Timeless Citation

Group 5
Martin P. Tielemans, Emil Gade, Kasper Vestergaard Jensen, Li Thao Phung,
Eto Kasia Kibanga and Jacob Kwasi Autzen

AALBORG UNIVERSITY
STUDENT REPORT

Computer Science
Aalborg University
http://cs.aau.dk

Title:
Timeless Citation

Theme:
Networking

Project Period:
Spring Semester 2024

Project Group:
Group 5

Participants:
Martin P. Tielemans
Emil Gade
Kasper Vestergaard Jensen
Li Thao Phung
Eto Kasia Kibanga
Jacob Kwasi Autzen

Supervisor:
Abiram Mohanaraj

Date:
29. May 2024

Copies: 1

Page Numbers: 62

Abstract:

The digitization of content has revealed a vulnerability in the preservation of digital content, known as link-rot, where links stop working. This results in disrupting the flow of accessing digital content and undermines the credibility of websites. This report provides a decentralized solution to mitigate link-rot.

For this project, a Chrome extension was developed utilizing suitable technologies such as React, Valtio, Plasmo and Typescript. These technologies enable the Chrome extension to provide users with the ability to upload the content of a webpage directly onto the IPFS network, ensuring the preservation of the information without relying on centralized systems.

The design section describes the flow of the different interfaces in the Chrome extension as well as the core functionalities such as the ability to download and upload pages. Thereafter the implementation highlights the flow of the functions as well as the key recursive algorithm that traverses the DOM to download external sources. Moreover, manual- and unit tests are done to ensure the solution works as intended.

Finally, the limitations and further improvements of the application are discussed, followed by a conclusion.

Contents

Preface

We would like to express our gratitude to Abiram Mohanaraj, the supervisor of our group, for his constructive criticism and direction, which led to a good cooperation.

Martin P. Tielemans
mtiele23@student.aau.dk

Emil Gade
egade23@student.aau.dk

Kasper Vestergaard Jensen
kj23@student.aau.dk

Li Thao Phung
lphung23@student.aau.dk

Eto Kasia Kibanga
ekiban23@student.aau.dk

Jacob Kwasi Autzen
jautze23@student.aau.dk

This project is developed with the help of multiple tools. This includes Typst as a collaborative Latex editor, as well as Git & Github for version controlling the solution. Visual Studio Code is used as the code editor of choice and Exalcidraw/Figma has been used for graphical interpretations. Additionally, generative AI has been utilized for brainstorming and grammatic feedback

1 Introduction

The transition to the digital age has fundamentally reshaped how knowledge is preserved, moving away from traditional paper-based methods toward digital formats. This shift has significantly enhanced the accessibility of information, facilitating it across the internet. However, this reliance on digital platforms for storing information has exposed a critical vulnerability known as link-rot [1].

Link-rot is a phenomenon where links throughout the internet stop working, effectively creating dead ends in the digital landscape. The degradation of links threatens the integrity of digitalized content and information sharing. The importance of addressing link-rot cannot be overlooked, as it directly impacts the accessibility of digital content, hindering the flow of knowledge in the digital realm [1].

Unlike the durable nature of paper-based references used in the past, digital information, and references are inherently more fragile and vulnerable. As such, safeguarding against link-rot has become an urgent priority in preserving the integrity and accessibility of digitalized knowledge. If this issue is not addressed, it could undermine the foundations of digital information exchange and jeopardizing the longevity of our collective digital heritage [1].

The report aims to provide a solution to the initial challenge of link-rot, with a focus on creating a solution that ensures the possible longevity of digital content. Through the problem analysis, an in-depth understanding of link-rot and its implications is achieved. Moreover, to develop a solution, the report examines the current solutions and the different system architectures such as centralized-, decentralized-, and distributed systems to evaluate their strengths and weaknesses. The current solutions such as Perma.cc, Wayback Machine, Archive Box, etc are all predominantly centralized. Furthermore, the main difference in the system architecture lies in the fact, that centralized solutions depend on a single entity, which poses risks, while decentralized solutions offer resilience and more control for the user. The knowledge of link-rot is used to develop a Chrome Extension, which will save content from a webpage and upload it onto the IPFS network. Thereafter, tests are done to ensure the solution works as intended. The solution will then be discussed, to look at the limitations and challenges encountered by the solution as well as improvements to implement. Finally, the report draws to a close with a conclusion by reflecting on how the solution fulfills the problem statement.

2 Problem Analysis

This problem analysis will look at the multifaceted nature of link-rot, its effects, and the current solutions available addressing the issue. To highlight the impact of link-rot multiple studies will be included, showing statistics of how link-rot impacts digital content over time. To better understand the current solutions, the internet structure and different system architectures will be explained before diving into the current solutions available. This knowledge is fundamental to understand the rest of the report, as it is a recurring topic throughout the report.

2.1 The Structure of the Internet

The internet is a system architecture that is used for communication between computer networks around the world. This huge "network of networks" has revolutionized communication and is one of the underpinnings of today's society [2]. In short, the internet does not contain information but serves as the infrastructure that enables information sharing [3], [4].

On top of this internet infrastructure lies the World Wide Web (WWW), often simply called "the web". The web is a system on top of the internet that organizes information into user-friendly websites. To communicate between machines on the web, protocols such as HTTP and HTTPS are used. Hypertext Transfer Protocol (HTTP) is the original protocol for web communication and transmits data un-encrypted, making it vulnerable to eavesdropping [5]. A more secure version of HTTP is Hypertext Transfer Protocol Secure (HTTPS), which encrypts the data sent between the servers and the browsers [6].

When a user navigates to a website, they are presented with a webpage that displays information like text, images, videos, and, importantly, hyperlinks (often called "links") [7]. A link is an element on a webpage that directs a user to another webpage or resource. They can act like bridges between webpages, allowing users to seamlessly navigate from one webpage to another. Clicking on a link instructs the computer to retrieve and display a different webpage, essentially following a new "road" within the internet network [8].

A link connects to a webpage using a combination of a domain name and a specific path within the website structure to connect a user to the correct content. A domain name such as google.com acts as a human-friendly way to navigate the web, instead of having to use complex numerical IP addresses (e.g. 192.0.2.2) [9], [10]. Humans access information through domain names, whereas web browsers use Internet Protocol (IP) addresses. The conversion from domain name to IP addresses happens through the Domain Name System (DNS) [11]. Structurally, a domain looks like it is depicted in Figure 1. It is typically read from right to left and consists of (see Figure 1):

- Top Level Domain (TLD): This tells users the general purpose of the service behind the domain name (e.g., .com, .org, .edu).
- Labels: A domain name can have multiple labels, such as "www" or "subdomain" preceding the main domain name (e.g., mail.google.com) [10].

Figure 1: Example of a domain name.

Now that the general structure of the internet structure is established, the next section will section will delve into what link-rot is, and what its implications are.

2.2 Link-rot

Link-rot or link "decay" is a phenomenon that defines the deterioration of links over time to the point of the link becoming non-functional or "dying" [1]. The phenomenon arises due to many factors and involves various aspects of web architecture, content management, and technological advancement [12].

One of the fundamental reasons for link-rot is the ever-changing nature of web content. Websites undergo various updates, redesigns, or closures, leading to changes or removal of existing content [13]. With these changes, link-rot occurs when the referenced link is not updated accordingly and still refers to the old webpage causing the link to become obsolete [13]. This process is particularly prevalent when websites undergo significant structural changes that make the originally linked content inaccessible.

Web developers usually implement HTTP status code 301 to mitigate link-rot, which means a permanent redirect [13], instead of simply deleting a webpage or moving it without proper redirection. This is done to inform users and search engines that the link has moved to a new location and helps maintain the integrity of the web. It ensures that links pointing to the old location are automatically redirected to the updated content, preventing the occurrence of link-rot over time. Returning to link-rot vulnerabilities, the connection between websites and domain names introduces another layer of vulnerability. When the assigned domain name expires, all the links associated will become non-functional and can break the link between the URL and the associated content, resulting in link-rot [13].

Link-rot can be categorized into the following categories.

- **Dropped**: Cases when the linked-to webpage no longer exists, either because it was deleted or the website shut down
- **Link removed**: When the webpage is still there but the specific link has been removed. This could be due to website updates, content changes, or a deliberate decision to remove the link.
- **Crawl error**: Occasionally, search engine crawlers face challenges while trying to access a webpage, leading to crawling errors. If these issues are not resolved, they can contribute to link-rot over time.
- **301/302**: The original link has been rerouted to another page. If the redirection is improperly configured or directs to a non-existent page, it results in link-rot.
- **Not found**: This category encompasses situations where the linked webpage cannot be located, often triggering a 404 error page. Such occurrences may arise when a webpage is relocated or deleted without implementing a proper redirection.
- **Not canonical and Noindex**: These scenarios involve deliberate actions to prevent search engine indexing of the linked page. This can be achieved either by implementing "noindex" tags or by specifying an alternative "canonical" version of the webpage [14].

Figure 2 illustrates the distribution of the most prevalent causes of link degradation. The primary cause, accounting for 47.7% of instances, is links being dropped, indicating that the URLs initially included in the articles are no longer accessible or functional. Following closely behind, at 34.2%, is the removal of links, suggesting that the content to which the URLs pointed has been deleted or relocated. Whereas the categories, such as crawl error, not found, not canonical, noindex, and broken redirect, are less common issues.

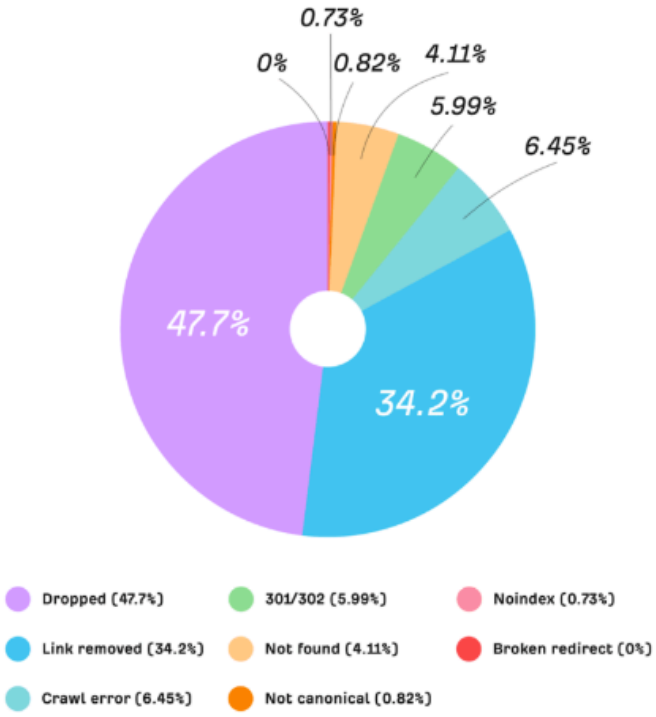

Figure 2: Percentage of the occurrence of link-rot in categories [12]

Another factor that contributes to the link-rot problem is Technological obsolescence [1]. This is when web technologies become more advanced, and web standards make older content incompatible with newer systems. This is also known as "software rot"[1], which, in short, describes the decline in the performance or functionality of a software system over time. Thus, the software becomes less efficient or reliable as it ages, even if it is not actively being modified or updated. This phenomenon is not something this report will discuss further.

Other causes of link-rot is the removal of online content due to legal or ethical concerns [13]. This can result in the link pointing at the removed content, contributing to the link-rot problem [15].

2.2.1 Unraveling Link-rot

This section will delve into various studies to uncover the presence of link-rot in the digital landscape and the effects of link-rot on information accessibility, reliability, and the integrity of online discourse.

One of the negative effects of link-rot is exemplified by a study from Harvard about link-rot in the United States Supreme Court opinions. For the majority of its history, the Supreme Court relied on citations from permanent sources like books, enabling lawyers and scholars to access and comprehend the court's evidence and reasoning. However, since 1996, there has been a notable shift, with the justices citing materials found on the internet a total of 555 times [16]. The study found that within the links published in the United States Supreme Court opinions, which is a written explanation of the court decision, over 50% of cited links no longer lead to their intended page.

