prefix. An unavoidable issue if you are using the browser *save-as* feature, and then simply opening the downloaded index file from your local system.

2.2.2 HTTrack

A more robust already available solution is the HTTrack software. Although its performance is better than just the basic “save-as” option, the interface on Windows is outdated and complex for users.



You can exlude or accept several URLs or links, by using wild cards
You can use both comma (,) or space between filters.

Example: +*.zip, -www.*.com,-www.*.edu/cgi-bin/*.cgi

```
+*.gif +*.jpg
+www.someweb.com/review*11*/r*.*
-*/review0.*
```

Tip: If you want to accept all gif files on web(s), use something like
+www.someweb.com/*.gif.
(+*.gif will accept/forbid ALL gif on ALL sites)

Figure 1: Screenshot of the HTTrack windows GUI

Additionally, on UNIX based devices your only option to run the program is through the terminal. And this specific solution apart from the organizational issues also suffers from all of the aforementioned shortcomings of the browsers “save-as” features.

2.3 *Used Technologies*

- **Node.js** - is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux. [2] It is written in C++ and built upon the V8 JavaScript runtime. To extend the performance of Node.js applications, it allows the use of native modules (managed by the NPM – Node Package Manager) to create full pledged applications that are not limited to JavaScript, if a problem requires the use of a low-level solution. [3]

- **Electron Framework** - As it was decided to create a cross-platform desktop solution, the whole project is based upon the Electron Framework. Electron is an open-source framework based on the Node.js runtime, and thus lets you write cross-platform desktop applications using JavaScript, HTML and CSS. [3] Essentially Electron uses web page rendering (HTML + CSS) as the GUI of the application (by utilizing Chromium's rendering library on which Google Chrome is built on) and JavaScript as the logic behind it. With various C++ API modules that allow to extend the system with native operating system task capabilities (e.g. creating native dialogs, accessing the file system). [4]

- **Photon UI kit -** Photon is a 3rd party UI Toolkit for Electron which provides the GUI with native layout options. It comes with fonts and CSS styling classes, so that one is able to make their web application look like a native desktop application. The kit allows the developer to display native elements such as a native looking file explorer, application navigation, pane-view, etc.

2.4 Node.JS modules

As described earlier Node.js allows for the use of native modules. Hence, to be able to effectively implement the specified functionality of the project, a number of open-source node.js modules were used to save development time. E.g. using the "connect" or "Axios" module to create HTTP requests rather than writing the request from scratch or provide native functionality using the fs module to easily access the local file system.

- *Cheerio*

Cheerio parses mark-up and provides an API for traversing/manipulating the resulting data structure. It allows to load, parse and query the downloaded html documents by their HTML tags or CSS classes. [5]

- *Axios*

Axios is a promise based HTTP client for the browser and Node.js, which I used to make asynchronous http requests to access and download all of the necessary media files. [6]

- PhantomJS

PhantomJS is a headless web browser, which was used for page rendering, automation (i.e. scrolling to the bottom of the page) and screen capture.

- Ytdl-core

YouTube downloading module, which given a URL allows to download the source video from YouTube

- url

A built in node module which provides utilities for URL resolution and parsing. [7]

- connect

Extensible HTTP server framework for node [8]

- serve-static

Enables to serve static files from within a given root directory to serve html files through the application from the local host [9]

- fs

The fs module provides an API for interacting with the file system in a manner closely modelled around standard POSIX functions. [10]

- electron-prompt

Electron helper module to prompt for a value via input or select [11]

- rimraf

Essentially the UNIX command rm -rf for Node.js. Used to delete folders from the saved_websites directory [12]

2.5 ***Model-View-Controller (MVC) pattern***

The project will try to follow a Model-View-Controller (MVC) software design pattern. In short MVC is a design pattern that separates the applications logic from the user interface. MVC consists of three main components:

- Model – Which describes the behaviour regarding data, logic and rules. Essentially where most of the applications logic is defined
- View – Displays the system status to the user. The UI of the application
- Controller – Acts as the 'middleman' between the View and the Model. It is responsible for capturing user action events and then passing on the data to the model.

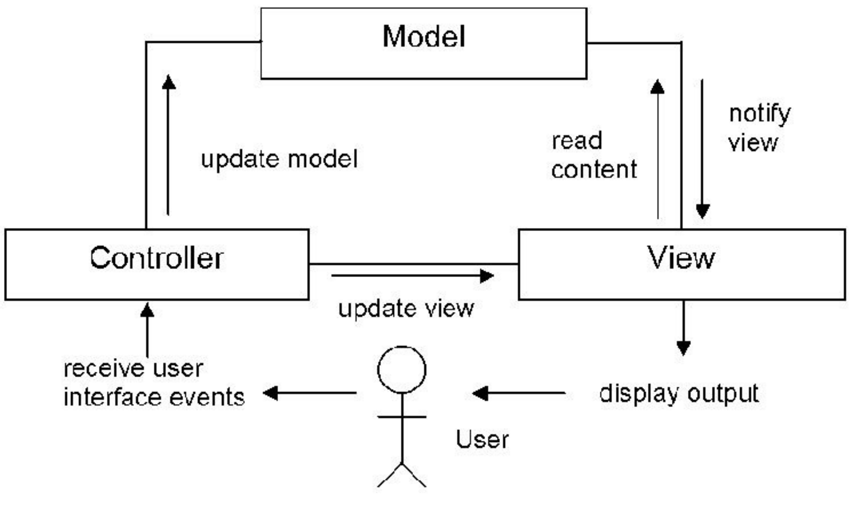


Figure 2: Overview of the MVC design pattern [13]

A simple analogy for the MVC design pattern would be that of a customer ordering food at a restaurant. The View is the menu, the controller is the server which takes your order and informs the model (kitchen) about it. Then the kitchen (model) is responsible for executing your order (request).

The reason for using a MVC design pattern for this project is that, it will have to deal with asynchronous http requests. It is important not to suspend the application whilst a new website is downloading. Meaning that whilst the model is busy downloading the website, the user can still interact with the application and open or edit the already stored web pages in the application. Although the use of MVC does increase the complexity of the application, if this project would be developed further with multiple views and added functionality, the MVC design pattern allows for a better scaling process, and code management.

2.6 *Restrictions and Assumptions*

To account for the difficulty of this project and that the system is going to be developed by a "one-man team" within a limited time frame:

- The system will assume that there will be no user errors on the application. E.g. the user will always provide valid links that lead to an accessible html document).
- The system will not be able to process websites, that require user authentication to access their content.
- The range of media types that the application will be able to successfully handle, will be limited to style sheet files, JavaScript files, embedded YouTube iframe videos, pictures in the <img> tag and background pictures for containers that have the image source defined inline.

Specifications and Design

To fully understand the project, what it is trying to achieve and what was achieved at the end, we must first specify the systems requirements and lay out the overall design of the proposed solution.

3.1 System Requirements

Requirements management is the essence of successful project outcome. Effective requirements management goes a long way of eliminating most design mistakes and reducing failures during the development process. That is why it is crucial to identify them at an early stage. Requirements are divided into two parts, Functional and Non-functional. The report follows the MoSCoW notation to group and prioritize the projects functional requirements.

3.1.1 Functional Requirements

M (must have):

- Given an URL as an input, the system must download the necessary html file and supporting asset files.
- The system must display the downloaded website in its User Interface

S (should have):

- *The displayed local website should have the same styling and arrangement as the online version*

- Users should be able to delete the downloaded websites

C (Could have):

- *The system could have an implemented web-crawler that gets the next 1-2 levels of linked URLS from the main URL, allowing the user to navigate to the same domain links.*
- *The system could have a primitive video handling mechanism for embedded video files.*
- *The system could have a built in internet connectivity listener, that updates the already downloaded websites once the user connects to the internet.*
- *The system could allow to query saved websites*

W (Won't have):

- *The system will not be able to access content that is protected with some kind of user authentication system.*
- *The system will not be able to accommodate the whole range of embedded media files available online e.g.* GitHub code snippets, flash containers, GIFs, etc.

3.1.2 Non-Functional Requirements

Non-functional requirements are used to govern the performance and the design of the system. They do not specify how the system will execute tasks, but they are an important aspect of any system design.

- The UI should have an intuitive minimalistic feel to it.
- The download of the webpage must not take longer than 10 seconds.
- The application should feel 'responsive' and not 'sluggish'

3.2 Website acquisition system

The underlying system will have to heavily rely on HTML parsing and asset source extraction to function as indented. The flowchart below provides a description of the basic system functionality the application should follow once the user has entered the URL for the website that they wish to download.

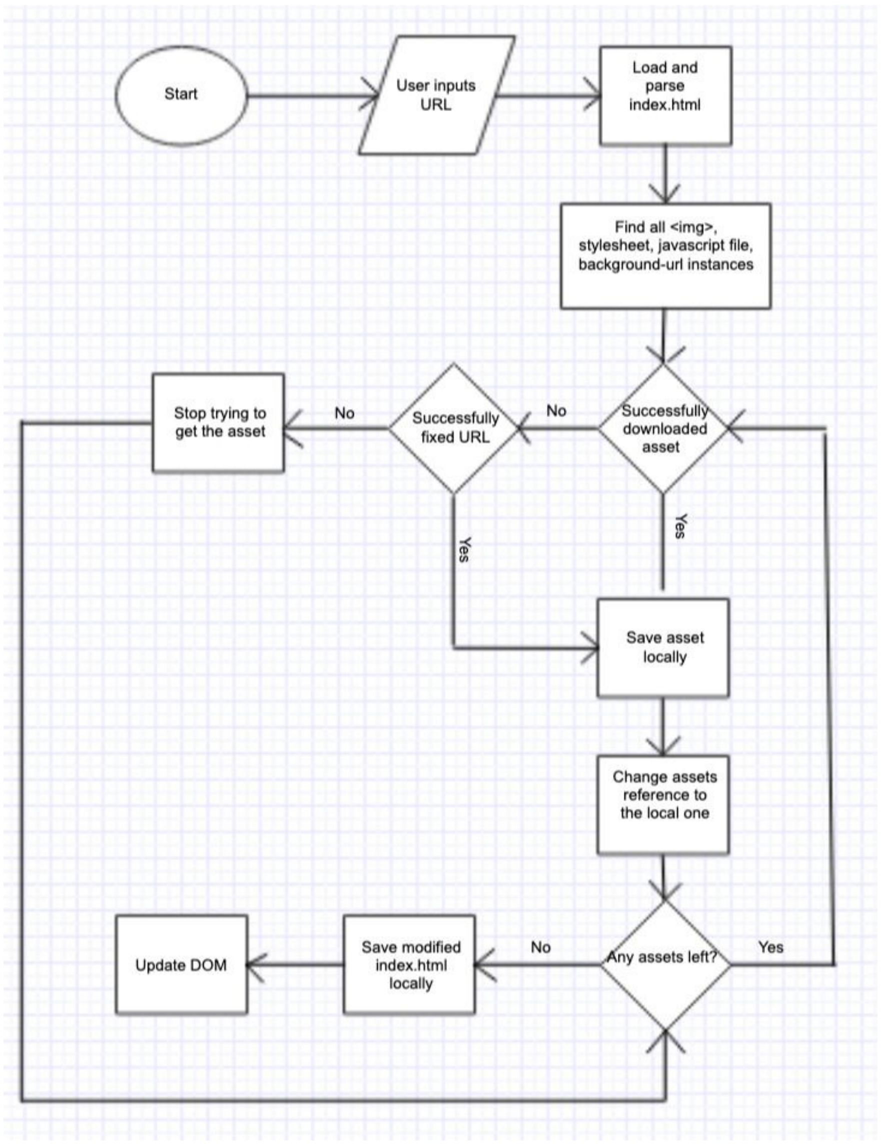

Start
User inputs URL
Load and parse index.html
Find all <img>, stylesheet, javascript file, background-url instances
Successfully downloaded asset
No
Successfully fixed URL
No
Stop trying to get the asset
Yes
Yes
Save asset locally
Change assets reference to the local one
Any assets left?
Yes
No
Save modified index.html locally
Update DOM

Figure 3: Flowchart of the underlying logic of the application

It is worth noting that this flowchart does not include the logic for web crawling the associated links from the main provided page. But as one can see the the chart itself is pretty self explanatory.

1. Once the user has entered the URL of the website that they wish to download, the system should then be able to parse the index HTML in a way so that, it is able to query original index file either by HTML tags or CSS classes.
2. Then it should try to find all the instances of explicitly specified media types. In this projects case those would be all the HTML <img>, <link>, <script>,

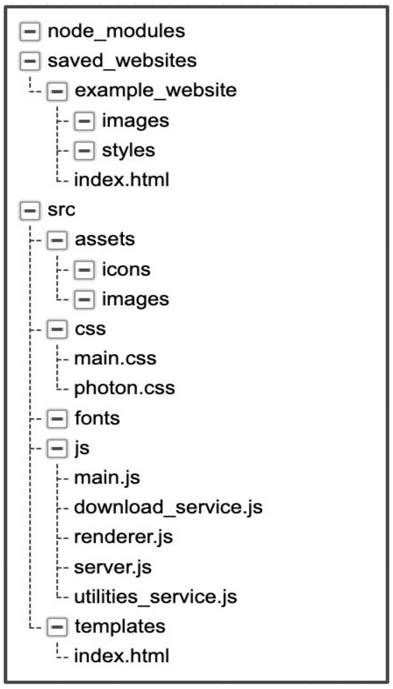
<iframe> and <div> tags and then try to extract the corresponding assets source URLs.
3. Once collected, the system should then try to download and save the assets to the local system. If the download of an assets fails because the URL is locally defined to the websites servers' directory or somehow differently invalid, then the system should attempt to fix the URL and try again. If the download fails after the system has tried to fix the URL, ignore the asset and continue until there are no more assets left.
4. Once the asset has successfully been downloaded to the local system, its important to change the reference to the asset in the main index HTML file to the local one.
5. Finally, once all the assets have been handled, the system should then save the modified index HTML file locally and update the applications DOM displaying to the user the newly acquired webpage.