This phenomenon extends to academic legal journals such as the Harvard Law Review, the Harvard Journal of Law and Technology, and the Harvard Human Rights Journal. More than 70% of the cited links from 1999 to 2012 from the legal journals suffered from link-rot [17].

Another study conducted by Ahrefs, a Software as a Service (SaaS) company, sheds light on the prevalence of link-rot in the digital realm. This study is the largest of its kind, encompassing the most recent web version. An analysis of over 14 billion links shows that approximately 1.3% of these links succumb to link-rot weekly. Furthermore, Ahrefs' extensive research delved into links pointing to 2,062,173 websites, revealing that a significant 66.5% of them had become link-rotten [12]. Moreover, the study underscores the critical role of a link's age in determining its survival. Ahrefs' findings highlight a stark contrast: while merely 5% of links from articles published in 2018 were inactive, a staggering 72% of those published in 1998 had met the same fate. Compelling data from January 2013 further supports these findings, showing that out of a sample of 174.3 million links 26.9 million had been lost [12].

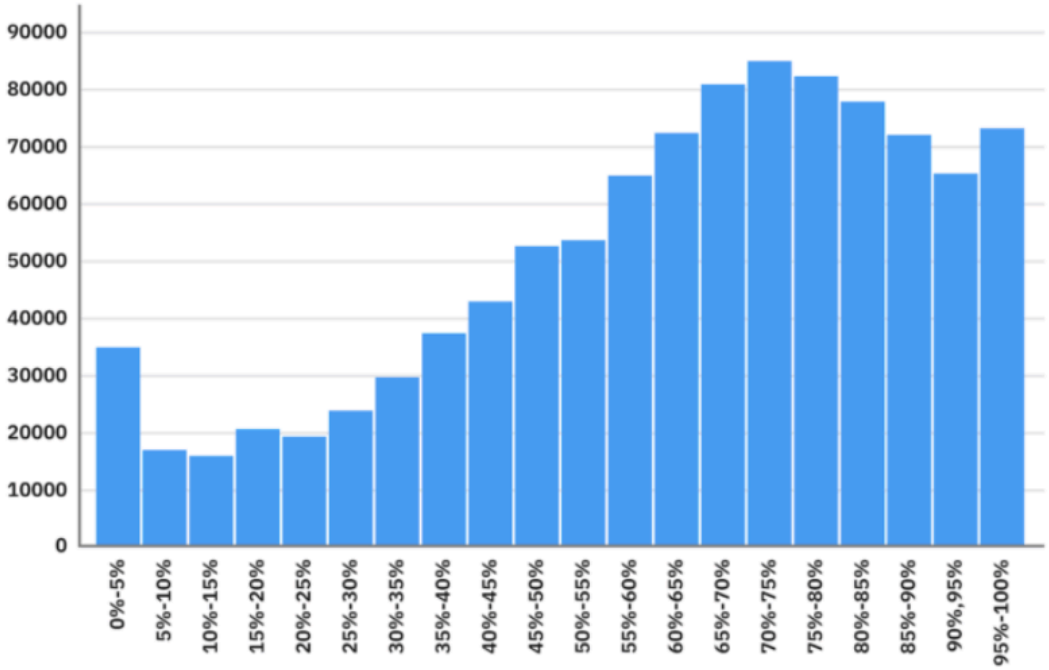

Figure 3: Link-rot percentage on sites with more than 10 live links [12]

This examination spanning nine years indicates that a significant portion, at least 66.5 %, of the sampled links have fallen victim to link-rot. Figure 3, shows link-rot by domain, with more than 10 alive links, since small sites do not have much link-rot. Where the y-axis is the number of websites with more than 10 alive links, and the x-axis is the percentage of link-rot on those pages. Figure 3 shows that larger sites have quite a bit of link-rot [12].

Additionally, a study carried out by an information scientist at the Los Alamos National Laboratory Research Library in New Mexico, in collaboration with researchers from the Hiblink project, examined 3.5 million articles published by Elsevier between 1997 and 2012 [18]. Elsevier serves as a platform for facilitating knowledge sharing and discoveries [19]. Figure 4 depicts the correlation between the probability of link degradation and publication year. Figure 4 shows that in 2012 alone, 22% of links within Elsevier articles were found to be broken, compared to approximately 60% in 1997 [20]. Figure 4 shows that links that originate from older publications are more prone to become rotten [21].

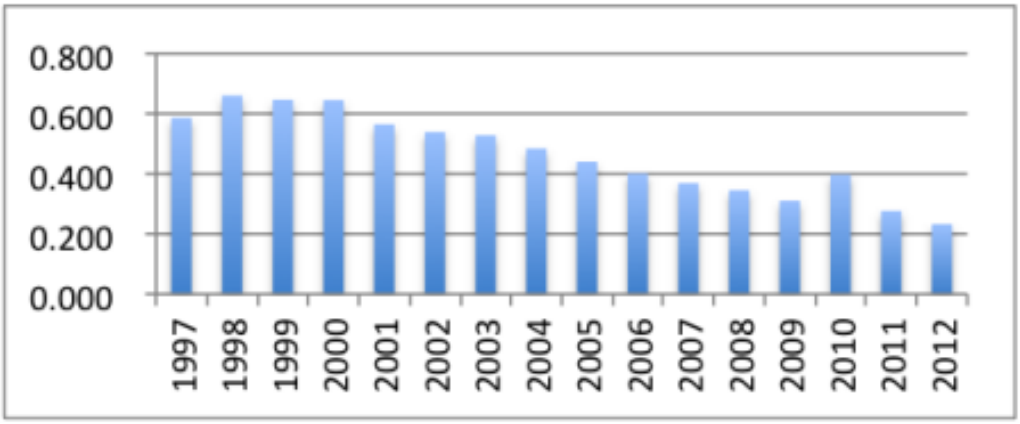

Figure 4: The percentage of links likely to be rotten in relation to publication year [21]

The studies about link-rot underscore its presence and highlight the need for proactive measures to address this pervasive issue in the digital landscape [22].

2.3 Implications of Link-rot

Link-rot poses a significant challenge for website owners, publishers, content creators, and users. This section looks at the many consequences of link-rot, analyzing how they affect credibility, disrupt user experience, and hinder search engine optimization efforts.

- **Loss of credibility**:
  Numerous occurrences of link-rot on a website can give the impression that the content of the website is outdated or not properly maintained. Furthermore, link-rot conveys a sense of unreliability. Users rely on links to navigate the web and access information efficiently. When these links fail to deliver the expected content, users may question the website's reliability. This skepticism can significantly undermine the website's reputation and discourage users from returning to the website in the future [13].

- **Poor user experience**:
  Link-rot significantly negatively affects a users browsing experience as it can prevent access to internet resources the user could find valuable. When users encounter link-rot, they are unable to reach the intended destination, leading to frustration and dissatisfaction. Moreover, it disrupts the flow of search for information and may discourage further interaction with the website [13].

- **Search Engine Optimization (SEO) and Web Ranking**:
  Websites with a lot of link-rot face significant challenges in terms of search engine optimization (SEO) and web ranking. Search engines like Google prioritize user experience and content quality when determining a website's ranking in search results. Broken links can signal to search engine algorithms that a website is poorly maintained or outdated, which can result in a negative impact on its SEO performance. When search engine crawlers encounter broken links on a website, they may interpret them as a sign of low-quality content or a lack of attention to site maintenance. As a result, the website's overall ranking in search results may suffer. In addition, broken links can disrupt the crawling and indexing process, making it more difficult for search engines to accurately assess the relevance and authority of the website's content. Therefore, websites with a high number of broken links are likely to experience a drop in both traffic and search engine rankings [13].

The consequence of broken links is a spectrum of challenges that affect credibility, user experience, and search engine optimization. Each aspect is interconnected, with broken links disrupting the seamless flow of information and hindering the effectiveness of websites. Before looking into existing solutions for combating link-rot, centralized-, decentralized-, and distributed systems will be explained.

2.4 Centralized-, Decentralized- and Distributed Systems

In computer science, a centralized system refers to a system in which all decisions are made by a single node [23]. A node refers to a computing element, whether it is a high or low-performant machine, and this definition will be used going forward [24]. In a decentralized system, no node is the central command node, allowing all nodes to independently make decisions. This is also known as a partially distributed system and uses multiple nodes to fulfill its requirements [23]. Lastly, according to Maarten van Steen from the university of Twente and Andrew S. Tanenbaum from the Vrije university Amsterdam, a distributed system is defined in the article, "A brief introduction to distributed systems" as such:

"*A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system*" [24].

To get a better understanding of the difference between central-, decentral-, and distributed systems, take a look at Figure 5. On the left-hand side is the centralized system. The single decision-making node is in the middle and could consist of multiple machines running concurrently with each other. This means that if each node is given a specific task and the nodes depend on each other, the system is still centralized. This means that it depends on the system architecture [23].

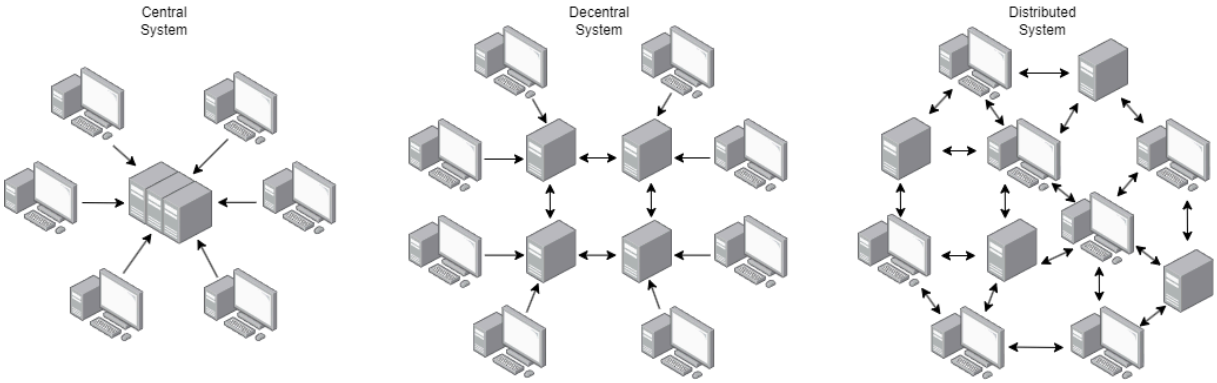

Figure 5: Central-, Decentral-, and Distributed system comparison. The computers without a screen are nodes, and the computers with a screen are users.

In the middle, a decentralized system is visualized. As seen in Figure 5 there are also multiple computers in this system, but they act independently of each other. In this system, users will not necessarily access the same node, which is illustrated with all the users arrows not pointing to the same node. In this specific scenario, the system consists of 4 nodes, and should 1 of the nodes fail, the traffic would be redirected to one of the other nodes. Thus, all 4 nodes must fail for the system to fail completely [23]. Furthermore, this also indicates that no single entity is responsible for storing the data.

Lastly, on the right-hand side of Figure 5, a distributed system is visualized. As seen in the figure all entities in the system are nodes and they are all independent of each other, distributing the decisions to all the devices in the system. Even if there is only a single node left in this

system, it will still work. It might be a lot slower, but the system will not fail before all nodes have failed [24].

Now that the distinguishment between central-, decentral-, and distributed systems has been clarified, the pros and cons of the different system architectures will discussed to determine which system would be more intuitive for addressing link-rot.

2.5 Centralized- vs. Decentralized- vs. Distributed System: Pros and Cons

There are several advantages to centralized systems. Firstly, updates can be executed swiftly due to the central management of the system. Secondly, centralized control makes enhanced coordination and decision-making easier, thanks to the authority consolidated in one entity. Moreover, the management process can be more straightforward with a single central node, easing maintenance tasks. Lastly, the implementation process of a centralized system can use simpler algorithms and protocols, contributing to a smoother deployment overall [25].

On the other hand, centralized systems also come with several drawbacks. If nodes lose connection to the central node a likely outcome is system failure. Moreover, scalability is limited as the central node can only handle a finite number of clients, restricting adaptability for growing demands [25].

Property	**Centralized**	**Decentralized**	**Distributed**
Points of Failure	One central point of failure	Mulitple independent central points before system failure	All nodes have to fail before system failure
Maintenance	Easy	Morderate	Difficult
Vulnerability	Vulnerable to downtime	Less Vulnerable to down time	Almost not vulnerable to down time
Scalability	Low scalability	Good scalability	Great scalability
Complexity	Less Complex	Moderate Complex	Very Complex

Table 1: Pros and Cons of central-, decentral-, and distributed systems modified from [26]

A decentralized system is also known as a partially distributed system due to its nature of having multiple points of failure. Decentralized systems are less vulnerable to downtime comparted to a centralized system, due to having multiple nodes working independently. It is also moderately easier to scale up the system, since you can add more nodes when the workload becomes too much for the existing system.

There are also some disadvantages to a decentralized system. Due to it having multiple nodes, it can be more complex to set up, since it has to run smoothly on all nodes. The maintenance of a decentralized system can also be more complex due to it having multiple nodes needing updates at the same time.

Finally, a distributed system also has several advantages. Bottlenecks are minimized since the workload can be evenly distributed across all nodes, resulting in few to no instances of bottlenecking. Furthermore, high availability is ensured as certain nodes, including computers, mobile

devices, and servers, remain consistently online and accessible for tasks, thereby maintaining uninterrupted availability. Moreover, scalability is enhanced by adding nodes, which boosts capacity and distributes workload efficiently to meet growing demands [24].

A con of distributed systems is the complexity of setting up the system and making sure communications protocols and coordination mechanisms are set up correctly. Therefore managing a distributed system can be difficult if it is not structured carefully. Distributed systems are also not well-suited for small scale setups because their cost/benefit ratio is low. Additionally, there is a lack of node regulation without superior oversight, leaving behavior potentially unchecked. Lastly, identifying failed nodes can be challenging, requiring individual checks and workload partitioning [24].

Now that the different system architectures have been discussed, the current solutions for combating link-rot will be described.

2.6 Current Solutions

Having established that link-rot is an issue and looked at centralized-, decentralized-, and distributed systems, the next step is to uncover the adversities in current solutions. There already exist multiple solutions to counter link-rot. To describe the current solutions, they will be categorized into three groups: free web archives, paid storage solutions and self-hosting (see Table 2).

Free Webarchives	**Paid Storage**	**Self-Hosting**
Wayback Machine [27] Archive.today [28]	Perma.cc [29] Stilio [30]	Archive Box [31] Cyotek [32] Pocket [33]

Table 2: Categorization of current solutions

Webarchives

Wayback Machine and Archive.today have in common that they provide access to an archive of older versions of websites and do it completely free. They are examples of initiatives that preserve sites on the web and other digital cultural artifacts.

Wayback Machine is a non-profit organization funded through donations, grants, and by providing web archiving services for their partners. Wayback Machine is focused on building a digital library of the internet, as well as other digital cultural artifacts and currently takes up more than 145+ Petabytes of server space. It is a solution controlled by one entity, but they store at least 2 copies of everything. Currently, the Wayback Machine stores more than 835 billion webpages. On a user level, Wayback Machine is simple to use. A user just has to supply a link to the website, to be stored, and the site then attempts to store the webpage. If it is successful in storing the webpage, the archive will supply the user with a link to the stored site [27]. Furthermore, Wayback Machine also has its own extension, that allows for easy saving of webpages to Wayback Machine. In addition, their extension makes it possible to automatically store any webpage a user visits, as long as it has not previously been saved to Wayback Machine. Furthermore, a user can view all versions of the webpage stored by the Wayback Machine, allowing for an easy way to check for altered content [34].

Similarly to Wayback Machine, Archive.today is an internet archival site, but run by an individual instead of an organization and privately funded [28]. Similarly to Wayback Machine,

the site also stores duplicates of its content among different data centers, ensuring some safety in regards to data storage. Like Wayback Machine, it is simple to use since a user only has to supply a link to the website to be stored.

The webarchives as a solution has both pros and cons. Both Wayback Machine and Archive.today are free services that are simple to use. All that is required is for a user to know the address of the webpage they want to backup, everything else is handled by the archive website. One potential problem that may arise from the use of web archives is that if any of the services decide to shut down or discontinue their archiving services, all the links created would rot. A potential example of this could be the service Webcite, which is a site that focuses on academic links [35]. They have not been taking new archiving requests since 2018 [36], although viewing already stored links is still possible. This makes their future uncertain, and a journal like the Journal of Medical Internet Research (JMIR) that used to require Webcite links, no longer does so [36]. Another potential example is Wayback Machine. Wayback Machine has an extensive archive of different literature, such as articles, webpages, and 3.6 million copyrighted books [37]. They are currently in a lawsuit and now have the potential of losing some of their archive. This shows that a solution where the data is in other people's hands is vulnerable to legal implications [38]. Another significant issue with these services is the control exerted by website owners. Owners of websites can prohibit services from archiving their pages, effectively removing them from their services [39]. This allows website owners to selectively delete specific pages from the archive, removing the concept of ensuring that links work forever.

To summarize, the currently available online archives offer a valuable service for accessing older versions of webpages but have the drawback of being controlled by other people. Therefore, susceptible to external pressures, making users vulnerable to losing their links.

Paid Solutions

There are multiple paid solutions for storing webpages or their content, such as Stilio and Perma.cc. A solution such as Stilio is targeted toward tracking changes on webpages [30]. In contrast, a solution such as Perma.cc [29] is a website storage solution, and like Webcite [35] it is targeted mainly towards academics. Perma.cc specializes in the storage of webpages, and the creation of permanent links for use in articles and reports [29].

Stilio is a solution that is based on automatic website screenshots, that allows a user to track changes to websites [30]. Stilio is targeted towards companies and is used for different cases, compared to the web archives, but both allow users to see previous versions of webpages. Some of the use cases, that Stilio themselves highlight, are content verification, competition tracking, and website compliance [30]. The service functions by the user supplying the site with the links they want to follow, and specifying when they want screenshots taken. Stilio will then automatically take screenshots, at the specified time(s) that are then stored on their Stilio account, or transferred to their preferred storage option [40]. Unlike, The web archives this way of preserving a website is a personal solution as only the user has access to the stored content unless further steps are taken to publicize the material. Furthermore, Stilio does not guarantee that their solution will capture every webpage or element of a webpage [40]. However, it does make it simple to track changes to a webpage over time.

Perma.cc is a website and storage solution developed, and maintained, by the Harvard Law School Library in collaboration with other university law libraries in the US and other organizations in the "forever" storage business. It is a service targeted specifically at the issue of link-rot by making it possible to preserve online sources, and make those records available to view even after the original source is gone [29].

Users affiliated with academic libraries partnered with Perma.cc can have free unlimited usage plans. For users unaffiliated with a Perma.cc partner, the website functions by the user making a account on the webpage, which gives a trial of 10 free storage links. If the user want additional storage/links, they will either have to pay a monthly subscription fee for a set amount of monthly links, or pay upfront for a specific amount of links. A user can then generate permanent links to the webpages of their choosing by inputting the URL, they want to save, which causes Perma.cc to download the material of the selected webpage, and generate a Perma.cc link that the user can then use for their reference. Following the link, it takes the user to a Perma.cc site with the downloaded content, the date of link creation, and a link to the source of the material. Perma.cc also have their own chrome extension that makes it simple for users to create and manage their Perma.cc links directly from the browser [41], [29].

Like the Webarchive solutions, usage of Perma.cc has both pros and cons. The service itself is relatively simple to use, and if a user is affiliated with one of their partners it is free to use. The main drawback of using Perma.cc is the cost, if the user are not affiliated with one of their partners who have free plans available to them. Furthermore, Perma.cc is an option other people control so if the site ever shuts down, users risk losing their links. However, this issue is somewhat addressed by the contingency plan, they have made in case the project ever winds down. Currently, one aspect of this contingency plan is that every link stored by Perma.cc results in an archive file, that they will replicate to available third-party sites such as the internet archive described previously. In the case of a shutdown, they also plan for a two-year phaseout period, where they will initially freeze link creation but continue to serve existing links. Furthermore, in this period, they also plan to provide tools for users to move to new hosts and try to find hosts for existing files [42]. However, while Perma.cc has this contingency plan for shutdowns, they also explicitly state in their terms of service [43] that they reserve the right to suspend or terminate the website, at any time with no notice and without accepting any liability for doing so.

Self-Hosting and other solutions

Another approach, to combating link-rot, is to store the content yourself. If this tactic is employed solutions such as ArchiveBox, Cyotek and Pocket can help a user. These solutions are programs designed to retrieve online webpages onto a local device, allowing users to store the information for however long they want to.

Archive Box is an open-source self-hosted web archiving tool, that makes it possible for users to download and view content offline. The user supplies the software with a URL for the site to be stored, and the tool then downloads the content of the webpage and preserves it inside a folder [31].

Similarly to ArchiveBox, Cyotek webcopy [32] is a free tool for downloading a website onto a local device. It will try to "crawl" the website to discover all linked resources and will download any resources found and continue to search for more [32].

Pocket is a Chrome extension that allows a user to press a button on they web browser to store a webpage for later viewing [33]. After saving the content the user just have to go to the extension and press saves to be brought to a website containing a list with links to all their saved pages. Furthermore, a user can then choose to archive the webpage, in which case pocket will save an offline version of the webpage, which the user can then keep for however long they wish. In addition, the archived webpage also contains a link to the original webpage should it still be available. Pocket also allows the user to control settings such as the font used on the archived page, font size, and the webpage background color. Furthermore, a user can also

highlight different parts of the text on the webpage that they feel is especially important. This highlighting then both highlights the text on the webpage but also adds the text to a sidebar on the left side of the webpage.

Like the previously mentioned solutions, self-hosting comes with both pros and cons. On the positive side, self-hosting allows a user to keep the content they want to save, safe for as long as they are willing to keep it. Furthermore, if a user wants to, they can share the content with other people as long as they keep a server active and available. On the negative side, self-hosting introduces economic barriers and includes technological complexities in the form of upkeep and maintenance of servers. It is not inexpensive to have a server running to host your references. From acquiring the correct hardware to the electrical bill for constantly running the server. Alternatively, the user could rent a server to host the content, but in that case, it could make more sense to utilize an external service such as Perma.cc since they include this service.

This makes the Archive Box and Cyotek rather impractical if users want their references to be stored online forever and more of a solution for storing references for their own use like for Pocket.

Pros and Cons of Current Solutions

The current solutions for combating link-rot have both upsides and downsides. If a person wants their content to be available to other people they either have to keep it available for other people themselves or use a service such as Perma.cc or Wayback Machine. Usage of those services are generally simple. Wayback Machine and Archive.today do not even require a user to make an account to store the webpage of interest and only requires a user to have the link to the webpage they want to save. On the other hand, a solution like Perma.cc does require a user to have an account and to either be connected to an affiliate organization with access to perma.cc or pay themself for storing their content. However, usage itself is simple once you have access. Likewise, the solutions that allow the user to download the webpages to their own computer, such as cytotek or Pocket are also relatively simple to use. Archive Box and Cyotek do require a user to download an external tool which makes them a bit more involved to use, but Pocket allows a user to store a webpage directly from their browser through their extension. These ways of countering link-rot also comes with the downsides, such as that a user must have faith that the people behind the solution will keep it running. While solutions like Wayback, Archive, and Perma.cc, have multiple copies of the stored information, the solutions are still vulnerable to site shutdown, as mentioned in their respective sections, whether due to external pressures, legal trouble, financing, or unforeseen events such as bad actors or catastrophes. A more controlled solution is to self-host the content, but as described in the self-hosting section, this solution comes with other downsides, such as the cost, the resources required, or server malfunctioning.