3.3 Project Structure

Generally, once you create a Node.js application you are greeted with a very basic file structure that most Node.js projects follow. In essence you have the following three files:

- a JavaScript file "index.js" - which is responsible for all the functionality of the application.
- A HTML "index.html" file along with the corresponding style sheet, which provides the user Interface of the application.
- And a "package.json" file which holds all the metadata of the project and a list of the additional node modules that have been added to the project.

However, as this project is following the MVC design pattern, the application was split into multiple different services, each responsible for a different part of the applications functionality.
The resulting file hierarchy and the structure of the project is as follows:



```
[-] node_modules
[-] saved_websites
 └- [-] example_website
     ├- [-] images
     ├- [-] styles
     └- index.html
[-] src
 ├- [-] assets
 │   ├- [-] icons
 │   └- [-] images
 ├- [-] css
 │   ├- main.css
 │   └- photon.css
 ├- [-] fonts
 ├- [-] js
 │   ├- main.js
 │   ├- download_service.js
 │   ├- renderer.js
 │   ├- server.js
 │   └- utilities_service.js
 └- [-] templates
     └- index.html
```

Figure 4: Overview of the projects file structure

Ignoring the *saved_websites* directory, which is there to store and organise the local files of all the downloaded websites.

Under *src* we have the:

- "assets" folder, which holds all the local assets for the application, such as icons for buttons and images for placeholders.
- The *css* folder contains the "main.css" style sheet, which is the custom code used to represent the user interface and "photon.css" which holds all the classes for the aforementioned Photon UI Kit.

Under the *js* directory, we can see that the functionality of the system is divided into five main files or services:

- Main.js holds all the boilerplate electron code which sets up the application and its native events, such as restricting the window size, defining the main HTML file location, native event listeners and so on.
- Renderer.js is the bridge between the UI and the apps functionality (The Controller of the MVC design pattern). It listens for user input, and depending on the required action, either passes it further to the next service (e.g. submitting an URL as an input to the text box, the input URL would be passed to download service afterwards) or executes minor actions that would change the current state of the UI, such as deleting a website from the system within its own scope.
- The main chunk of the work is done in the download_service.js (The model of the MVC design pattern). Once it receives the passed URL from renderer.js it is responsible for processing the URL, getting the index file, the associated assets such as images, style sheets, etc. and managing the the local references and files.
- Server.js is responsible for creating a local host server that all of the websites are served from, to circumvent the CORS issue (refer to **4.6.2**)
- Utilities_service.js holds various helper functions such as regex string extraction, folder creation, etc.
- The template folder, currently only holds the main index.html (The View of the MVC design pattern) file, but if there would be a need to add additional views to the application, that is where they would be defined.

3.4 Process of adding a website to the system

To further reiterate how the different parts of the applications communicate with each other. The UML sequence diagram below expands upon the previously discussed flow chart on the how the system would download a website. And aims to explain in greater detail how the MVC design pattern is applied to the application, outlining the interaction with its three main components.

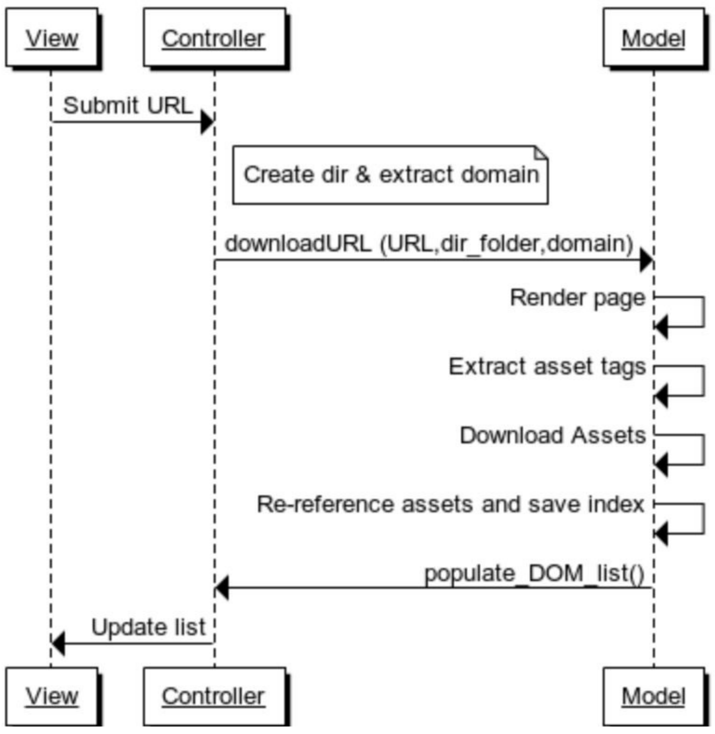


Figure 5: UML sequence diagram of adding a website to the system

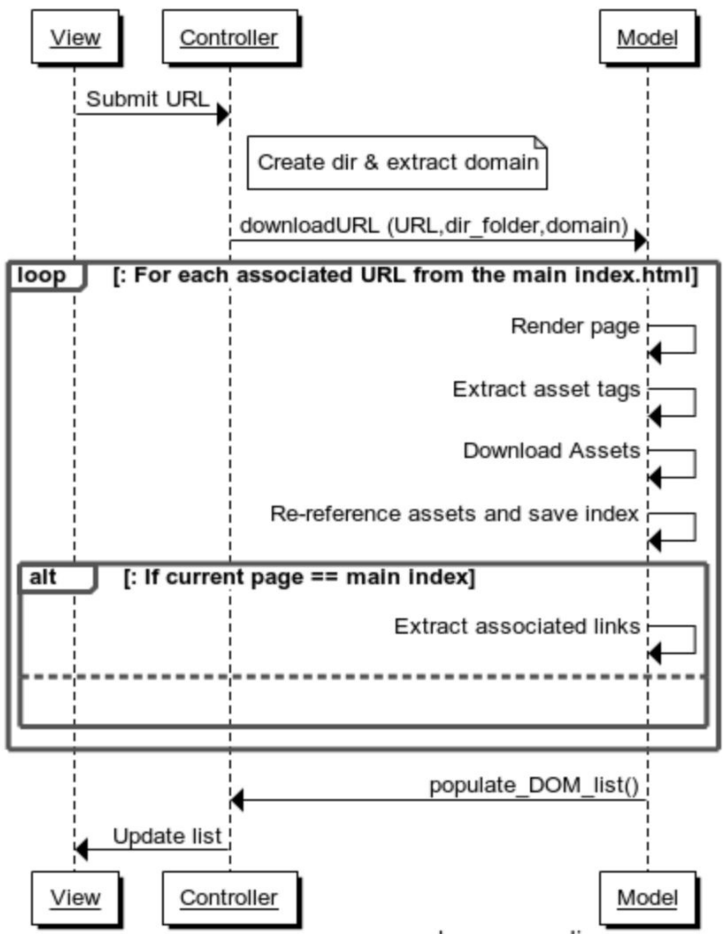
Once the user has entered and submitted the URL from the View. The controller aims to extract the base domain of that URL.

For the URL "*https://www.pcmag.com/encyclopedia/term/43329/flowchart*" the extracted domain would be "*www.pcmag.com*".

Then the system creates a folder with the domain as the folder name and passes all of this information to the Model for the downloading process of the webpage. Once the Model gets called with those parameters it follows the previously explained logic in the flowchart. Once the webpage has finished downloading, the Model calls the Controller by the *populate_DOM_list* function, which checks all the entries in the saved_websites directory and creates or updates the list of websites that are then displayed in the View. Indicating to the user that all of the processes have finished and that the webpage has been successfully downloaded.

3.4.1 Process of adding a website to the system with web crawling

If level 1 web crawling is used. Level 1 meaning that only the associated links of the main, submitted URL will be crawled and not the associated links of the crawled pages. Then the system processes would be relatively the same, with the exception of recursively calling the main download function for all the linked URLs in the main index HTML file.


View
Controller
Model
Submit URL
Create dir & extract domain
downloadURL (URL,dir_folder,domain)
loop
[: For each associated URL from the main index.html]
Render page
Extract asset tags
Download Assets
Re-reference assets and save index
alt
[: If current page == main index]
Extract associated links
populate_DOM_list()
Update list
View
Controller
Model
Figure 6: UML sequence diagram of adding a website to the system with level 1 web crawling

Once the URL has been successfully downloaded, before calling the *populate_DOM_list* function, the application would first check if the current page handled is the original URL submitted by the user, and aim to extract all the <a> href links from that page. Then it would loop through all the found links that have the same domain as the original URL, and call the *downloadURL* function again with a different URL, directory folder but the same domain. Until all of the links have been downloaded. Only then would the system call the *populate_DOM_list* function to update the view and let the user know that everything has finished downloading.

3.5 User Interface

It is important for the User Interface to have a clean, minimalistic look, with as few options as possible to not confuse the user the first time they see the application.

Since any application built upon the electron framework in its essence is basically a webpage. All of its UI is created through HTML and CSS. However, as the project is meant to be a desktop application, regular CSS styling looked too much like a website and not a desktop application. That's why an electron specific UI kit, Photon was used. Photon kit came as a separate CSS file with predefined classes and its own font package, which allowed to significantly cut down on development time on the front-end of the application as well as provided a native feeling to the overall UI.

Minimum windows size of the application was restricted to be precisely 800x600 pixels, so that the design would not have to account for screen sizes smaller than that.

3.5.1 Wire Frame

Before starting with the actual implementation, it is important to at least create a rough sketch how the application should look using wireframes. Since most of the functionality is either happening in the background or in the actual browser, it was important to keep the design to a 1-page application.

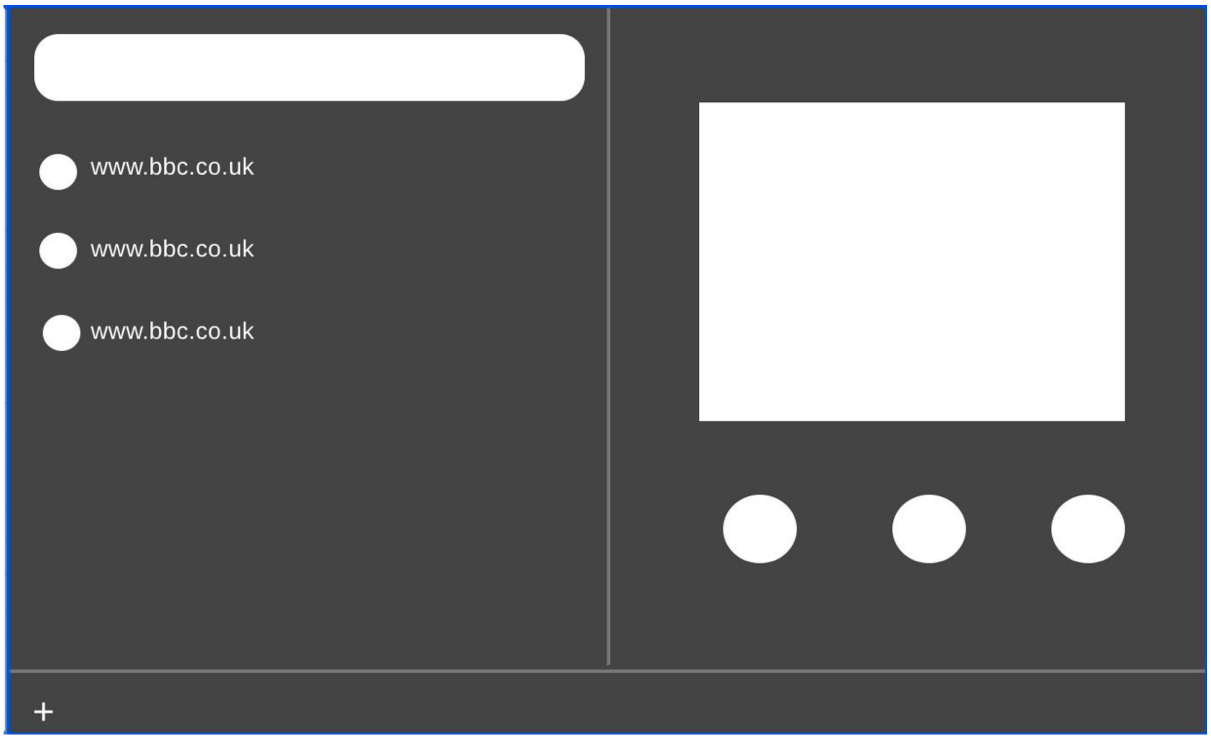


Figure 7: Wire-frame design of the application

As previously mentioned, this wire frame is an example of minimalistic design, which allows the user to easily understand the state of the system and does not confuse the user with a multitude of baffling and intimidating options or parameters. Right now, the design is a simple 2-pane display.

One the left side the user can see and search the list of the available downloaded websites that they have added. And the right pane changes depending of the selected website from the left pane.

The right pane shows the selected websites name, its preview of the form of a screenshot, and 3 buttons beneath: Open, Refresh, Delete.

Implementation

Now that it has been defined how the system should function, behave and look, we shall examine the actual implementation of the system in question.

4.1 UI implementation

As mentioned before, the whole UI is akin to a website, implemented solely by HTML, CSS and utilising JavaScript to dynamically display the state of the system.

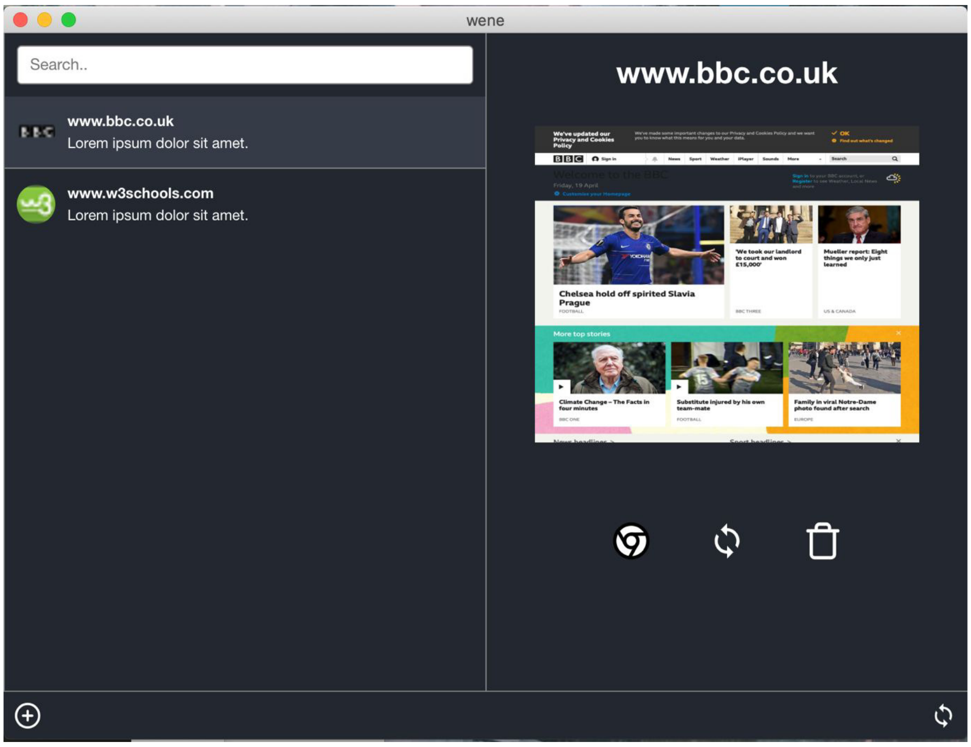


Figure 8: [Screenshot] Final design of the application

The list on the left is generated from the controller. It lists all of the folders in the saved_websites directory, and then iterating through the found folders a list item is created for each entry and then added to the unorganized list:

```
website_array.forEach(function (item, index) {
        var li = document.createElement('li');
        li.setAttribute("class", "list-group-item");
        li.setAttribute("id", index);

        ul.appendChild(li);
        li.innerHTML +=
             '<img class="img-circle media-object pull-left" src="' + item.favicon + '" w
idth="32" height="32">' +
             '<div class="media-body">' +
             '<strong>' + item.title + '</strong>' +
             '</div>';
   });
```

The right pane is displayed in a similar way. Whenever the 'click' event is triggered on any of the list items. The right pane is changed accordingly to match the index of the clicked website.

```
//highlight selected item
target.setAttribute("class", "list-group-item selected_item");
let selected_item = website_array[target.getAttribute("id")];
//change the preview
document.getElementById("preview_title").textContent = selected_item.title;
document.getElementById("thumbnail").src = selected_item.thumbnail;
```

4.2 Main download function

Once the user enters the URL, through the controller that URL is then passed to the *download_services* service downloadURL function.

```
// Variable assigned to the downloaded html through the cheerio module
// so that it would be possible to 're-appoint' external sources to local ones
var localHtml = null;

const downloadURL = (URL, dir_folder, hostname, calledFromCrawler = false) => {

  (async function () {
    const instance = await phantom.create();
    const page = await instance.createPage();
    var vWidth = 1080;
    var vHeight = 1920;
    await page.property('viewportSize', { width: vWidth, height: vHeight });

    const content = await page.property('content');
    localHtml = cheerio.load(content);
```

The code snippet above, first calls an anonymous asynchronous function, that initializes the phantom headless browser instance and sets the configuration for the website to load in.

It is important to note here the global variable *localHtml*, is set to equal to the cheerio object, which parses the accessed HTML file and will be used to access and modify all of the necessary index.html elements. Once the cheerio object is loaded, all of the asset handling functions are executed (explained in more detail in section **4.3**). Finally, once all the necessary assets have been acquired and their references changed to local ones in the index file. The new edited HTML file is saved to the system.

```
file_system.writeFile(dir_folder + '/index.html', localHtml.html(), (success) => { });
```

4.3 Asset handling

If all that was needed, would be the HTML index file, the application would already be finished. However, for the web page to run locally as it would run online, assets such as images, videos, JavaScript files and style sheets are also needed.

The first thing needed to access all of these assets is to parse them from the HTML file by accessing the parsed cheerio object localHtml, and extracting from it the necessary elements:

```
handleStyle(localHtml("link"), dir_folder);
```

```
handleImages(localHtml("img"), dir_folder);
handleBgImages(localHtml("div"), dir_folder);
handeJavascript(localHtml("script"), dir_folder);
handleVideo(localHtml("iframe"), dir_folder);
```

As all of these functions are relatively similar to each other, and they only differ in the source attribute that's being handled. As an example this report will only explain the *handleImages* function.

```
function handleImages(images_raw, dir_folder) {

  dir_folder = dir_folder + "/images";

  if (!file_system.existsSync(dir_folder)) {
    file_system.mkdirSync(dir_folder);
  }

  images_raw.each(function (index, images) {
    if (localHtml(images).attr("src")) {
      let image_url = checkURL(localHtml(images).attr("src"));
      if (image_url == null) {
        //create a placeholder for file not found;
      }
      let extension = get_url_extension(image_url);
      let file_name = dir_folder + "/" + index + "." + extension;

      downloadAssets(image_url, file_name);

      let local_html_name = "images/" + index + "." + extension;
      localHtml(images).attr("src", local_html_name);
    }
  });
}
```

The function has two parameters: images_raw (which is an array of all the extracted <img> elements) and dir_folder (the current directory folder. Which in this instance its current value would be *./saved_websites/www.example_website.com*).

Firstly, the system creates a new folder to store all the created images. Then the code iterates through the images raw array and checks if the "src" attribute is valid.

To properly save the image, one needs to know the proper extension .png, .jpeg, etc. of the image, otherwise saving with the wrong extension might result in a corrupted file.So the extension is extracted by using a get_url_extension helper function from utilities.js

```
function get_url_extension( url ) {
    return url.split(/\#|\?/)[0].split('.').pop().trim();
}
```

Finally, the extracted URL along with the file path where to save the images is passed to the *downloadAsset* function, where the image is downloaded and saved locally on the system. Then the global cheerio variable (which is essentially a parsed index.html instance) is called to change the original reference to the image to the relative local one.

4.4 Web Crawling

One of the biggest gaps in the current solutions on the market, is that you can not navigate away from the specified page that you have downloaded. To rectify this, this project tried to implement a web crawling algorithm, that takes all the *href* links in the anchor tags of the original URLs page source and downloads the web pages associated to those links. The only problem is determining how far should one go crawling the associated links. It was originally assumed that the application would be able to go at least 2-3 levels deep. However, once only the first level was implemented the download time of bbc.co.uk increased substantially. But even, if we only decide to crawl just through the first level of associated links from the original URL, it still takes a substantial amount of time, to download all of the links.

For http://www.bbc.co.uk/ , at the time of testing there were 160 associated links found. A portion of the found links were links that the users usually never click e.g. privacy policy. But filtering out the relevant links that the user might find useful would be outside the scope of this project. Additionally, to try to decrease the crawling time, it was decided to only crawl associated links that have the same domain as the source website. So this implementation will only explore getting the “first level” of associated links from the source website if those links are on the same domain as the source.

```
function collectInternalLinks(anchor_tags, dir_folder) {
  displayDownloadProgress("Listing linked URLs");

  let internalLinks = [];

  anchor_tags.each(function (index, link) {
    var found_link = localHtml(link).attr("href");

    if (found_link == null || getHostName(found_link) != hostName) return;

    internalLinks.push(localHtml(link));
  });

  linked_url_count = internalLinks.length;
  getInternalLinks(internalLinks, dir_folder);

}
```

For each of the found anchor tags if the source link isn’t null or not from the same domain as the original URL, then add that link to the *internalLinks* array.
When all of the links are collected the *getInternalLinks* function is called:

```
function getInternalLinks(internalLinks, dir_folder) {

  let regexSlpit = new RegExp("(" + hostName + ")");
  internalLinks.forEach(function (foundUrl, index) {

    let url = foundUrl.attr("href");
    displayDownloadProgress("Gettign linked URL: " + url);

    var dir_location = "/crawled_sites" + url.split(regexSlpit)[2];
    var relative_dir_folder = dir_folder + dir_location;
    if (!file_system.existsSync(relative_dir_folder)) {
      shell.mkdir('-p', relative_dir_folder);
    }
14.     var refrenced_location = "/" + hostName + dir_location
15.     localHtml(foundUrl).attr("href", refrenced_location + "/index.html");
16.     downloadURL(url, relative_dir_folder, hostName, true);
17.   });
18. }
```

Similarly, to the previously discussed *handleImages* function. The above code iterates through all the previously collected anchor tags. Accesses their href attribute to get the URL of the associated website. Then from line 9-14 creates a folder where to store all the downloaded files. After the folder is created, it changes the source link in of the specific <a> tag to the local destination and then recursively, calls the *downloadURL* function.

The *downloadURL* function has a calledFromCrawler parameter, which defaults to false, if only 3 parameters are passed to the function.

```
const downloadURL = (URL, dir_folder, hostname, calledFromCrawler = false)
```

This is to ensure that crawling is executed only for the main page and not the associated pages when they call the same downloadURL function.

```
if (!calledFromCrawler) { collectInternalLinks(localHtml("a"), dir_folder); }
```

Once this function has executed, all the subsequent calls to the *downloadURL* function, regarding the gathering of associated links will have the *calledFromCrawler* parameter set to TRUE. Thus making sure that only the main websites associated links are being crawled.

4.5 Error Handling

As this project is unlikely to be viewed by the general public, to save on development time, the project was developed under the assumption that the passed URL will always be valid and it was decided not to spend any time in handling user error cases.
But since, multiple http calls are involved in the process of cloning a website. Failure to handle errors when a found assets URL is invalid might be considered a core error in the functionality of the whole project.

The largest point of failure when trying to gather all the assets from the webpage, was that sometimes the assets were locally or weirdly referenced e.g.