Leaving your content in other people's hands leaves you vulnerable to data loss and downtime, and an example of such is the fire OVHcloud data center in Strasbourg [44]. OVHcloud is the largest European cloud solution provider [45] and in 2021 a fire broke out in their data center in Strasbourg. This fire caused downtime for several popular sites [46] and a data loss for many international companies [47].

Furthermore, a solution you do not directly control can also be taken down by bad actors, for example, the ransomware attacks on the Danish cloud hosting sites CloudNordic and AzeroCloud in August of 2023. The attacks led to the encryption of data and data backups and caused extensive data loss [48], [49].

Currently, only solutions where you either leave the data in other people's hands or spend a lot of resources to host it yourself, are available on the market and as the previous examples show, those types of solutions to link-rot come with a set of downsides. These downsides have the traits and limitations of a centralized system architecture. These limitations underscore the importance of exploring other systems, such as decentralized and distributed approaches, to address the complexities of link-rot.

The different current solutions and their pros and cons have been discussed. Now the process of encountering link-rot will be discussed.

2.7 The Process of Encountering Link-rot

To address the issue of link-rot, it is important to identify where this problem occurs. This makes it possible to isolate the problem on a larger scale. In Figure 6, the process of using links to access webpages is visualized. This helps illustrate the connections between certain systems, and how the system as a whole is interconnected.

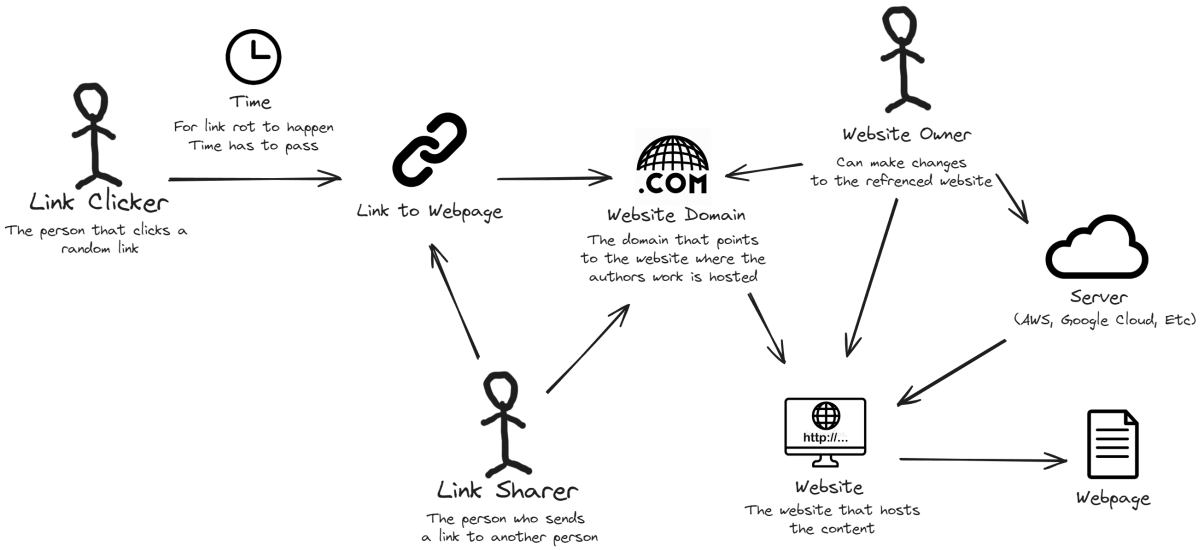

Figure 6: Rich Picture depicting the process of using links to share webpages

To understand the rich picture, the different terms along with the arrows meaning will be explained

Webpage
A webpage is a specific hypertext document on a website containing information such as text, images, or links. Hypertext is a markup language for making webpages [50].

Website
A website is a collection of webpages all hosted under the same domain usually controlled by a website owner [51].

On the rich picture the website has an arrow pointing to the webpage, since this is where the webpage is located.

Website Owner
A website owner is a person, who owns a website thereby giving them control over the website both operationally and legally.

The website owner has arrows pointing to multiple things. The arrows resemble the things, which the website owner is in control. The owner has the power to change the website domain, server host, and the content on the website itself.

Server
A server is a constantly running computer making it possible for people to access a website through e.g. the World Wide Web.

The server has an arrow pointing to the website, since this is where it is hosted.

Website Domain
A website domain is the name of the website you type into the search bar, when accessing a specific website. A website owner typically buys a domain for the website making it more attractive for users.

The domain has multiple arrows pointing to it, and one point away from it. The link sharer has an arrow pointing to it, which means the linker sharer accessed the domain at some point. The link to the webpage points to the domain, since the link accesses the webpage through the domain. Lastly, the domain points to the website, where the webpage can be found.

Link to Webpage
A link to a webpage is a URL that points to a webpage. A URL is a uniform resource locator and is an address pointing to a location on the web. This can and usually consists of the domain followed by the specific webpage name [52].

The link to the webpage has links from the clicker and sharer and is pointing to the website domain. The links pointing to it resemble people clicking on it, and the one away from it the routing happening for its way towards the webpage.

Link Sharer
A Link sharer is a person with a link to a specific location on the web, who wants to share it with another person.

The link sharer has two arrows, one to the domain and one to the link. This is because the link has been obtained on the webpage, which was accessed through the domain.

Link Clicker
A link clicker is a person who has received or found a link to a specific location on the web and clicks on it.

The process of link sharing

Initially, a link sharer will find a link they want to share by accessing a webpage on a website referred to by a domain. The link sharer can then share the link with a link clicker through e.g. an email, a messenger service, etc. After some time, the link clicker wants to access the linked webpage, they will click the link and access the specific webpage on the website, hosted by a server. If this process works as intended, the link clicker can view the linked webpage. However, sometimes the link clicker will run into link-rot, due to different factors.

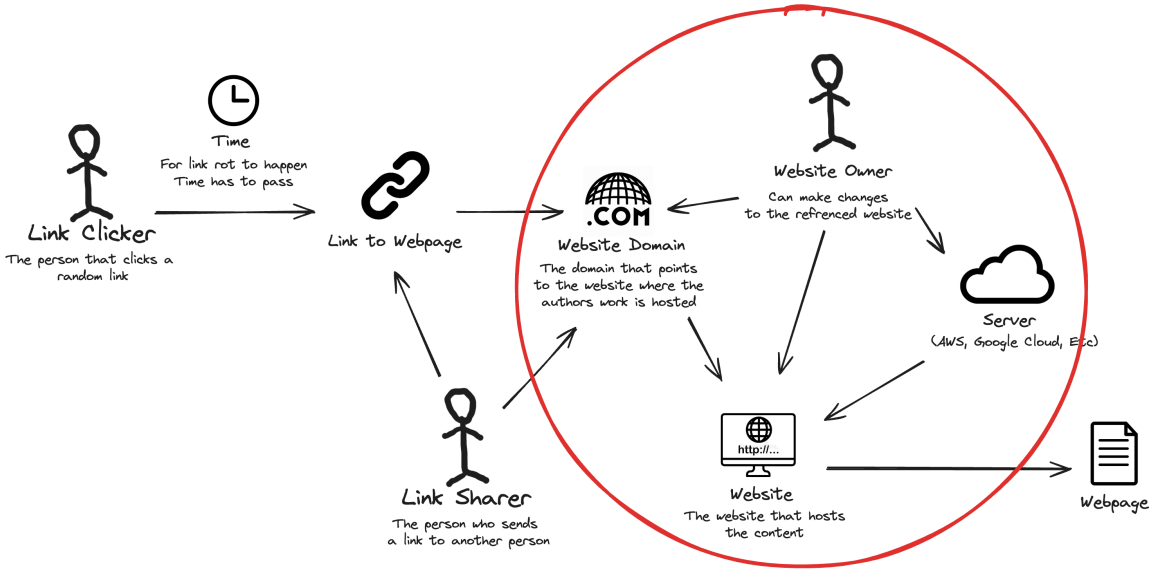

Figure 7: Rich-Picture with problem circled in red

Where link-rot happens

When the link clicker runs into link-rot there can be multiple reasons why the link is unavailable. To access the linked webpage the link clicker first has to interact with a website domain and then the website that contains the linked webpage. Furthermore, the website itself is likely to be hosted by a service like AWS, Google Cloud, ect. And lastly, the website is controlled by a website owner.

Each of these four steps presents a potential cause of link-rot. If the website domain is down or the website owner no longer wishes to pay for the domain, link-rot will occur. Likewise, the website owner can choose to take down the website that contains the webpage and stop paying for the server hosting service, which will also result in link-rot. Moreover, the website owner can choose to change the website to no longer contain the specific linked webpage which also leads to link-rot. In addition, the server host can also experience an outage or problem causing temporary link-rot.

In short, from the moment the link sharer has noted down the link that worked for them, link-rot could occur for multiple reasons. A solution must be placed between the link sharer and the webpage itself to combat the issue.

“Solving” link-rot

Currently, all solutions that have been examined have not solved link-rot directly. Instead, the existing solutions make a backup of the webpage and ensure the longevity of the backup. The original link can still stop working, and the link clicker must find the backup themselves. The backup works around the unpredictable actions and lifetime of the server, domain, website, and website owner.

To completely solve link-rot, everything circled in Figure 7 would have to be maintained for all time. Therefore, solving link-rot entirely is unrealistic. The approach of the existing solutions is viable and works well for selectively preserving important links.

The current solutions are websites and browser extensions, which you can give a link you want to store, and they then store a backup of the webpage. However, it does not fix the vulnerability of a single entity being in control of the data. It just adds an extra way of accessing the content. To address this, a possible solution could include a decentralized or distributed system to store the digitalized content to be saved.

3 Problem Statement

In the age of digitalization, the preservation of digitalized content relies heavily on digital citations and hyperlinks. However, this reliance has revealed a significant vulnerability known as link-rot. The multifaceted nature of link-rot involves aspects of web architecture, content management, technological advancements, and even legal and ethical considerations, all contributing to the decay of links over time.

From the moment a link is shared, link-rot can occur. Various factors such as domain, hosting, and website stability play a crucial role in its occurrence. Currently, there are several ways link-rot is combated. A user can utilize free web archives, paid storage solutions, and self-hosting strategies, each with their own inherent limitations. These limitations make them vulnerable to data loss, legal issues, and operational vulnerabilities. Instances like the OVHcloud data center fire and the ransomware attacks described on the previous chapter, illustrate the fragility of current solutions. This shows the need for alternative methods to ensure the permanence and reliability of digital links.

While current solutions provide temporary relief, they do not address the root cause of link-rot, which is the transient nature of digital content and the predominately centralized architecture that manage them.

Problem Statement:

"*How can we develop a resilient webpage preservation system to mitigate the risks of link-rot, ensuring the sustained accessibility and integrity of the linked content?*"

4 Requirements - MoSCoW model

Keeping the problem statement in mind, the report's objective is to develop a solution that addresses link-rot by preserving online information. This chapter will outline the necessary requirements for solving the problem, by using the MoSCoW model. The MoSCoW analysis is a prioritization technique used to manage requirements. It categorizes them into four groups: must-have, should-have, could-have, and will not have. These categories represent the varying degrees of significance the different requirements have to the solution.[53]

Identifier	**MoSCoW**	**Requirement**
1	Must have	A web-based front end
2	Must have	Download the HTML of a webpage to main memory
3	Must have	Upload the webpage to a decentralized network
4	Should have	Allow other user to share uploaded webpages
5	Should have	Share the HTML file as a static webpage on a decentralized network
6	Should have	Include images from the webpage in the uploaded content
7	Should have	Include CSS from the website in the uploaded content
8	Should have	Make it possible to stop sharing specific uploaded content
9	Could have	Include JavaScript in the uploaded content.
10	Could have	Include a progress bar for downloading content
11	Could have	Upload a digital representation of the webpage
12	Will not have	Allow an organization to easily share all uploaded webpages
13	Will not have	Website interface with list of all previously generated links
14	Will not have	Automatically upload and create links for visited webpages
15	Will not have	Compression of webpage content
16	Will not have	Upload all webpage links referenced on the uploaded webpage

Table 3: MoSCoW requirements

5 Technology Analysis

Before looking at the design and implementation of the solution, we consider the technologies that are relevant to the application and look at what type of front-end is suitable for the solution.

The technologies used in this project — TypeScript, React, Valtio, Plasmo, and the InterPlanetary File System (IPFS) — were chosen primarily because a group member already had experience with them. This familiarity ensured that the capabilities of these technologies could meet the requirements of the solution.

5.1 Extension vs Website

In the context of developing a solution to combat link-rot, the decision between creating a browser extension versus a traditional website is a crucial consideration. This subsection looks into the advantages and disadvantages of choosing either an extension or a webpage as a front-end.

Cases for webpage preservation

For a web-based solution, the user will have to navigate to the website and manually copy and paste the link of the webpage they wish to preserve. This approach typically requires multiple steps: opening a new browser tab, accessing the webpage to preserve, copying the desired link from the browser's address bar, and pasting it into the input field on the preservation website. While effective, this method can be time-consuming, especially if the user needs to preserve multiple links frequently. It also interrupts the user's current browsing session.

For an extension-based solution, the user clicks on the extension icon and selects the "backup" option to preserve the current webpage they are viewing. This approach significantly streamlines the process. With a single click, the user can preserve a webpage without needing to leave the webpage they are on or manually copy and paste links. This method offers a more seamless and efficient user experience, allowing for quick and easy link preservation. Furthermore, it minimizes disruption to the user's browsing activity.

Sharing and storing

The link generated for the preserved webpage can be shared in various ways such as via email, social media or other messaging features. The difference between an extension and a website is in their way of operating and managing resources.

A website hosted on a server has centralized storage controlled by a website operator. While this may offer robust storage and management capabilities, it also introduces dependencies on the website's infrastructure and policies. Users rely on the website's features and policies for sharing and managing their content. They may have limited control over access permissions and ownership rights, as these are determined by the website operator.

With a Chrome extension, sharing content onto the IPFS network, lets the content get distributed across multiple nodes. Each node maintains a copy of the content, ensuring fault tolerance.

Data Retrieval and User Location Concerns

The process of data retrieval presents another point of differentiation. For websites, the server is tasked with retrieving and serving the content to the user. This scenario poses complications, especially when considering the geographical location of the server relative to the user or differences in browser compatibility. An extension, by operating directly within the user's browser,

sidesteps these concerns, providing a more seamless and efficient data retrieval process that is agnostic to the user's location or browser choice.

Ease of Use and Scalability
Extensions seamlessly enhance user experience by integrating directly into the browser interface, allowing users to access their functionalities without navigating away from their current tasks. This integration fosters a more intuitive user experience, enhancing user engagement and satisfaction.

From a scalability perspective, websites typically employ vertical or horizontal scaling to manage increased loads, both of which involve significant complexity and cost. Extensions, however, leverage client-side computing, distributing the computational load across the user base. This model not only reduces the reliance on server-side resources but also scales more effortlessly as the user base grows, given that each new user inherently contributes additional computational resources.

Considerations and Trade-offs

While extensions offer distinct advantages, they also present their own set of challenges. The initial version of the extension may be limited to Chromium-based browsers, potentially excluding users of other browsers. Moreover, extensions rely heavily on client-side code execution, which, if not carefully managed, could introduce security vulnerabilities.

Despite these considerations, the inherent scalability of extensions, where computational efforts are distributed among the users, presents a compelling advantage. This approach has no server-side load and therefore enables the extension to scale as more users join the network, each contributing their computing power to the collective.

In conclusion, while both websites and extensions have their respective merits and drawbacks, the advantages of an extension — particularly its local hosting, efficient data retrieval, ease of use, and client-side scalability — underscore its potential as a superior solution for combating link-rot in the digital ecosystem.

Taking the tradeoffs into consideration as well as the choices of existing solutions, the group have decided to move on with making a Chrome extension for the solution.

5.2 Chrome Extension

When developing a chrome(chromium) extension there are different tools to be utilized. For the development of this solution content/background scripts and Plasmo are the most relevant for the use case.

Content script
Content scripts are JavaScript files that run in the context of webpages. By doing so, they can read or modify the details of webpages visited by the user. Here are key aspects of content scripts:

- **Interaction with Webpages**:
  Content scripts can interact directly with the web content of the pages the user visits. They can manipulate the DOM (Document Object Model), change styles, and respond to events on the page. This makes them powerful tools for modifying the user experience on any given webpage, such as auto-filling forms, highlighting text, or altering page layout.

- **Isolation from Webpage's JavaScript**:
  Although content scripts can interact with the DOM of a webpage, they do not share the JavaScript environment with the webpage. This means variables and functions defined in content scripts are not accessible from the webpage's scripts and vice versa, which provides a layer of security and prevents interference.
- **Communication**:
  Content scripts communicate with other parts of the extension (like background scripts) using messaging APIs provided by Chrome. This allows them to send and receive messages to perform higher-level operations like storing data or fetching information from external APIs, actions that content scripts can not do directly.

Background script

Background scripts act as the central processing unit of Chrome extensions. They run in the background, independently of webpages, and manage the extension's overall behavior. Here are some important aspects of background scripts.

- **Long-running Processes or Event Listeners**:
  Background scripts are ideal for handling long-running processes or persistent state information across different parts of the extension. They listen for events from the browser, other parts of the extension, or external triggers (like alarms or timers).
- **Interaction with Browser APIs**:
  Unlike content scripts, background scripts have access to a broader range of Chrome APIs. This allows them to perform tasks like modifying browser settings, interacting with tabs, and using bookmarks or the browser history.
- **Communication**:
  Background scripts also handle communications between various components of the extension and external servers. They typically manage message passing between content scripts and popup scripts, maintain local storage, and handle network requests outside the extension's scope

Communication between background and content

Content scripts and background scripts often need to communicate to coordinate their actions. For instance, if a content script needs to save data persistently or fetch external data, it sends a message to the background script, which then performs these operations. The interaction is typically managed through Chrome's messaging API, which includes methods like "chrome.runtime.sendMessage()" and "chrome.runtime.onMessage.addListener()".

In the coming sections, the technologies used to develop the extension will be explained.

5.3 Typescript

TypeScript is an open source programming language developed by Microsoft that builds on JavaScript by adding static typing options [54]. This means developers can define the types of variables, parameters and return values, improving code reliability and catching errors at compile time rather than at runtime [55].

TypeScript supports object-oriented programming features such as classes, interfaces and inheritance, making it suitable for building complex applications. It is designed to be aligned with the ECMAScript standard, which JavaScript is based on. This means it supports features from the latest versions of JavaScript (ES6, ES7, etc.) while adding its own enhancements [55].

Its tool support, including features such as code completion and type checking, improves developer productivity. By providing type annotations and supporting modern JavaScript features, TypeScript promotes more readable and maintainable code, especially in larger code bases where understanding types and structure of data becomes crucial [54].

All in all, it aims to address JavaScript's limitations while retaining its flexibility and familiarity.

TypeScript was chosen over Javascript due to its improved developer experience.

5.4 React

React is a javascript library designed to streamline the development process of graphical interfaces. Furthermore, it offers developers a robust toolkit to build dynamic and interactive user experiences. The key features of React are its component-based architecture and Virtual Dom rendering, which lead to simplicity and efficiency.

The component-based architecture of React, allows developers to create reusable interface components, promoting modularity and extensibility within applications. Additionally, React encapsulates functionalities into distinct modules promoting a modular design paradigm. A component in React can represent various UI elements such as a button. These components are essentially JavaScript functions or classes that return the React element, specifying what content should be displayed on the screen [56]. By breaking down complex UIs into smaller, self-contained building blocks, React promotes code reusability and maintainability, ultimately resulting in cleaner and more organized codebases [57].

React leverages a Virtual Dom to optimise rendering performance. Instead of directly manipulating the real DOM, React utilizes a lightweight and efficient abstraction known as the Virtual DOM. The Virtual DOM serves as a JavaScript representation of the actual DOM tree. The way the virtual DOM works in React involves three main steps: Initial rendering, Virtual DOM drifting and Updating [57].

When a React component is initially rendered, it creates a corresponding Virtual DOM tree mirroring the structure of the Browser DOM. Subsequently, whenever there is a change in the component's state, React compares the previous Virtual DOM tree with the new one and identifies the minimal set of changes required to update the Browser DOM. Whereafter, it updates the browser DOM, where React applies the necessary changes to update the Browser DOM efficiently. Instead of re-rendering the entire DOM tree, React only updates the specific parts that have changed, minimizes the reducent re-renderings, and improves overall application efficiency [58].

Overall, React is a powerful tool for modern web development, offering developers comprehensive solutions for creating dynamic and responsive user interfaces, even in resource-intensive applications. Moreover, React empowers developers to build scalable, maintainable, and high-performing web applications with ease.

5.5 Valtio

Managing the state effectively is crucial in React applications. Valtio is a state management tool with ease and flexibility, leveraging proxies to provide a mutation-style API for managing state in React applications. At its core, Valtio utilizes proxy classes to serve as an interface to another object. These proxies "wrap" the object, allowing for additional operations to be performed before returning the object. This mechanism enables Valtio to track changes to the state object and all nested objects, notifying whenever an object is modified [56]. Valtio is a lightweight and user-friendly state management library for React, which offers advantages:

- **Reactive updates**: Leveraging Proxies, Valtio enables reactive updates, ensuring that state changes are automatically reflected in the UI without the need for manual intervention.
- **Performance**: With its small footprint and high performance, Valtio is well-suited for projects requiring optimal performance, providing smooth user experiences even in resource-intensive applications.
- **Flexible data structures**: Valtio have flexible data structures such as maps and sets, facilitating easier manipulation of complex data within React components.
- **Easy Integration**: Valtio seamlessly integrates with other libraries and tools, including React and TypeScript, simplifying the development process and enhancing productivity.

In today's modern applications, effective management of state information is essential. Valtio emerges as a powerful tool in this regard, owing to its straightforwardness and robustness. Furthermore, it harnesses the capabilities of proxies in React and JavaScript, offering a simplified approach to modifying and managing state data [56].

5.6 Plasmo

Plasmo is a toolbox that gives the right set of tools for developers to build extensions efficiently. Extensions are small programs running inside a browser e.g. Google Chrome, that adds a feature to the browser, thereby extending its functionality.

What makes Plasmo great is its ability to cooperate with already known programming languages such as React and Vue.js. Plasmo lets programmers use these already existing languages instead of having them learn a completely new language, which saves them a lot of time. In addition to not having to learn a new programming language, it also has live reloading, which makes the developing process much more fluent since the extension automatically reloads every time you make a change to the code.

Plasmo also supports the programming language typescript, explained earlier in the technology section, which helps catch errors faster in the coding process, leading to more stable code [59]. To conclude, Plasmo is a toolbox that gives developers the right tools to build extensions efficiently.

5.7 InterPlanetary File System (IPFS)

The InterPlanetary File System (IPFS) is a protocol and peer-to-peer (P2P) network that allows users to store and share data in a different way than traditional storage and sharing solutions [60]. The advantage of using IPFS in the solution is that it is already developed, making it simpler to use than developing a new peer-to-peer network and storage solution. Furthermore, IPFS offers several advantages such as resiliency, safety, and efficiency. As data is stored decentralized in a network of nodes it is less vulnerable to outages and censorship than centralized solutions. As a cryptographic hash is used to identify content, a user can always be sure of the data integrity of the content they are accessing. The use of content focused addressing facilitates efficient data retrieval. Moreover, as IPFS is an open-source protocol that encourages participation, it further fosters innovation and continuous improvement of the network [61], [62].

5.7.1 The structure of IPFS and key concepts

There are several key concepts to IPFS that will be explored in this section:

- The IPFS Node system
- Representing and addressing data
- Routing data
- Seeding
- Transferring data
- Gateways
- Pinning
- Anti-duplication
- Garbage collection
- Deleting content

The IPFS node system

At the core of the IPFS network are the nodes. They are the fundamental building blocks that make IPFS function. These nodes represent instances of IPFS implementations that operate on local machines, either directly or through a browser interface, enabling storage of files and connection to the IPFS network. Nodes serve as the backbone of the IPFS ecosystem, and is essential for the network to function [63].