//ichef.bbci.co.uk/images/ic/$recipe/p07735m0.jpg

is an invalid link, but once the first two characters are removed it becomes a working URL.
Or

/images/ic/$recipe/t0234n34.jpg

is a locally references link, to make it valid, the host name has to be added in front of it. In this case a valid image link would be

htttps://www.bbc.co.uk/images/ic/$recipe/t0234n34.jpg

That's why the *checkURL* function was implemented. It checks whether or not the link resembles the previous examples of broken links and tries to fix them by either adding the https prefix or adding the domain name to transform the local link to a remote URL.

```
function checkURL(URL) {

  if (isUrlValid(URL)) return URL;

  let tempURL = "https://" + URL.substring(2);

  if (isUrlValid(tempURL)) return tempURL;

  tempURL = "https://" + regexDomainExtraction(original_URL) + URL;

  if (isUrlValid(tempURL)) return tempURL;

  return null;
}
```

The *isUrlValid* function is a regex expression that checks if a given URL is valid.

```
function isUrlValid(userInput) {

var regexp = /(ftp|http|https):\/\/(\w+:{0,1}\w*@)?(\S+)(:[0-9]+)?(\/|\/([\w#!:.?+=&%@!
\-\/]))?/
 return regexp.test(userInput);
}
```

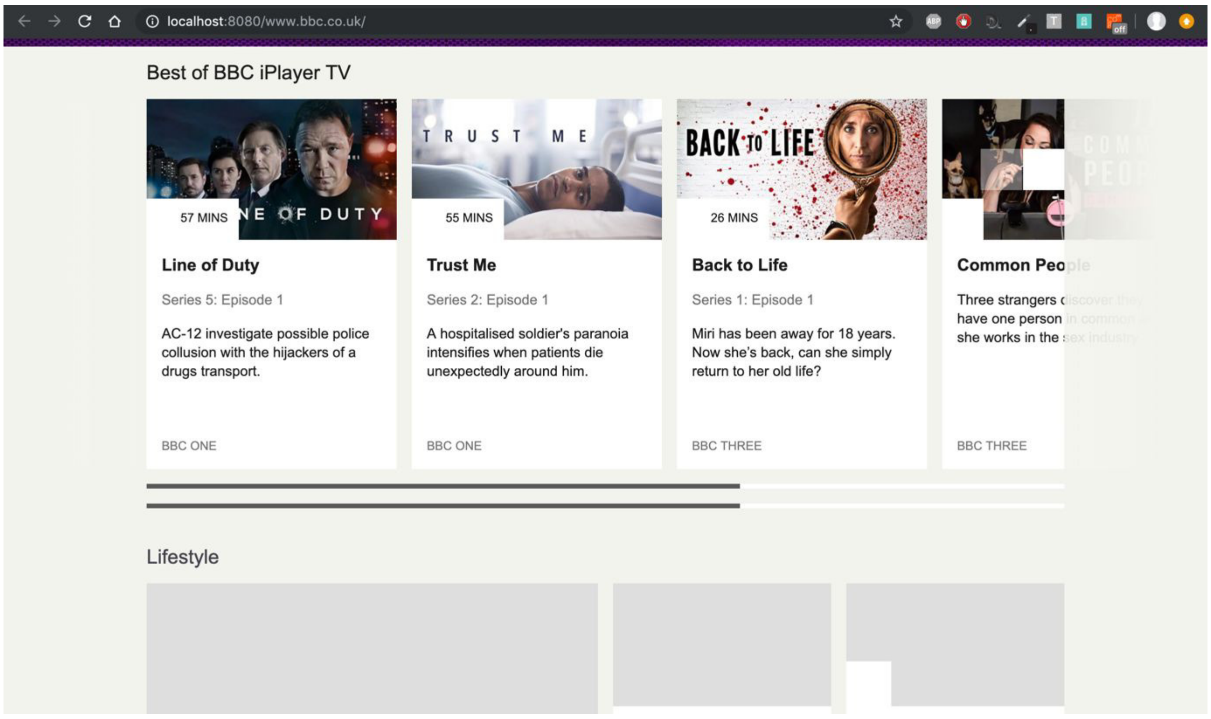
The issue with this approach is that, every possible case of a broken link has to be explicitly programmed.

4.6 Unforeseen problems

When first hearing of the project, I assumed that it was a fairly straightforward task, with the biggest possible difficulty being capturing Facebook pages or acquiring videos from YouTube. However, once the development of the application started, it was clear that even getting one of the most fundamental webpages such as bbc.co.uk would prove to be a difficult undertaking.

4.6.1 Lazy-Loading issues

The first issue was that most of the tested websites used lazy-loading. Lazy-loading is used to decrease bandwidth and loading times by only requesting the necessary assets when they are actually needed. Meaning that the content of the page would only load once the user has scrolled to that segment of the page. The problem arises if one tries to capture the html of said website. They will only be able to get the assets for the first segment of the page that has loaded.


localhost:8080/www.bbc.co.uk/

Best of BBC iPlayer TV

57 MINS
Line of Duty
Series 5: Episode 1
AC-12 investigate possible police collusion with the hijackers of a drugs transport.
BBC ONE

55 MINS
Trust Me
Series 2: Episode 1
A hospitalised soldier's paranoia intensifies when patients die unexpectedly around him.
BBC ONE

26 MINS
Back to Life
Series 1: Episode 1
Miri has been away for 18 years. Now she's back, can she simply return to her old life?
BBC THREE

Common Peo
Three strangers
have one person
she works in the
BBC THREE

Lifestyle

Figure 9: [Screenshot] Example of failing to capture lazy loading images

The solution to this problem was to emulate user behaviour that would scroll down to the end of the page. This required a headless browser such as PhantomJS which allows to be programmed to emulate such action as fill in the input boxes if one knows the ID of the input box or more importantly in this case - to simulate scrolling.

```
var scrollBottom = page.evaluate(function () { return document.body.scrollHeight; });
await page.property('scrollPosition', { top: await scrollBottom, left: 0 });
```

The above code snippet demonstrates how simulating that the user has scrolled to the bottom of the page was achieved. Firstly, the page.evaluate() function returned the rendered pages scrollHeight. The scrollHeight property returns the entire height of an element in pixels, including padding, but not the border, scrollbar or margin. [14] Then on the rendered page, the code sets the scrollPosition attribute to the returned scrollHeight property, which allows to scroll to the bottom of the page.

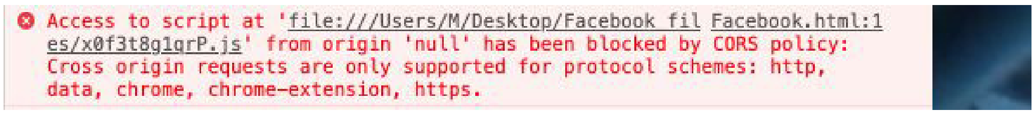
Although this solution wouldn't be sufficient enough for a website that utilises vertically scrollable pagination, meaning that additional content is added at the end of the page once the user has scrolled to the bottom. It is still a viable solution to the presented problem.

4.6.2 *CORS*

Additionally, some websites had implemented a Backend Cross-origin resource sharing (CORS) restriction. CORS is a mechanism which aims to allow requests made on behalf of you and at the same time block some requests made by rogue JS and is triggered whenever you are making an HTTP request to:

- a different domain (e.g. site at example.com calls at anotherexample.com)
- a different sub domain (e.g. site at example.com calls api.example.com)
- a different port (e.g. site at example.com calls example.com:3001)
- a different protocol (e.g. site at https://example.com calls http://example.com [15]

In this case CORS refused to load certain assets if the website was launched locally from the file:// prefix. Which happens by default if you simply try to open your HTML file on your local machine. These websites would return an error which would look something like this.



```
Access to script at 'file:///Users/M/Desktop/Facebook fil Facebook.html:1
es/x0f3t8g1grP.js' from origin 'null' has been blocked by CORS policy:
Cross origin requests are only supported for protocol schemes: http,
data, chrome, chrome-extension, https.
```

Figure 10: [Screenshot] Blocked CORS policy error message

There were two options to bypass this issue. One was to not acquire any of the JavaScript files, but that would limit the functionality of the downloaded webpage. The other was that the application couldn't simply launch the web pages from the local machine, it had to create a local host server, and then serve that html file through the created local host server on the users' browser.

```
var connect = require('connect');
var serveStatic = require('serve-static');

const server_url = "localhost:8080/saved_websites/";
connect().use(serveStatic('./saved_websites/')).listen(8080, function(){
    console.log('Server running on localhost:8080...');
});
```

All the saved websites will be served through the URL *http://localhost:8080/saved_websites/example_website/index.html*

As this URL has the http:// prefix the CORS policy wont block it.

4.6.3 *Images as container backgrounds*

Another unforeseen issue was that some websites like the bbc.co.uk for example, don't use <img> tags for their image content but they create a <div> container and using inline style tags add the desired image as a background-image.

```
<div class="top-story__image" data-image-recipe="//ichef.bbci.co.uk/images/ic/$recipe/p076lhnj.jpg" style="background-image: url("//ichef.bbci.co.uk/images/ic/1040x585/p076lhnj.jpg");" data-reactid=".ksq93l3yug.0.1.0.$hp-ts-0.0.0"></div>
```

Figure 11: [Screenshot] a snippet form bbc.co.uk source code

Because of this a new function *handleBgImages* was needed that goes through every single <div> in the given HTML file and searches every inline style property of those divs where the CSS rule "background-image" is used.

```javascript
const handleBgImages = (backgroundImg_raw, dir_folder) => {

  dir_folder = dir_folder + "/images";
  backgroundImg_raw.each(function (index, div) {
    if (div.attribs.style) {
      let style_def = div.attribs.style;
      if (style_def.includes("background-image")) {
        count = count + 1
        // returns url:(http://linktoimage.com) so we need to strip the string of 'url(
' and ')'
        let backgroundImgURL = localHtml(div).css('background-image');
        backgroundImgURL = backgroundImgURL.replace("url(", "");
        backgroundImgURL = backgroundImgURL.slice(0, -1);
        backgroundImgURL = backgroundImgURL.replace(/"/g, "")

        backgroundImgURL = checkURL(backgroundImgURL);
        if (backgroundImgURL == null) {
          //Show a placeholder
        }

        let extension = get_url_extension(backgroundImgURL);
        let file_name = dir_folder + "/" + index + "_bg." + extension;

        downloadAssets(backgroundImgURL, file_name);

        let local_html_name = "url('images/" + index + "_bg." + extension + "')";
        localHtml(div).css('background-image', local_html_name)
      }
    }
  });
}
```

The function itself is very similar to the previously discussed *handeImages* function, with the small modification that instead of the src attribute the code has to look for CSS ‘background-image’ rule and then parse it (because the cheerio module returns the whole rule) and extract the image source.

The only issue with this solution, is that it does not check external style sheet file properties, meaning that, if the background image is not defined inline, then the application will not be able to access it.

System Testing and Evaluation

Because it is difficult for a computer to verify whether or not a web page has been downloaded successfully as the specification require. There were very few programmatic tests that could be perform on the system to check its performance. However, to test and verify that the system is actually capable of doing on what its intended to do and whether or not the project has been successful, I performed test cases based on some of the projects functional requirements. Additionally, I chose a few popular and content diverse websites and checked whether or not the application was able to successfully download and display them.

5.1 Test Cases

Below are a few test cases that were used to analyse whether or not the system has met its functional requirements.

Test Case ID: 1		● **Test Purpose:** Given an URL as an input, the system must download the necessary html file and supporting asset files.	
Preconditions: The system is connected to an active internet connection and the provided link is valid			
Step No.	**Procedure:**	**Response:**	Pass/Fail
1	The "add" button was clicked	The prompt box will appear.	Pass
2	A valid URL is submitted to the prompt box	Prompt box disappears, and after a while an entry is created representing the newly added URL.	Pass
3	The browser icon is clicked	A browser window is opened and the downloaded website is loaded.	Pass

Test Case ID: 2		● **Test Purpose:** Users must be able to delete the downloaded websites	
Preconditions: A website has already been downloaded and added to the list.			
Step No.	**Procedure:**	**Response:**	Pass/Fail
1	User click on the website from the list	The selected website is highlighted on the list and appropriate data is shown on the right pane of the application	Pass
2	User clicks the delete button	The website is deleted from the list and all of the local files associated with that instance have been removed	Pass
3	Entry is removed from the list	The next available item is selected on the list and displayed on the right pane of the application	Fail

Test Case ID: 3	● **Test Purpose:** Application is able to crawl the first level of associated links from the source page.
Preconditions: A website has already been downloaded and added to the list with the web crawling option enabled.	

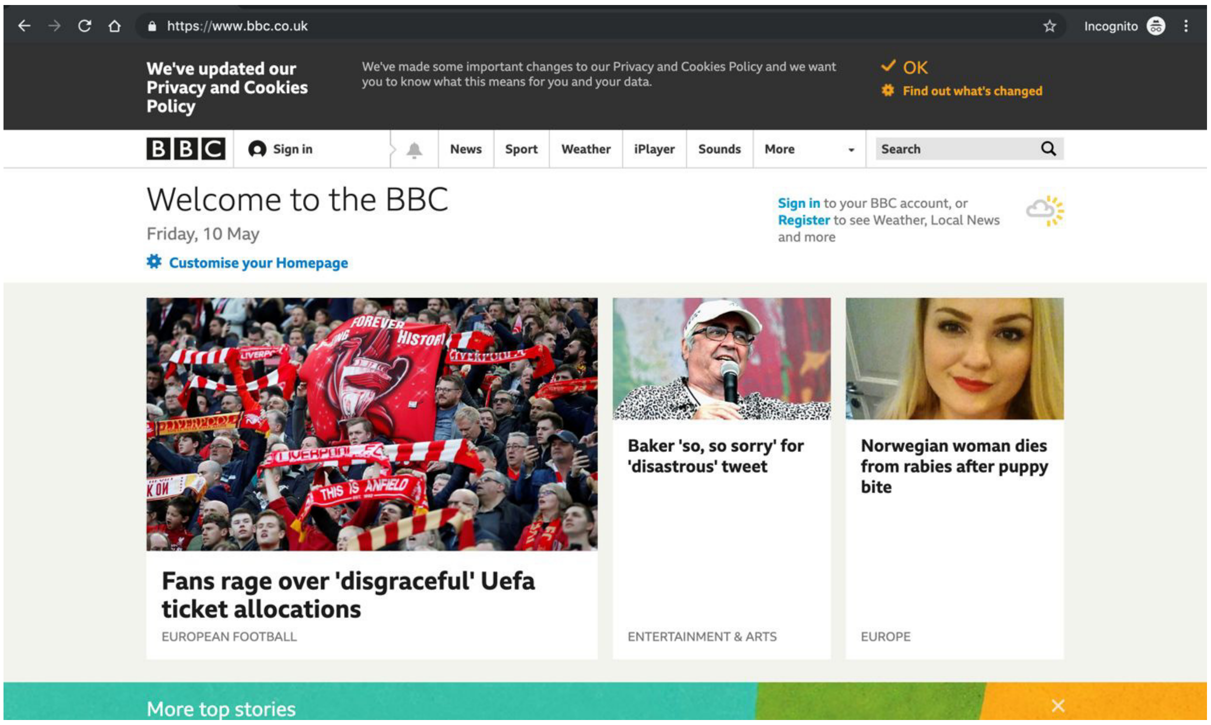
Step No.	**Procedure:**	**Response:**	Pass/Fail
1	User opens the downloaded website from the list	A browser window is opened and the downloaded website is loaded.	Pass
2	User clicks on any link that has the same domain as the source URL	The website navigates to the clicked URL, displaying all the required data, styling and images	Pass
3	User goes back and chooses a different link	The website navigates to the clicked URL, displaying all the required data, styling and images	Pass

5.2 Manual website test results

The scope of this project mainly focused on properly downloading bb.co.uk with the reasoning that if the application is able to handle such a complex website then it should be able to handle most of the available websites with relative accuracy. However, for the sake of evaluating the success of this project, it was decided to test popular websites with different designs and see how well the application is able to handle these requests and how well an already existing solution (in this case Google Chrome "Save-As" feature) would be able to handle the same request. To avoid personalised content from the live version, some websites which use user data and cookies have been run in an incognito mode.