The term "node" permeates throughout IPFS documentation, discussions, and codebase, albeit with varying connotations depending on the context. There exists multiple types of nodes but the main type of node relevant for this project is the IPFS nodes [63]. IPFS nodes represent an individual point within the network, implying a general presence and potential for interaction with other nodes. For instance, upon launching IPFS Desktop, a user establish themself as a node capable of engaging with the broader network. The term peer is used to indicate the relationship of a node, including one's own, with other nodes as equals, devoid of central authority. In this peer-to-peer dynamic, a node operates as a peer among peers [63].

Representing and Addressing Data

Unlike URLs that point to a file's location on the web, IPFS uses Content Identifiers (CIDs) to address data based on its content. CIDS can be thought of as fingerprints for data and is, in general, computed by combining the cryptographic hash of the data with a unique identifier for the hash algorithm used. This ensures that data can not be altered without changing its CID, promoting data integrity and security. By default, IPFS uses the sha-256 hashing algorithm, but there is support for other algorithms [62], [64], [65].

IPFS uses InterPlanetary Linked Data (IPLD), a set of specifications in support of decentralized content-addressable data structures for the web, to structure and link CIDs. Using IPLD, IPFS can structure data into a directed acyclic graph (DAG) that allows the data to be broken down into smaller chunks called blocks, where each chunk is linked to others using a unique fingerprint. This allows for efficient storage and retrieval and makes sure everything stays consistent. IPLD allows for flexible data representation, accommodating various data types beyond just files and directories [64].

Content Addressable Archive (CAR) files, is a type of compressed archive file similar to zip files and is designed to store collections of content-addressed data and is used to store and transfer collections of IPLD-linked data efficiently [64].

Routing Data

To locate data based on its CID, IPFS employs a combination of several mechanisms. Kademlia Distributed Hash Tables (DHT) helps nodes find peers with the CIDs they seek. Essentially, it is the fundamental component of the content routing systems, and it acts like the cross between a catalogue and a navigation system and maps that have the content addressed by the CID. Functionally, the DHT is distributed across the network as no single node holds the entire table. When a user wants to "upload content" to the IPFS network, they add an entry to the DHT that maps the CID to their IP, thereby announcing to the network that they have the content. Another user could then look up that CID in the DHT and find the uploaders IP address, and download the content from the uploader [62], [66].

IPFS nodes use the Bitswap protocol to route and transfer data. This peer-to-peer protocol allows nodes to directly query their connected peers for specific CIDs, bypassing the DHT when possible. In addition, peers also store want-lists so that if the peer receives the content later, they can then send it to the node that originally requested it [64].

IPFS also uses delegated content routing, which is a mechanism by which IPFS implementations offload content routing to another process/server using an HTTP API. This provides IPFS nodes with a standard interface that allows flexibility regarding how content routing works [64], [67].

Seeding

One of the essential things that keep files on IPFS available is seeding. The IPFS utilizes users as nodes to keep files on the IPFS network available for other users. This is referred to as seeding. The intended use of seeding is for users to seed their own file when they upload it, making it available for other users. Other users can then begin to seed the same file as the creator and make the system more fluent, increasing the chance of keeping it available at all times [68].

Transferring Data

Once a node locates the desired CID, data transfer can occur through several methods. As mentioned earlier, IPFS nodes use Bitswap, a message-based peer-to-peer network protocol for the routing and transfer of data. Bitswap can facilitate the direct transfer of blocks of data between connected peers. The two main jobs of Bitswap are the acquisition of blocks requested by the client from the network and in reverse, sending blocks that it possesses to other peers who want them. When a node wants to retrieve a file from the network, it sends out a want-list to its connected peers. The want-list is a list of CIDs for blocks that the node wants to retrieve. Based on responses from the network, Bitswap then builds up a map of which nodes have and do not have each block. It then uses this map to request the block from nodes that have it, and they respond with the block itself. If no peers have the block, Bitswap will query the DHT and ask who can provide the block. Verification ensures the integrity of the fetched blocks before constructing the Merkle DAG, thereby ensuring trustless retrieval from any node in the network. With all blocks retrieved the Merkle DAG can be reconstructed which makes the directory or file underlying the CID, available for use [64], [69], [70].

Gateways

To allow applications that do not support or implement all IPFS subsystems to fetch data, IPFS HTTP gateways are used. Gateways achieve this by translating CIDS into URLs accessible via standard web browsers. When a user requests content through an IPFS gateway, the gateway dynamically retrieves the requested data from the IPFS network and serves it to the user's browser as if it were hosted on a conventional web server. This process allows the user to interact with IPFS content without needing specialized software or knowledge of the underlying protocol [64], [71].

For instance, consider the following IPFS link:

ipfs://bafybeic3y6oc2dai3uypyyuaggp4xx3krocpgzbwst2z4ha73jdh7y6nea

To view this content in a browser, an IPFS gateway provided by Cloudflare is used. By prepending the gateway URL to the CID, a link can then be accessed via any standard web browser:

https://cloudflare-ipfs.com/ipfs/bafybeic3y6oc2dai3uypyyuaggp4xx3krocpgzbwst2z4ha73jdh7y6nea

The final link will be fetching the IPFS content through the Cloudflare gateway and making it possible to view it in any modern web browser.

Pinning

Pinning plays a vital role in ensuring the availability and persistence of data on the IPFS network. When content is pinned, it is saved on an IPFS node or pinning service, making it retrievable to the network. A pinning service runs many IPFS nodes and allows users to pin data on those nodes for payment. Pinning involves advertising and providing. Through advertising, a CID is made discoverable to the IPFS network by linking it with the node's IP address in the DHT. Advertising is a continuous process that typically repeats every 12 hours. Providing involves a web3.storage IPFS node (a server running an IPFS node) and ensures that the content-addressable representation of the CID is persisted on IPFS nodes [69], [72].

Anti-duplcation

Anti-duplication is a fundamental feature of IPFS that enhances storage efficiency by avoiding the redundancy of storing identical data multiple times. When a file is added to the IPFS network, it is broken down into smaller blocks, each identified by a unique cryptographic hash. If the network already contains blocks with the same hash, IPFS reuses these existing blocks instead of creating new ones. This process ensures that only unique data is stored, significantly reducing the amount of required storage space. By using the combination of content-addressing and cryptographic hashing, IPFS can integrate anti-duplication into its operations, optimizing data storage and retrieval across the distributed network

Garbage Collection

Garbage collection in IPFS involves the identification and removal of data on individual nodes that is no longer pinned or referenced by any node in the network. When a file or content is pinned on an IPFS node, it essentially ensures that the data remains available and retrievable. However, there are cases where data is no longer needed or relevant. In such cases, nodes need to perform garbage collection to reclaim storage space by removing unpinned data. The garbage collection process typically involves scanning the local repository of a node to identify and mark unpinned data [72], [73].

Deletion

Deletion is the final stage in the lifecycle of IPFS data, where blocks associated with a CID are removed from a node. Notably, deletion is always a local operation, meaning it only affects the node from which it is initiated. If a CID has been replicated on other nodes, it continues to be available on the IPFS network, ensuring data redundancy and availability [69].

The next section will delve into the design of the solution.

6 Design

In Figure 7, the area where the problem occurs is circled. Therefore a system will be designed to work its way around it.

The software solution will work in a similar fashion to some of the other previous solutions, except it will utilize the IPFS peer-to-peer protocol as described in section 5.7. The decision to use IPFS it is based on its reliability and decentralized architecture, which helps to prevent single point of failure and address other drawbacks of centralized systems, as described in section 2.5. The solution will be making a backup of the original webpage and making a new decentralized link that the link clicker, and sharer, can access.

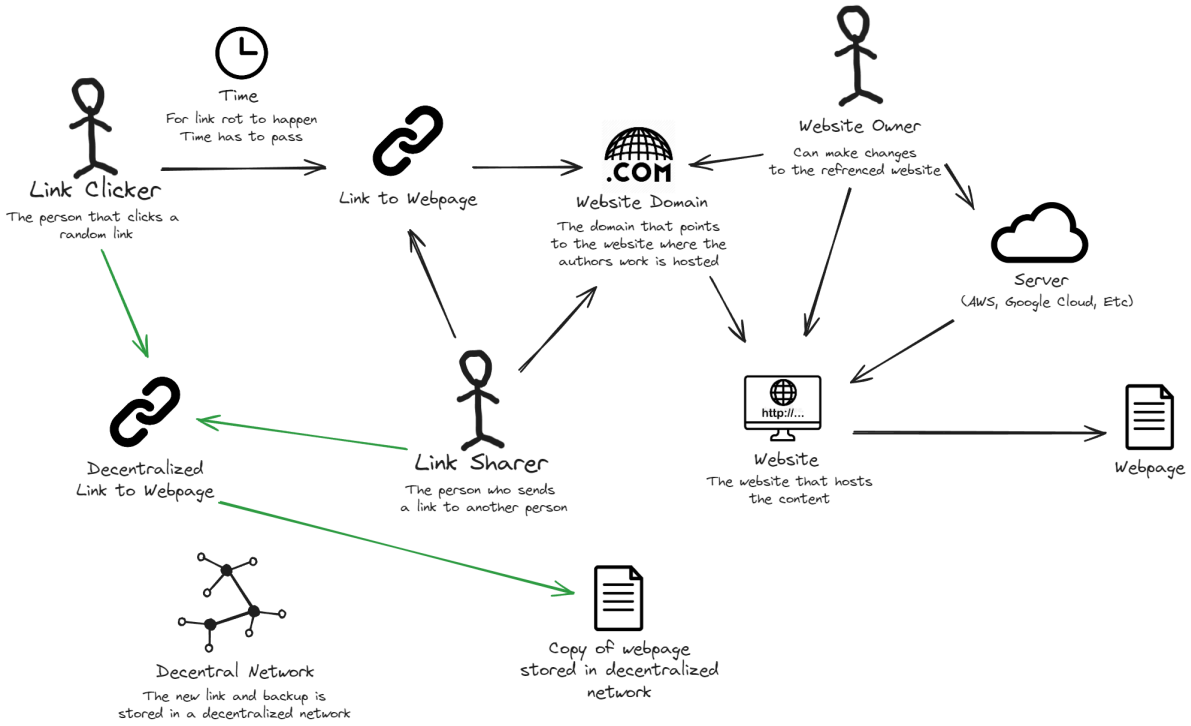

Figure 8: Rich-Picture with decentralized backup solution

In Figure 8, an updated rich picture can be seen with a proposed solution highlighted with the green arrows. The solution includes a backup of the webpage, which is stored in a decentralized network, and a new link is shared between them. This solution will not ensure the lifetime of the original link but instead, make a new link that will last forever as long as the decentralized network is kept alive.

For this solution to work, the link sharer has to make the backup of the webpage and upload it to the decentralized network. This solution will not address link-rot for existing links, but it will preserve webpage content for as long as users find it valuable. The next section will describe use and UI of the chrome extension.

6.1 User Story

The UI of the Chrome extension can be viewed in Figure 9, and is designed for anyone who values the permanence and accessibility of online content. Users ranging from researchers to casual internet surfers can benefit from ensuring that the webpages, that they value, remain accessible over time despite the ever-present threat of link-rot.