- ***bbc.co.uk***

Testing justification: The benchmark website that this project was tested against, which provides a diverse and rich content.


https://www.bbc.co.uk

We've updated our Privacy and Cookies Policy

We've made some important changes to our Privacy and Cookies Policy and we want you to know what this means for you and your data.

Welcome to the BBC

Friday, 10 May

Customise your Homepage

Sign in to your BBC account, or Register to see Weather, Local News and more

Fans rage over 'disgraceful' Uefa ticket allocations

EUROPEAN FOOTBALL

Baker 'so, so sorry' for 'disastrous' tweet

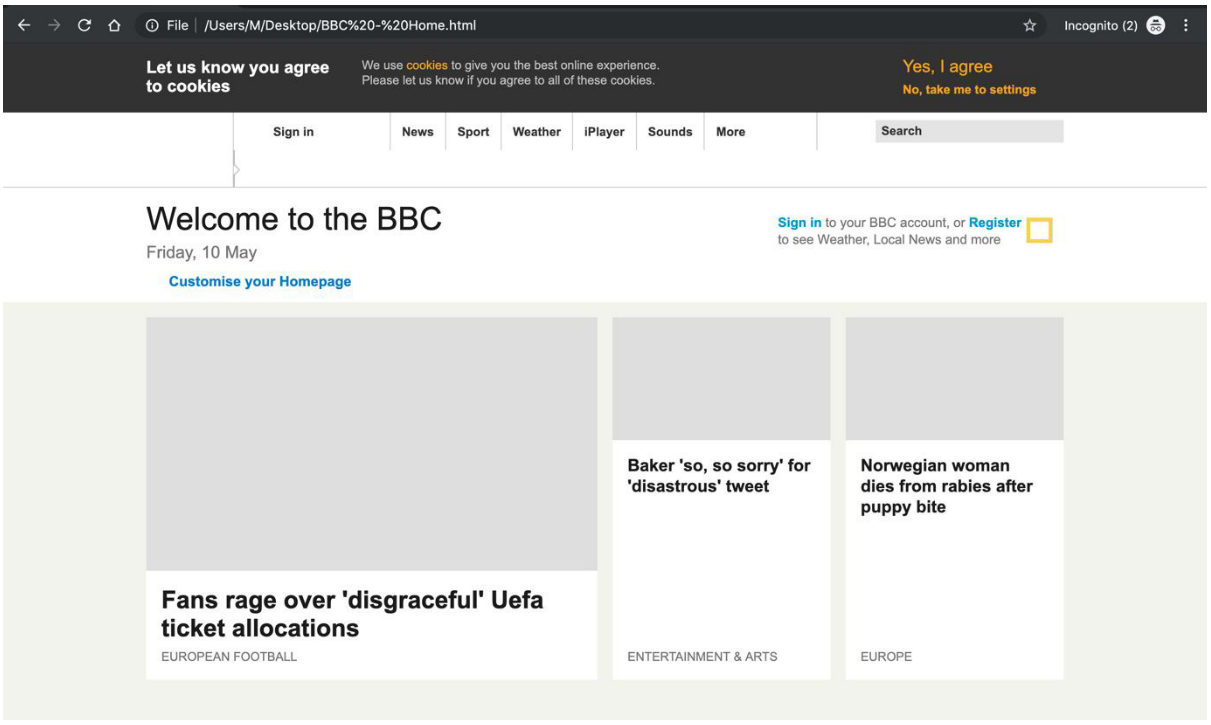
ENTERTAINMENT & ARTS

Norwegian woman dies from rabies after puppy bite

EUROPE

More top stories

Figure 12: [Screenshot] live version of https://bbc.co.uk


File | /Users/M/Desktop/BBC%20-%20Home.html

Let us know you agree to cookies

We use cookies to give you the best online experience. Please let us know if you agree to all of these cookies.

Welcome to the BBC

Friday, 10 May

Customise your Homepage

Sign in to your BBC account, or Register to see Weather, Local News and more

Fans rage over 'disgraceful' Uefa ticket allocations

EUROPEAN FOOTBALL

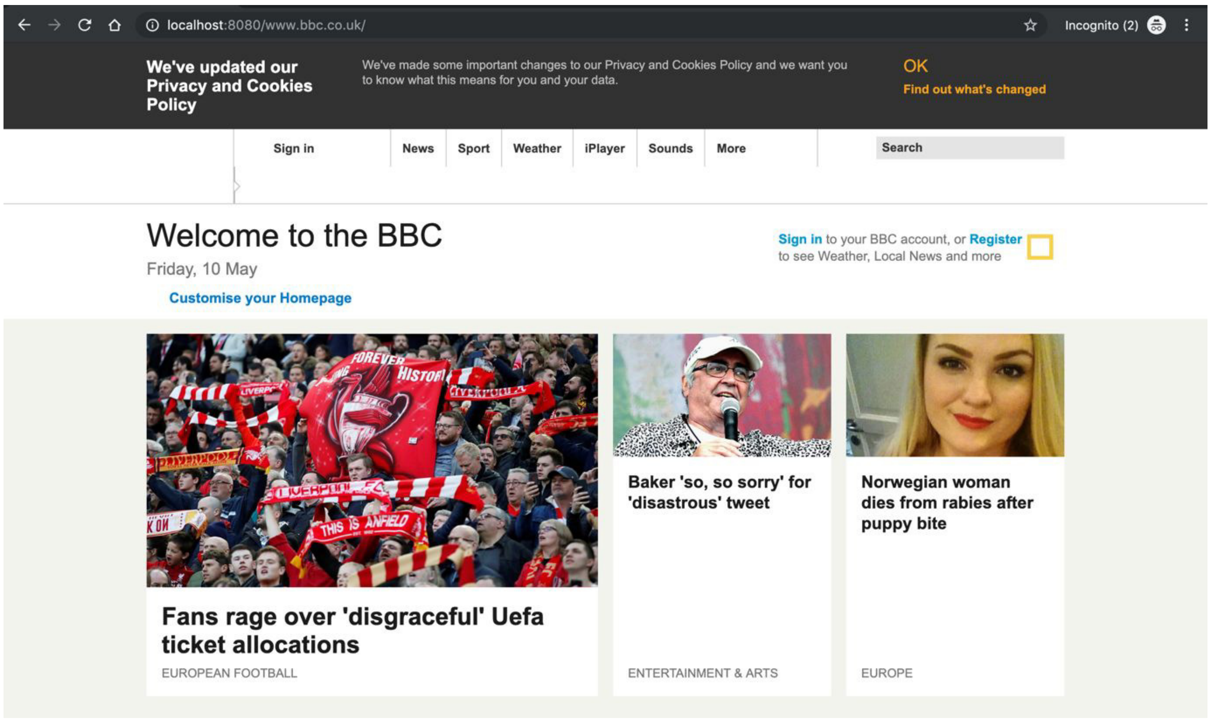
Baker 'so, so sorry' for 'disastrous' tweet

ENTERTAINMENT & ARTS

Norwegian woman dies from rabies after puppy bite

EUROPE

Figure 13: [Screenshot] Google Chrome “Save-As” representation of https://bbc.co.uk


localhost:8080/www.bbc.co.uk/

Incognito (2)

We've updated our Privacy and Cookies Policy

We've made some important changes to our Privacy and Cookies Policy and we want you to know what this means for you and your data.

OK

Find out what's changed

Sign in | News | Sport | Weather | iPlayer | Sounds | More | Search

Welcome to the BBC

Friday, 10 May

Customise your Homepage

Sign in to your BBC account, or Register to see Weather, Local News and more

Fans rage over 'disgraceful' Uefa ticket allocations

EUROPEAN FOOTBALL

Baker 'so, so sorry' for 'disastrous' tweet

ENTERTAINMENT & ARTS

Norwegian woman dies from rabies after puppy bite

EUROPE

Figure 14: [Screenshot] locally hosted version of https://bbc.co.uk

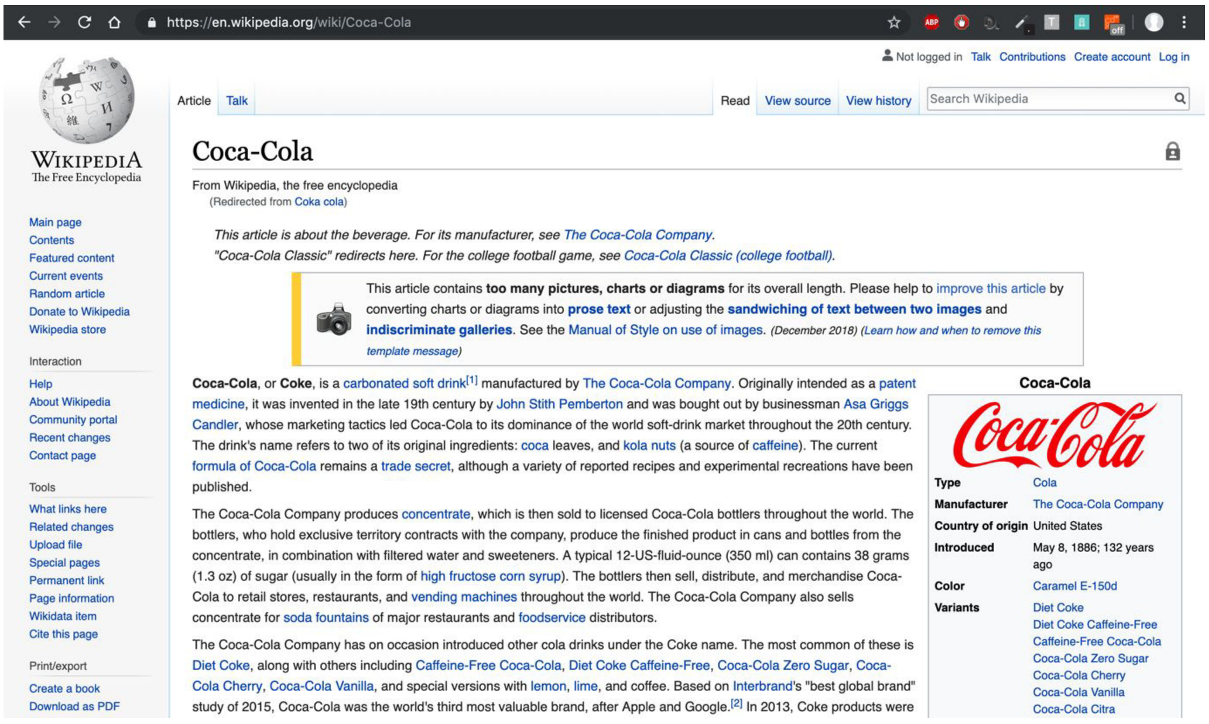
Observed Results:

Apart from the privacy cookie prompt at the top of the website, for which it is not clear why does the text differs from the live version. The local version is very similar to the live version with the exception of some icons and styling issues for the tab navigation. The icons aren't available because their source is referenced in an external style sheet document and not the index.html file itself.

The "Save-as" feature failed to get the article images, because as previously discussed BBC places their as a background image for a div container.

- ***Wikipedia***

Testing justification: Wikipedia is one of the most popular websites with a unique and rich content.


https://en.wikipedia.org/wiki/Coca-Cola

Not logged in Talk Contributions Create account Log in

Article Talk Read View source View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF

Coca-Cola

From Wikipedia, the free encyclopedia
(Redirected from Coka cola)

This article is about the beverage. For its manufacturer, see The Coca-Cola Company.
"Coca-Cola Classic" redirects here. For the college football game, see Coca-Cola Classic (college football).

This article contains **too many pictures, charts or diagrams** for its overall length. Please help to improve this article by converting charts or diagrams into **prose text** or adjusting the **sandwiching of text between two images** and **indiscriminate galleries**. See the Manual of Style on use of images. *(December 2018) (Learn how and when to remove this template message)*

Coca-Cola, or **Coke**, is a carbonated soft drink[1] manufactured by The Coca-Cola Company. Originally intended as a patent medicine, it was invented in the late 19th century by John Stith Pemberton and was bought out by businessman Asa Griggs Candler, whose marketing tactics led Coca-Cola to its dominance of the world soft-drink market throughout the 20th century. The drink's name refers to two of its original ingredients: coca leaves, and kola nuts (a source of caffeine). The current formula of Coca-Cola remains a trade secret, although a variety of reported recipes and experimental recreations have been published.

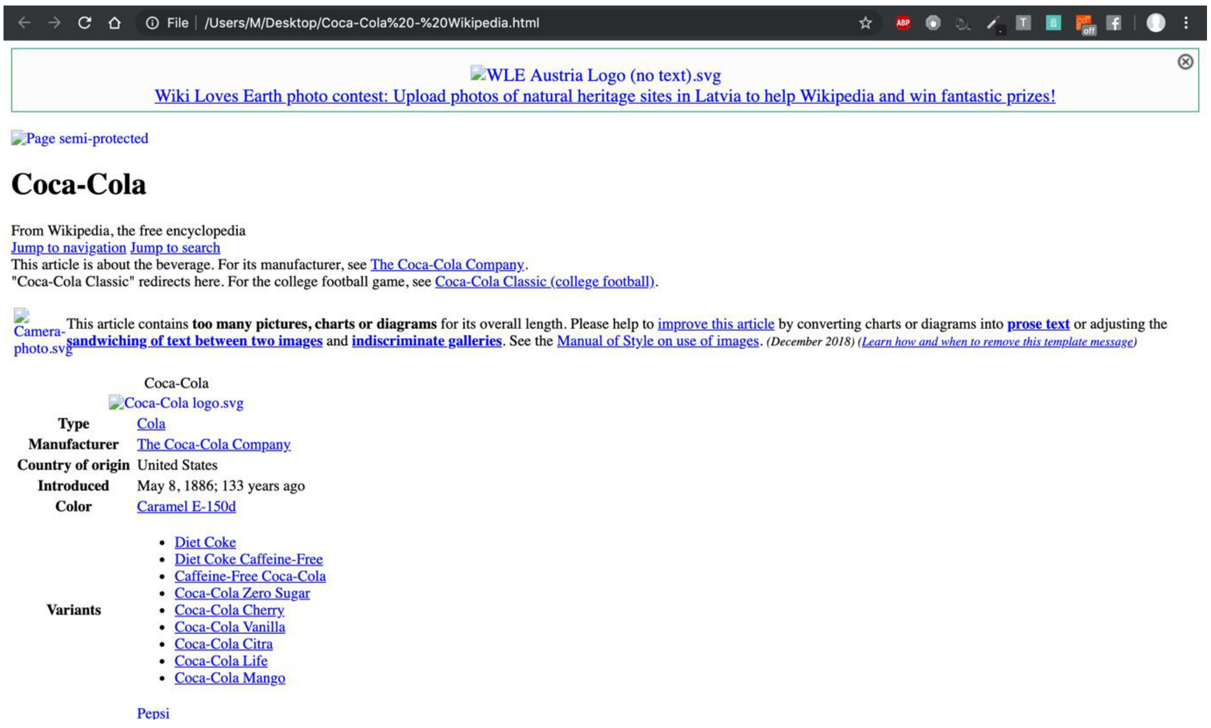
The Coca-Cola Company produces concentrate, which is then sold to licensed Coca-Cola bottlers throughout the world. The bottlers, who hold exclusive territory contracts with the company, produce the finished product in cans and bottles from the concentrate, in combination with filtered water and sweeteners. A typical 12-US-fluid-ounce (350 ml) can contains 38 grams (1.3 oz) of sugar (usually in the form of high fructose corn syrup). The bottlers then sell, distribute, and merchandise Coca-Cola to retail stores, restaurants, and vending machines throughout the world. The Coca-Cola Company also sells concentrate for soda fountains of major restaurants and foodservice distributors.

The Coca-Cola Company has on occasion introduced other cola drinks under the Coke name. The most common of these is Diet Coke, along with others including Caffeine-Free Coca-Cola, Diet Coke Caffeine-Free, Coca-Cola Zero Sugar, Coca-Cola Cherry, Coca-Cola Vanilla, and special versions with lemon, lime, and coffee. Based on Interbrand's "best global brand" study of 2015, Coca-Cola was the world's third most valuable brand, after Apple and Google.[2] In 2013, Coke products were

Coca-Cola

Type	Cola
Manufacturer	The Coca-Cola Company
Country of origin	United States
Introduced	May 8, 1886; 132 years ago
Color	Caramel E-150d
Variants	Diet Coke Diet Coke Caffeine-Free Caffeine-Free Coca-Cola Coca-Cola Zero Sugar Coca-Cola Cherry Coca-Cola Vanilla Coca-Cola Citra

Figure 15: [Screenshot] Live version of https://en.wikipedia.org/wiki/Coca-Cola


File | /Users/M/Desktop/Coca-Cola%20-%20Wikipedia.html

WLE Austria Logo (no text).svg
Wiki Loves Earth photo contest: Upload photos of natural heritage sites in Latvia to help Wikipedia and win fantastic prizes!

Page semi-protected

Coca-Cola

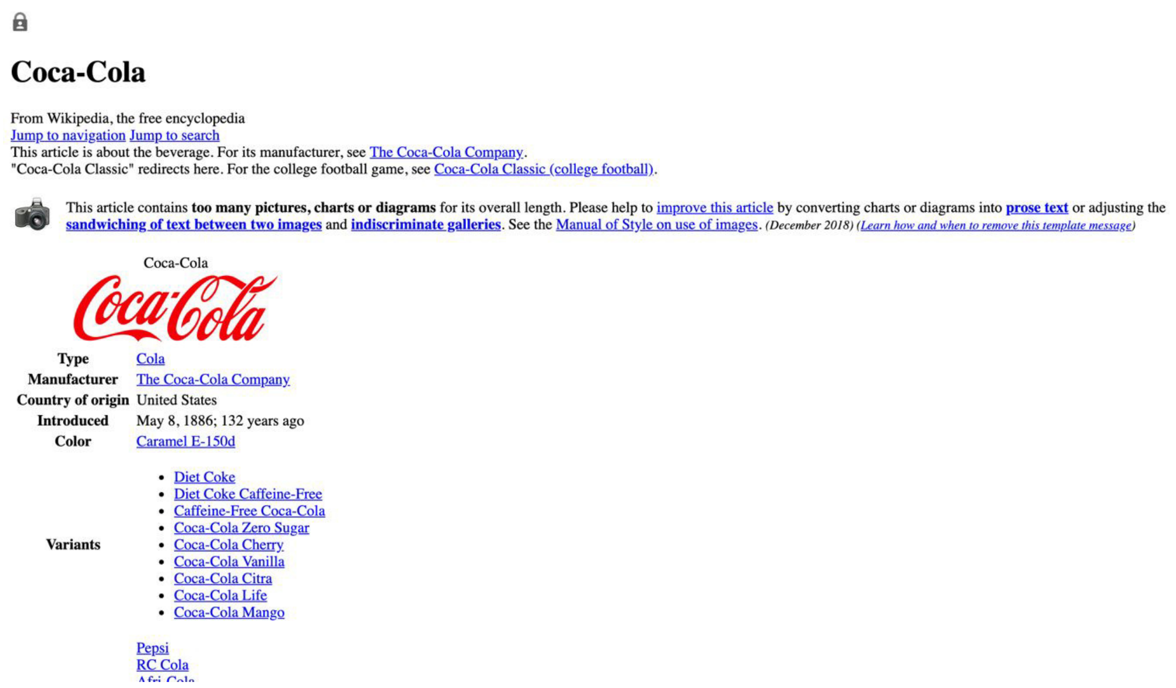
From Wikipedia, the free encyclopedia
Jump to navigation Jump to search
This article is about the beverage. For its manufacturer, see The Coca-Cola Company.
"Coca-Cola Classic" redirects here. For the college football game, see Coca-Cola Classic (college football).

Camera-photo.svg This article contains **too many pictures, charts or diagrams** for its overall length. Please help to improve this article by converting charts or diagrams into **prose text** or adjusting the **sandwiching of text between two images** and **indiscriminate galleries**. See the Manual of Style on use of images. *(December 2018) (Learn how and when to remove this template message)*

Coca-Cola
Coca-Cola logo.svg

Type	Cola
Manufacturer	The Coca-Cola Company
Country of origin	United States
Introduced	May 8, 1886; 133 years ago
Color	Caramel E-150d
Variants	• Diet Coke • Diet Coke Caffeine-Free • Caffeine-Free Coca-Cola • Coca-Cola Zero Sugar • Coca-Cola Cherry • Coca-Cola Vanilla • Coca-Cola Citra • Coca-Cola Life • Coca-Cola Mango
	Pepsi

Figure 16: [Screenshot] Google Chrome "Save-As" representation of https://en.wikipedia.org/wiki/Coca-Cola



Coca-Cola

From Wikipedia, the free encyclopedia
Jump to navigation Jump to search
This article is about the beverage. For its manufacturer, see The Coca-Cola Company.
"Coca-Cola Classic" redirects here. For the college football game, see Coca-Cola Classic (college football).

This article contains **too many pictures, charts or diagrams** for its overall length. Please help to improve this article by converting charts or diagrams into **prose text** or adjusting the **sandwiching of text between two images** and **indiscriminate galleries**. See the Manual of Style on use of images. *(December 2018) (Learn how and when to remove this template message)*

Coca-Cola

Type	Cola
Manufacturer	The Coca-Cola Company
Country of origin	United States
Introduced	May 8, 1886; 132 years ago
Color	Caramel E-150d
Variants	• Diet Coke • Diet Coke Caffeine-Free • Caffeine-Free Coca-Cola • Coca-Cola Zero Sugar • Coca-Cola Cherry • Coca-Cola Vanilla • Coca-Cola Citra • Coca-Cola Life • Coca-Cola Mango
	Pepsi RC Cola

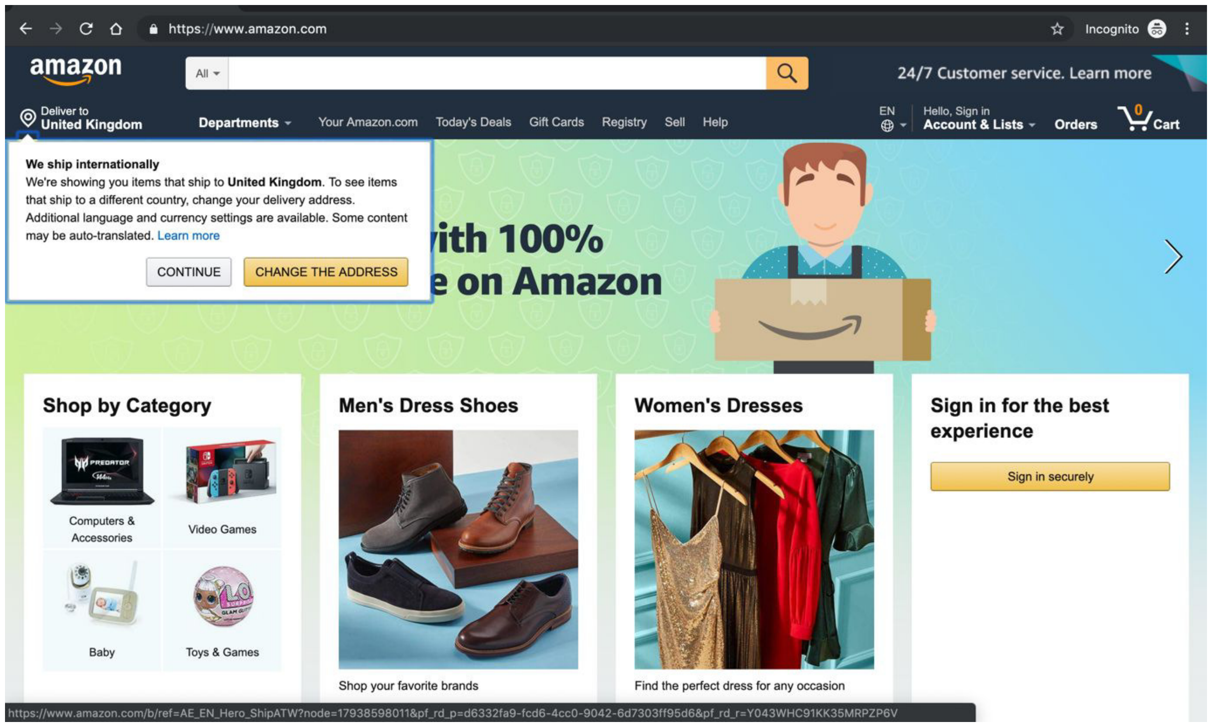
Figure 17: [Screenshot] Local version of https://en.wikipedia.org/wiki/Coca-Cola

Observed Results:

It is quite clear that in this case, that even though all the images are available, the application has failed to properly display the downloaded webpage as it should be displayed. It managed to get all the images, but failed to properly style the content.