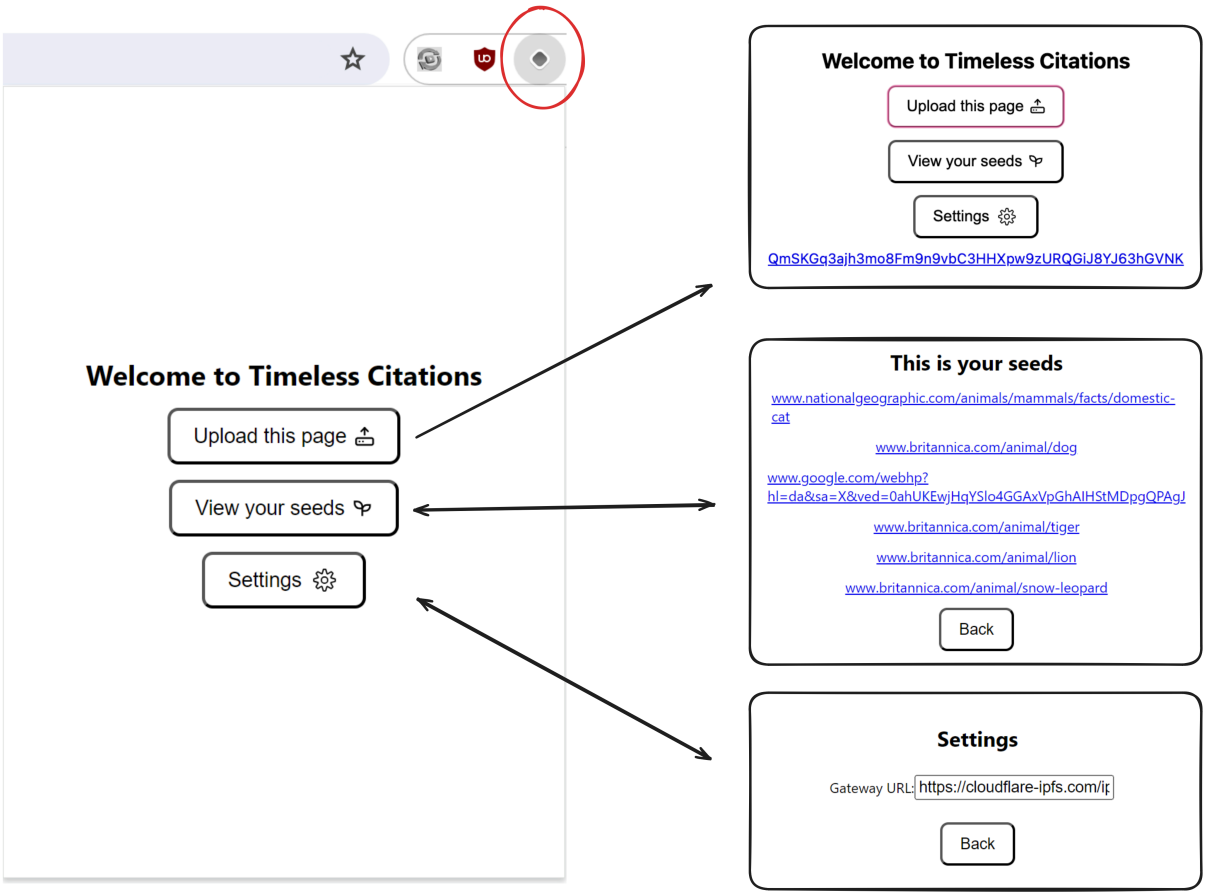

Figure 9: The UI of the extension. The extension has three buttons on its main page allowing users to upload pages, view their seeds and change settings.

The user journey begins when someone learns about the Chrome extension, perhaps through a recommendation or a search for solutions to prevent link-rot.

When a user finds a valuable webpage they wish to preserve, such as an academic article or a favorite blog post, or a random webpage, they navigate to the webpage and click on the extension icon (Marked by the red oval in Figure 9). If they then select the “Upload this page” option, the extension packages the content of the webpage and uploads it to a decentralized network.

Once the upload is complete, the extension generates a new, decentralized link to the backed-up content (Figure 9, right side top box). This link is designed to be permanent, providing a reliable means of accessing the webpage indefinitely.

Users could then share these decentralized links, ensuring that their audience can access the content as long as they are willing to share it. In addition, the extension also allows users to see the links they have uploaded (Figure 9, right side middle box) or change the gateway used to see uploaded content (Figure 9, right side bottom box). Furthermore, the extension also have

back buttons on the "View your seeds" and "Settings" pages that allow a user to return the main page when clicked (Figure 9, right side middle and bottom box).

6.1.1 General overview of the flow of the extension

The flow and general structure of the extension are illustrated in Figure 10. As mentioned and illustrated in the previous section, when the user opens the extension they are presented with the three choices illustrated to the left in Figure 9 and by the select action box in Figure 10. If the user presses "Settings," they are directed to a page where selected settings can be changed. If they input a new gateway in the gateway URL field seen in Figure 9 bottom right box, they can change the gateway used by the extension. If they press back, they are directed back to the page which allows them to select another action. If they press "view your seeds" they are directed to a page, where they can see their current seeds, and can then press back to return to select another action. If they press "Upload this page" in their current tab is downloaded by the extension, uploaded to the decentralized network, and then shown a link to their uploaded content. This is the main functionality of the extension, and it allows it to fulfill the must-have requirements 2 and 3, the ability to download said website down as an HTML file to main memory, and the ability to upload the HTML file to a decentralized network.

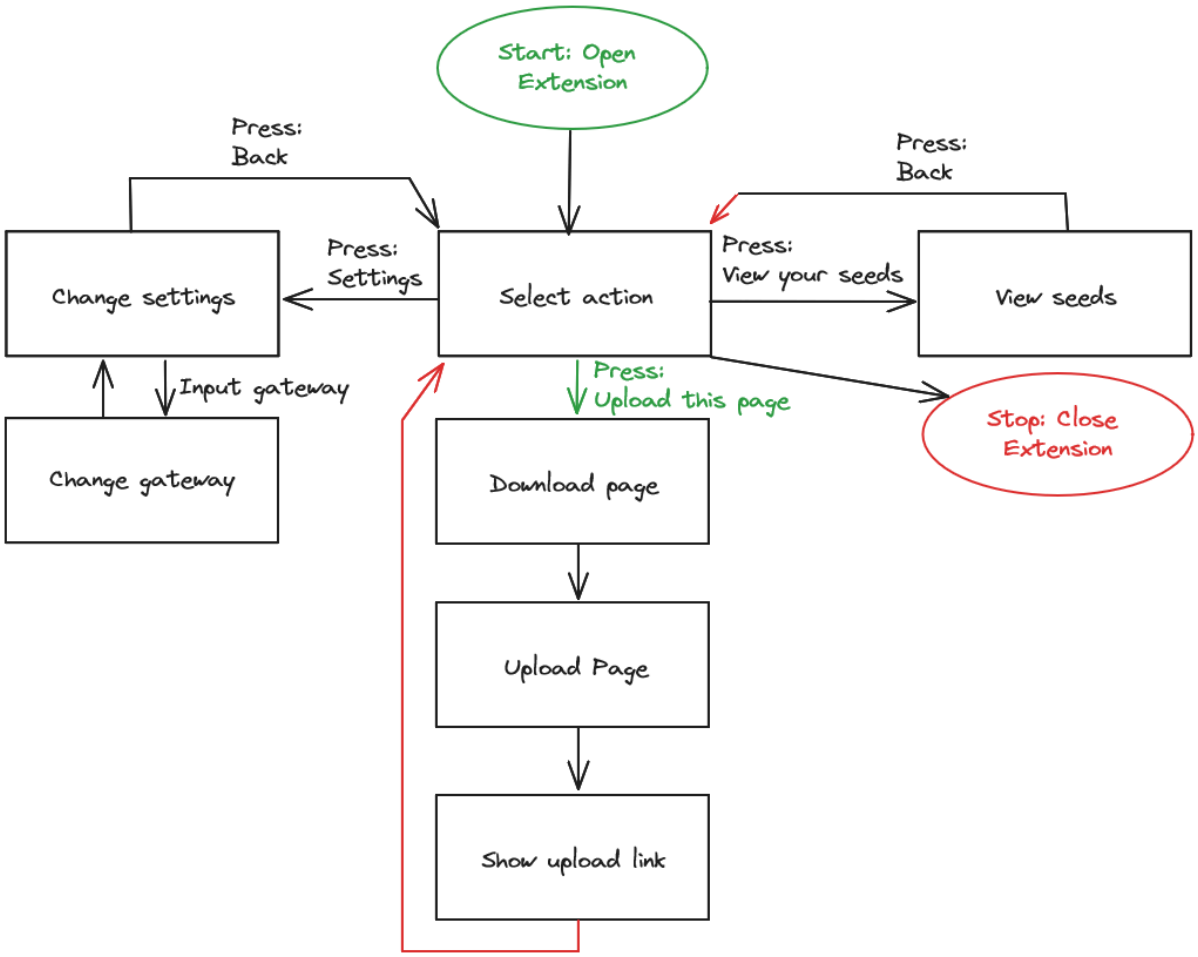

Figure 10: General flow usage of the extension

6.1.2 The "Upload this page" flow

To further expand on the flow of the upload this page choice the user can make from the main page, this choice is further illustrated with its own flow diagram.

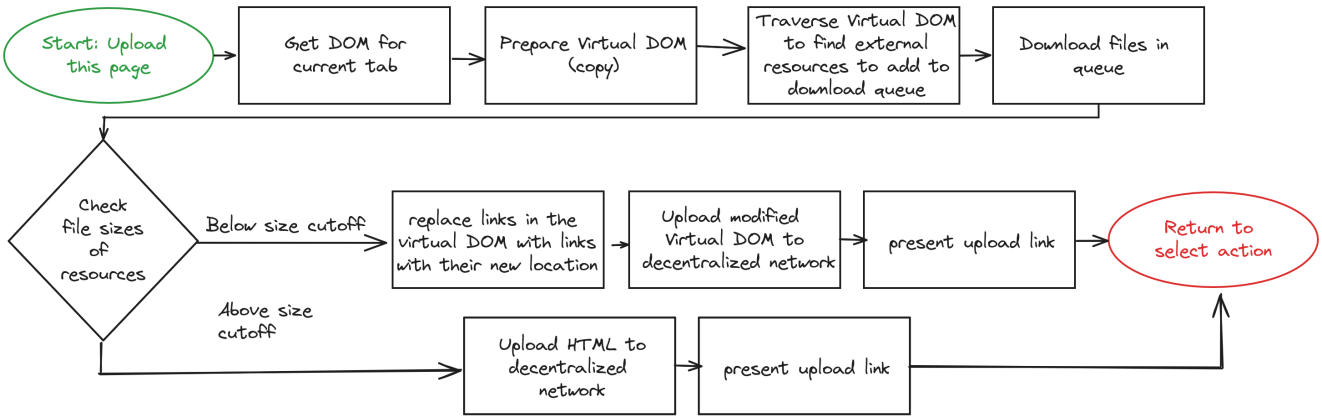

Figure 11: The flow when a user presses upload page

When the user presses "Upload this page" they will follow the flow illustrated in Figure 11. The extension will get the DOM of the page in the currently selected tab. It will construct a virtual DOM based on the information collected from the DOM, and this virtual DOM will then be traversed by the program to find resources that needs to be downloaded from external sources. These resources are then added to a download queue and downloaded if possible. The program will then check whether the total size of the downloaded content is below or above a size threshold to prevent too big pages from being uploaded (illustrated by the diamond in the flow diagram). If the total size is above the threshold, only the page HTML is uploaded, and the extension flow returns to allowing the user to select action. If the total size is below the threshold, the links to content in the virtual DOM are modified to their new relative locations, and it is uploaded to a decentralized network. The user is then presented with a link to the uploaded content. The flow then returns to allow the user to select an action.

The relevant technologies as well as the design choices will be used in the implementation.

7 Implementation

This chapter will delve deep into the architecture and mechanics of our solution. At its core, our solution revolves around the ability to package web content, and seamlessly upload it to the IPFS network. The functionality is pivotal in mitigating the pervasive risk of link rot, ensuring the longevity and accessibility of digital content. Additionally, the solution will be integrated into a Chrome extension, facilitating a streamlined process for users to preserve web content, thereby fulfilling requirement 1.

The solution is developed with the technologies, Typescript, React, Valtio, Plasmo and IPFS, as outlined in Chapter 6. Collectively, these technologies form a powerful toolkit that accelerates the development process and enhances code quality.