- ***Amazon***

Justification: It is the biggest ecommerce website, and its content rich page, provides an excellent test case



We ship internationally
We're showing you items that ship to **United Kingdom**. To see items that ship to a different country, change your delivery address. Additional language and currency settings are available. Some content may be auto-translated. Learn more

https://www.amazon.com/b/ref=AE_EN_Hero_ShipATW?node=17938598011&pf_rd_p=d6332fa9-fcd6-4cc0-9042-6d7303ff95d6&pf_rd_r=Y043WHC91KK35MRPZP6V

Figure 18: [Screenshot] live version of https://www.amazon.com/

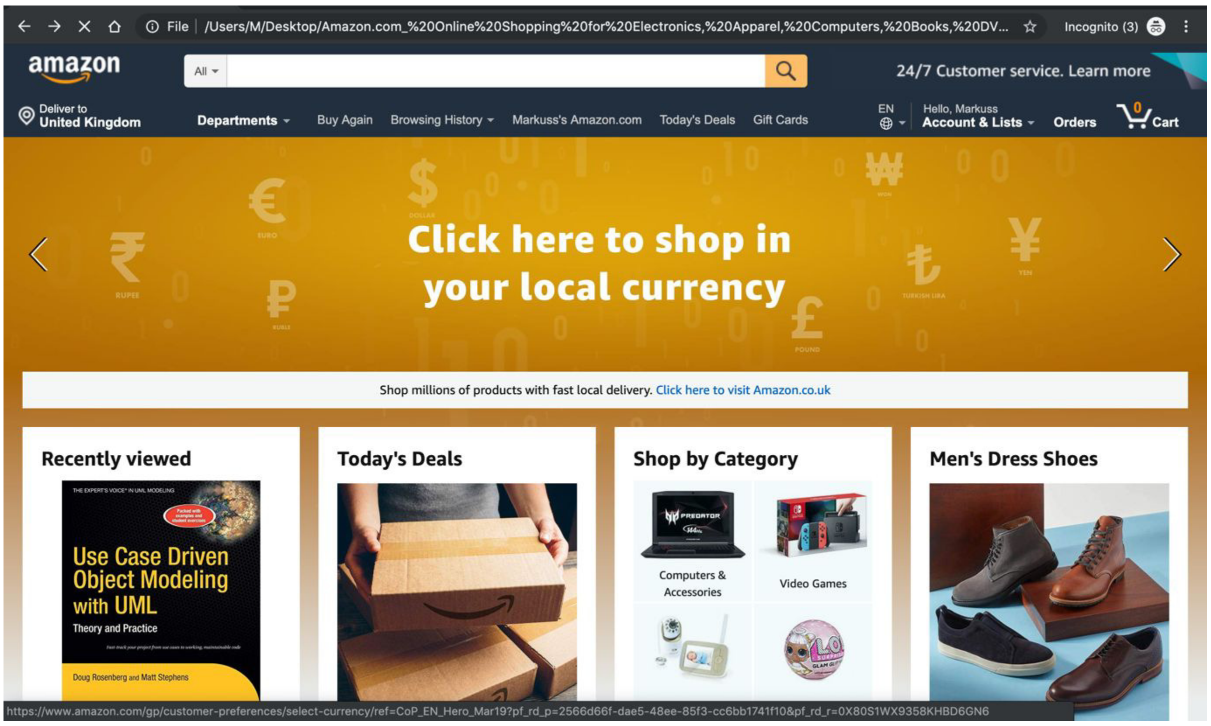

File | /Users/M/Desktop/Amazon.com_%20Online%20Shopping%20for%20Electronics,%20Apparel,%20Computers,%20Books,%20DV...
Incognito (3)
amazon
24/7 Customer service. Learn more
Deliver to United Kingdom
Departments
Buy Again
Browsing History
Markuss's Amazon.com
Today's Deals
Gift Cards
Hello, Markuss Account & Lists
Orders
Cart
Click here to shop in your local currency
Shop millions of products with fast local delivery. Click here to visit Amazon.co.uk
Recently viewed
Today's Deals
Shop by Category
Men's Dress Shoes
Computers & Accessories
Video Games
https://www.amazon.com/gp/customer-preferences/select-currency/ref=CoP_EN_Hero_Mar19?pf_rd_p=2566d66f-dae5-48ee-85f3-cc6bb1741f10&pf_rd_r=0X80S1WX9358KHBD6GN6

Figure 19: [Screenshot] Google Chrome “Save-As” representation of https://www.amazon.com/

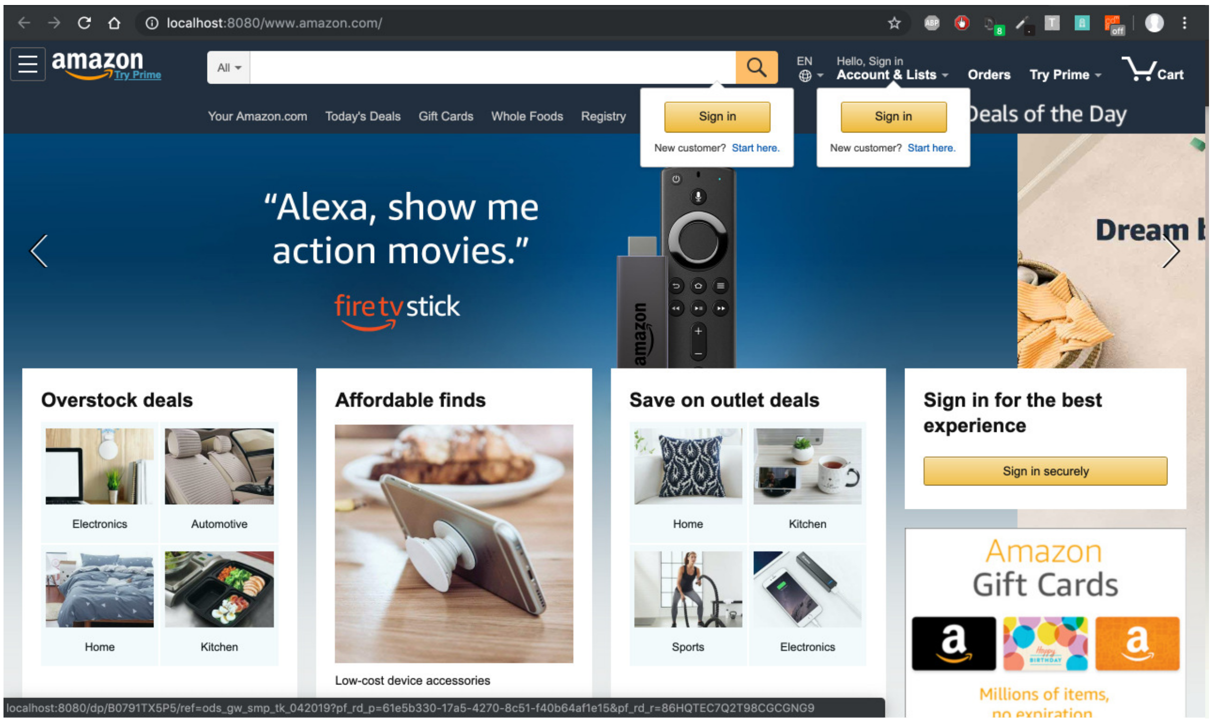

localhost:8080/www.amazon.com/
amazon Try Prime
Hello, Sign in Account & Lists
Orders
Try Prime
Cart
Your Amazon.com
Today's Deals
Gift Cards
Whole Foods
Registry
Sign in
New customer? Start here.
Deals of the Day
“Alexa, show me action movies.”
fire tv stick
Overstock deals
Electronics
Automotive
Home
Kitchen
Affordable finds
Low-cost device accessories
Save on outlet deals
Home
Kitchen
Sports
Electronics
Sign in for the best experience
Sign in securely
Amazon Gift Cards
localhost:8080/dp/B0791TX5P5/ref=ods_gw_smp_tk_042019?pf_rd_p=61e5b330-17a5-4270-8c51-f40b64af1e15&pf_rd_r=86HQTEC7Q2T98CGCGNG9

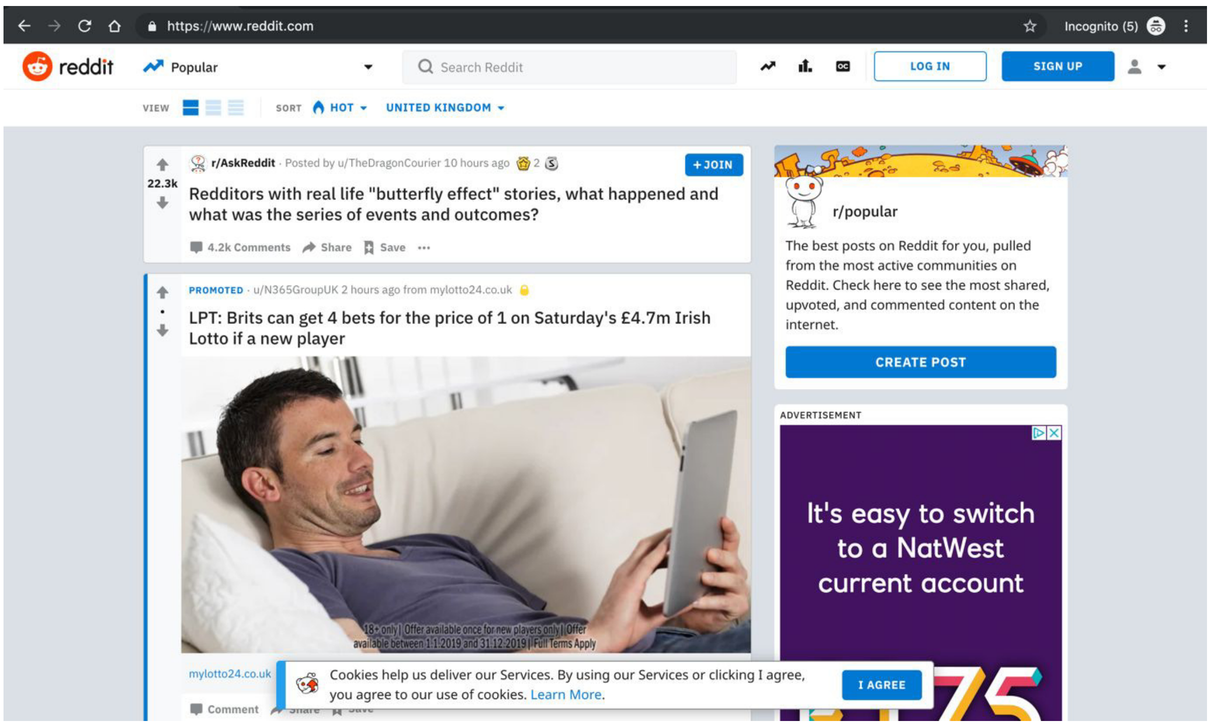
Figure 20: [Screenshot] locally hosted version of https://www.amazon.com/

Observed Results:

Apart from the fact that the local version couldn’t access the browsers location because of no internet connection. It seems that the local version is practically indistinguishable from the live version. The reason why the background image is different for the two screenshots, is because it uses an image carousel, changing the background image over time.

- ***Reddit***

Justification: “The front-page of the internet” It is a content diverse and rich page.



https://www.reddit.com

Incognito (5)

reddit Popular Search Reddit LOG IN SIGN UP

VIEW SORT HOT UNITED KINGDOM

22.3k r/AskReddit · Posted by u/TheDragonCourier 10 hours ago 2 +JOIN

Redditors with real life "butterfly effect" stories, what happened and what was the series of events and outcomes?

4.2k Comments Share Save ···

PROMOTED · u/N365GroupUK 2 hours ago from mylotto24.co.uk

LPT: Brits can get 4 bets for the price of 1 on Saturday's £4.7m Irish Lotto if a new player

18+ only | Offer available once for new players only | Offer available between 1.1.2019 and 31.12.2019 | Full Terms Apply

mylotto24.co.uk

Comment Share Save

r/popular

The best posts on Reddit for you, pulled from the most active communities on Reddit. Check here to see the most shared, upvoted, and commented content on the internet.

CREATE POST

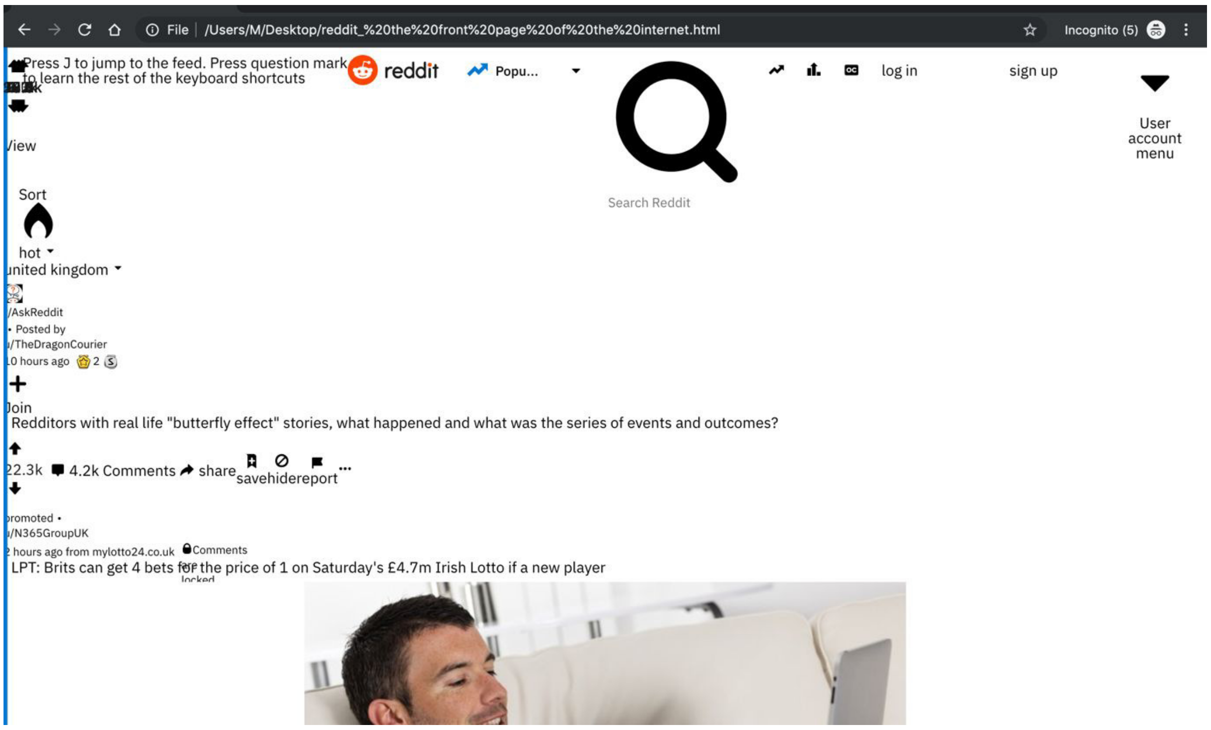
ADVERTISEMENT

It's easy to switch to a NatWest current account

Cookies help us deliver our Services. By using our Services or clicking I agree, you agree to our use of cookies. Learn More.

I AGREE

Figure 21: [Screenshot] live version of https://www.reddit.com/



File | /Users/M/Desktop/reddit_%20the%20front%20page%20of%20the%20internet.html

Incognito (5)

Press J to jump to the feed. Press question mark to learn the rest of the keyboard shortcuts

reddit Popu... log in sign up

User account menu

Search Reddit

View

Sort

hot

united kingdom

r/AskReddit
• Posted by
u/TheDragonCourier
10 hours ago 2

Join

Redditors with real life "butterfly effect" stories, what happened and what was the series of events and outcomes?

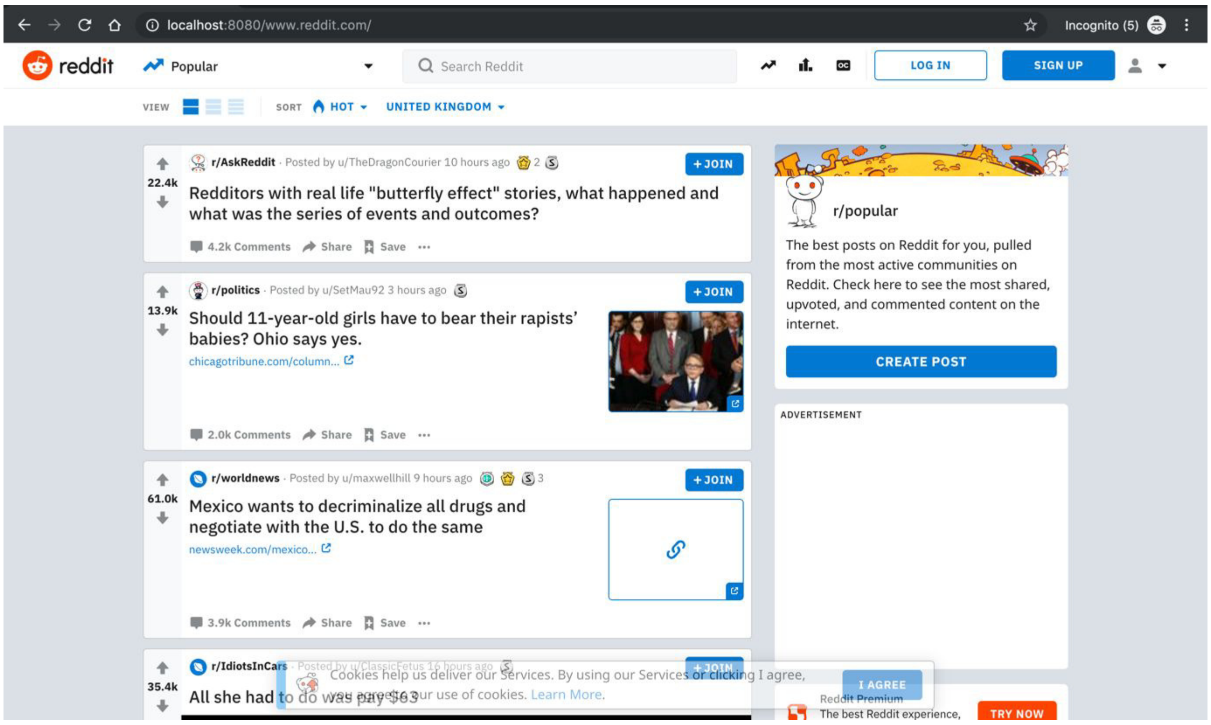
22.3k 4.2k Comments share savehidereport ...

promoted •
u/N365GroupUK
2 hours ago from mylotto24.co.uk Comments

LPT: Brits can get 4 bets for the price of 1 on Saturday's £4.7m Irish Lotto if a new player

locked

Figure 22: [Screenshot] Google Chrome “Save-As” representation of https://www.reddit.com/



22.4k r/AskReddit · Posted by u/TheDragonCourier 10 hours ago 2

Redditors with real life "butterfly effect" stories, what happened and what was the series of events and outcomes?

4.2k Comments

13.9k r/politics · Posted by u/SetMau92 3 hours ago

Should 11-year-old girls have to bear their rapists' babies? Ohio says yes.

chicagotribune.com/column...

2.0k Comments

61.0k r/worldnews · Posted by u/maxwellhill 9 hours ago 3

Mexico wants to decriminalize all drugs and negotiate with the U.S. to do the same

newsweek.com/mexico...

3.9k Comments

35.4k r/IdiotsInCars · Posted by u/ClassicFetus 16 hours ago

All she had to do was pay $63

r/popular

The best posts on Reddit for you, pulled from the most active communities on Reddit. Check here to see the most shared, upvoted, and commented content on the internet.

Cookies help us deliver our Services. By using our Services or clicking I agree, you agree to our use of cookies. Learn More.

Reddit Premium

The best Reddit experience,

Figure 23: locally hosted version of https://www.reddit.com/

Observed Results:

As with amazon, it looks like the local version is identical to the live version, with the exception that it failed to capture the advertisement image on the right. On the other hand, Google Chrome "Save-As" feature has completely failed to capture the style of the page.

5.3 Meeting requirements

Referencing the previously laid out functional requirements, the developed application met all of the Must have requirements. The Should have requirement "*The displayed local website should have the same styling and arrangement as the online version"* has been partially met. As observed in the previous section, the application struggled to faithfully replicate some of the tested webpages.

Additionally, the application has also met some of the defined could have requirements, such as the ability to crawl the next level of associated links to the source URL. Its partially able to handle embedded video files (restricted strictly to iframe), but it has failed to meet the rest of the could have requirements, such as:

- *The system could have a built in internet connectivity listener, that updates the already downloaded websites once the user connects to the internet.*
- *The system could allow to query saved websites*

Since at the beginning of the project it defined bbc.co.uk as its main test benchmark, indicating that if it can successfully download and display it, then the application should be able to handle most of the other websites found on the internet as they would fall under the complexity of bbc.co.uk. That turned out not to be the case. Even though the project was able to download bbc.co.uk almost flawlessly, with the exception of some obscurely defined missing icons. It did struggle with other complex websites such as Wikipedia. Those cases notwithstanding, the application was able to successfully copy Amazon, Reddit and other tested websites. Not to mention it out performed the existing “Save-As” feature of Google Chrome in every one of the test cases.

It is just a matter of fine tuning the existing solution and adding additional edge cases so that the application could also handle the websites it failed to capture in testing. Overall there is a solid base that if improved upon could turn into a go to solution for website offline viewing.

Future Work

It is quite apparent that even though the minimal viable product was reached in the course of this project, there are various things that the project could improve upon given a longer development time.

This includes such improvements as:

- Broader asset availability – The biggest issue with these kinds of projects. The application allows to save video files if the media is embedded in an <iframe> tag. If the video is embedded in any kind of different way, the application will not be able to display that video locally. So as a future possible solution the first priority would be to allow not only for a variety of embedded video formats but also for a wider range of different embedded media formats.
- Improved UI - UX guidelines suggest to always display the systems status to the users. A possible future continuance would include adding a progress bar on the download status. Currently the system doesn’t report to the user in any way, when the webpage has finished downloading, especially in the case when web crawling is used.
- Social Networks – Adding support for Facebook, Instagram, etc. would greatly improve upon the applications use cases.
- Webpage Updates – In the design and specification stage, the report talked about keeping web pages up to date when connected to an internet connection or providing a refresh button in the UI, which would then update the websites local files. Due to time constraints, these features were not implemented in the application, and if future work were to take place, it would be recommended to add said functionality to the existing solution.
- Paginated content – Currently the application has no way of loading paginated content (vertical or sectioned) a necessary future improvement would be the ability to scroll through a paginated feed.

Conclusions

In conclusion, the project has achieved its pre-defined goal, of creating an offline website viewing solution that improves upon the already available similar products.
The application is capable of properly downloading and displaying most of the websites on the internet. Some more successfully than others. Compared to the already existing solutions, the application offers a way to organise the downloaded web page content, circumvents issues such as lazy-loaded content and CORS restrictions. The program is able

to handle diverse embedded media types although with varying degrees of success. It also enables the user to view and access the associated links of the website through web crawling.

Overall, even if the project still has some issues with properly displaying some webpages or accessing their embedded content - and its current representation of tested webpages isn't perfect, it succeeded in what it set out to do. And given enough time to complete it, it is on the right track on providing the best offline viewing experience for static web pages.

Reflection

At the beginning of the project, I was worried that the task at hand might be too simplistic. Not fully realising the extent of the actual work needed to successfully implement this project.

The project not only made me realise that all of the meticulous planning "goes out of the window" as soon as you encounter your first major problem - which is inevitable. You realise that the scope of the project has to be changed so that the original image of the finished product could cater to reality.

Technical skills such as a better understanding of the Electron framework, increased proficiency in JavaScript and Node.js notwithstanding.

I feel like my project modelling prowess has increased exponentially over the course of the project. Like most students undertaking similar work. Before starting the project my exposure to designing a system before development was at most choosing the target platform and outlying the general needed requirements.

However, I now realise the importance of actually laying out the general idea of how the project should work in a flow-chart or a sequence diagram, instead of immediately getting down to the implementation part of the project and designing the system "as you go along".

Additionally, through trying to download the passed websites. I developed a mind-set of looking at a finished product and trying to deconstruct it and understand the technology behind it. Gaining a better understanding of various technologies and structures that hold most of the websites we visit daily. The project really made me value the holistic process of any development progress.

References

1. Nick Raboy, 2015, *Use SQLite In Ionic 2 Instead Of Local Storage* , Nick Raboys weblog, viewed 20 February 2019 <https://www.thepolyglotdeveloper.com/2015/12/use-sqlite-in-ionic-2-instead-of-local-storage/>
2. TutorialPoints, *Node.js – Introduction* , viewed 10 March 2019 <https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm>
3. Electron Documentation, viewed 29 February 2019 <https://electronjs.org/docs/tutorial/about>

4. Felix Rieseberg, *Introducing Electron,* Oreilly, viewed 29 February 2019 <https://www.oreilly.com/library/view/introducing-electron/9781491996041/ch01.html>
5. *GitHub Documentation,* viewed 3 March 2019 <https://github.com/cheeriojs/cheerio>
6. *GitHub Documentation,* viewed 3 March 2019 <https://github.com/axios/axios>
7. NodeJS API Documentation, viewed 29 February 2019<https://nodejs.org/api/url.html#url_url>
8. GitHUb Documentation, viewed 25 March 2019,<https://github.com/senchalabs/connect#readme>
9. NPM Documentation, Serve-Static, viewed 27 April 2019 <https://www.npmjs.com/package/serve-static>
10. NodeJS API Documentation, File System, viewed 29 February 2019 <https://nodejs.org/api/fs.html>
11. NPM Documentation, Electron-Prompt, viewed 27 April 2019 <https://www.npmjs.com/package/electron-prompt>
12. NPM Documentation, Rimraf, viewed 15 April 2019 <https://www.npmjs.com/package/rimraf>
13. Harshadkumar B Prajapati, Reasearch Gate, viewed 2 May 2019, <https://www.researchgate.net/figure/MVC-Design-Pattern_fig1_224398744>
14. W3SCHOOLS, *HTML DOM scrollHeight Property*, viewed 7 April 2019 <https://www.w3schools.com/jsref/prop_element_scrollheight.asp>
15. Bartosz Szczeciński, *Understanding CORS,* Medium blog post, viewed 11 April 2019 <https://medium.com/@baphemot/understanding-cors-18ad6b478e2b>
16. Nithya Sambasivan, *Connectivity, Culture, and Credit,* viewed 13 April 2019 <https://design.google/library/connectivity-culture-and-credit/>