The solution's core functionality lies in its capability to facilitate seamless communication among three key components: the background script, popup script, and content script. By establishing robust communication channels between these components, the solution ensures cohesive and synchronized behavior, enabling a seamless user experience.

7.1 Overview

Before delving into the implementation, an overview of the key functions, workflow, and important data structures will be described. The Figure 12 illustrates the overall workflow for the scripts (represented by white blocks), state management and external webpage (represented by grey blocks), along with the pages and important functions (represented by the different use of blue blocks).

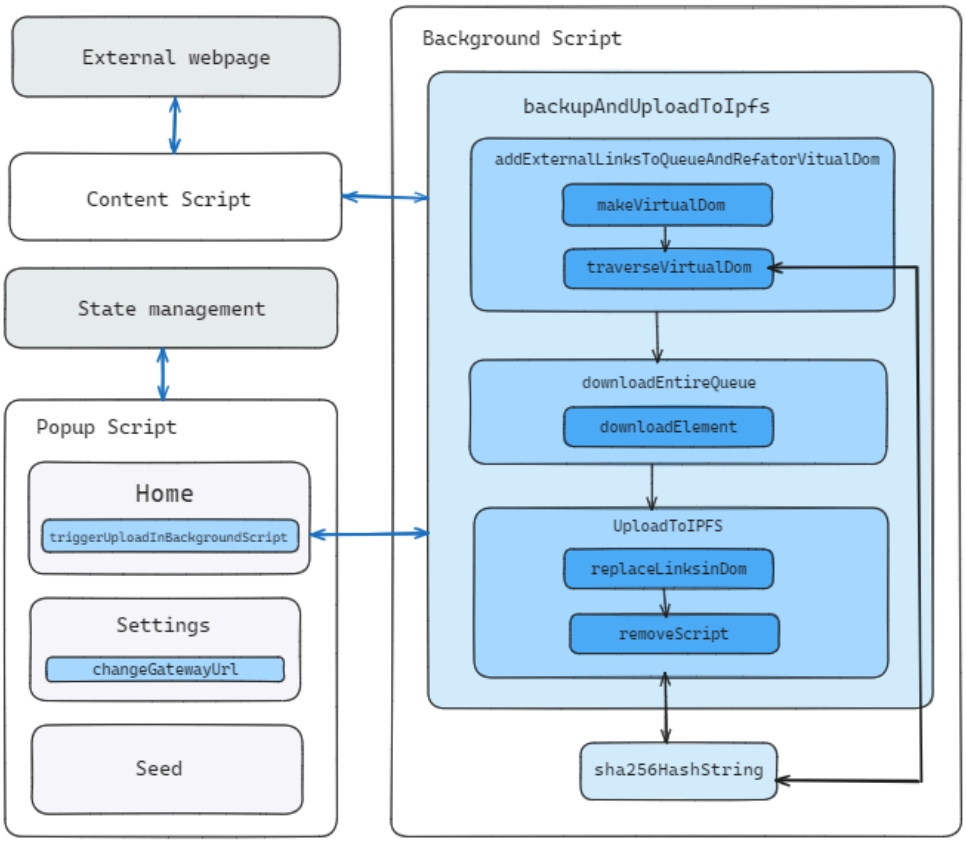

Figure 12: Figure that shows the general workflow of solution

The popup script consists of three pages: Home, Settings, and Seed. The figure graphically represents the code architecture through nested boxes, illustrating pages and functions contained in the scripts. Moreover, the nested structure of the blue blocks illustrates functions contained within functions. The black arrows depict the general flow of the functions, while the double-edged arrows indicate bidirectional flow, such as a function call followed by a returned value. Blue arrows emphasize communication between different scripts but also with the state management and external webpage. All of the functions in the figure will elaborated in the following sections.

When the user clicks the "Upload page" button, the **triggerUploadInBackgroundScript** function on the Home page is invoked, initiating the **backupAndUploadToIpfs** function in the Background Script. This function fetches the DOM string of the webpage through the Content Script, which has access to the external webpage. After retrieving the DOM string, the function **backupAndUploadToIpfs** processes it. Whereafter the HTML file together with the images and CSS are uploaded to the IPFS network, ultimately displaying the link for the uploaded content on the Home page. The external sources will be hashed through the function **sha256HashString**, since the folders for each of the external sources will be removed. The function is based on the hashing algorithm SHA-256 and the hashed value serves as a safeguard against files having the same file names but different paths associated with them.

The following functions in the **backAndUploadToipfs** are:

- **addExternalLinksToqueueAndRefactorVirtualDom**:
  The function creates a virtual DOM from the provided DOM string and traverses it to replace all image and CSS links with hashed URLs using the **sha256HashString** function.
- **downloadEntireQueue**:
  Function will download all the links from the images and CSS in the DOM.
- **UploadToIpfs**:
  Function will upload HTML and associated files to the IPFS network.

The Settings page features the **changeGateAwayURL** function, enabling users to modify the gateway URL. Any alterations made to the gateway URL are saved within the state management system and subsequently applied to URL when a webpage gets uploaded

Important data structures are:

- **downloadQueue**:
  array in Background Script containing objects with properties to be downloaded such as url, original url, new file name and a blob, which is a datatype used to represent binary data. Blobs can store binary data such as images [74]. The array will store links that will be downloaded. Queuing the downloads helps prevent overwhelming the system by limiting the number of concurrent downloads, which can lead to issues like network congestion.
- **Cid**:
  An object within a file for state management, featuring properties for both the seeded URL and the total website URL.

The library features used in the code of the solution will now be explained.

7.1.1 Library Implementation Features

This section will cover the importation of key features from various libraries. Each imported feature will be thoroughly described, highlighting its functionality. One of the important features used in React is hooks. Hooks give access to state and other React features without writing a class [75].

Features from React:

- **useState**:
  State hooks let a component remember information. It accepts an initial state as a parameter, and returns two values: the current state and a function that updates the state. This syntax is also called array destructing in JavaScript [76].
- **useEffect**:
  The effect hook performs side effects in a function component. Examples of side effects are data fetching, setting up a subscription, and directly updating the DOM in React components. Some side effects in React components require a cleanup [77]. The useEffect is a cleanup function that accepts two arguments, the first a function and the second an optional dependency. The purpose of cleanup function is to prevent unwanted behaviors, such as memory leaks [78].

Features used from the Valtio library will be described

- **useSnapshot**:
  Creates a local snapshot of the current reactive state object created with proxy. This hook enables components to access the current state snapshot, automatically triggering re-renders to reflect any updates to the state object. The snapshot returned can be used in the React component [79].
- **subscribe**:
  Method to access state and subscribe its change outside of the component. The function takes two parameters a state object, which is a reactive state object. The state object is created using Valtio's proxy function. The second parameter is a callback function, that will be invoked when the subscribed state object changes [80].

Now that the overview of the solution has been reviewed, the focus must shift to a key algorithm used within the solution.

7.2 Pseudo Code for traverseVirtualDOM

This section covers the primary algorithm used in the solution for downloading a webpage into memory. The aim of the function is to go through all elements in the DOM and replace the linked sources, such as images and CSS files with new hashed paths, to combat duplicates when uploaded to IPFS. Let us say there are two images with the same name but with different parents. Since we upload all the files in the same directory to IPFS these images would be interchangeable with each other. Therefore we hash the full path of the image and since they have different parents these will be unique hash values.

The DOM consists of nested elements with the possibility of elements having multiple children. This nesting of multiple children inherently forms a tree-like structure with nodes being both parents and children. This can be visualized in the image below of what Google.com looks like when you right-click the browser and inspect it.

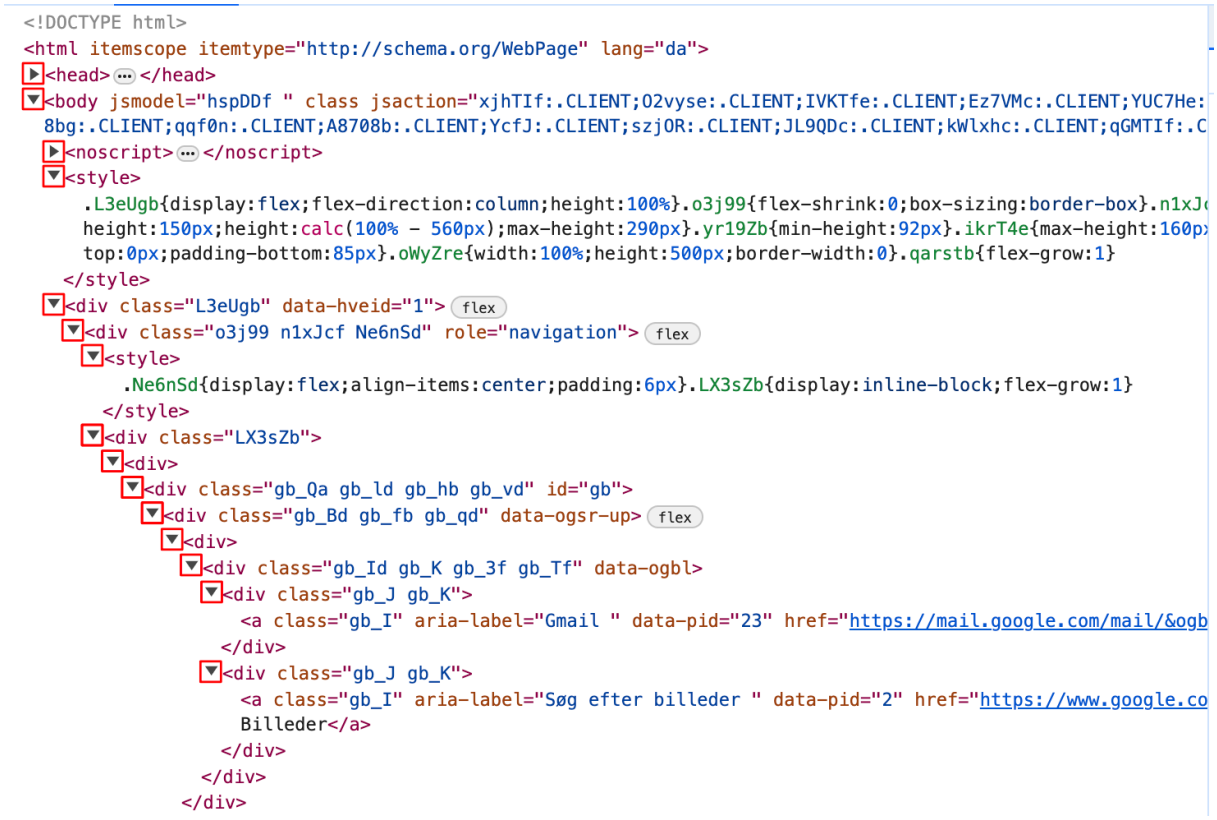

Figure 13: Image of the Google.com DOM

As seen in Figure 13 all the red squares are nodes with children and each child only has one parent. The relationship between parent and child has to fulfill the requirement of a child only having one parent, but a parent can have multiple children.

This is why it makes sense to use a recursive algorithm since it makes sure to go all nodes by recursively calling itself if a node has a child. The traverseVirtualDOM takes in a virtual DOM of the webpage you want to upload. The function traverseVirtualDOM then traverses through the tree structure and replaces all the images or links with new hashed paths.

To give an explanation of how the algorithm recursively calls itself, the pseudo-code for the function will be given below:

```
1                                                                    PseudoCode
2   function traverseVirtualDOM(node, origin = ""):
3
4                   [...] Assigns attributeToReplace
5
6                   [...] Finds the URL of the attribute
7
8                   [...] Checks if the URL works
9
10                  [...] Hashes the URL
11
12                  [...] adds URL to the download queue
13
14      if node has children:
15          promises = empty list
16          for each child in node.children:
17              promise = traverseVirtualDOM(child, origin)
18              add promise to promises
19          await Promise.all(promises)
20
```

Listing 1: Simplified Pseudo Code of traverseVirtualDOM

As seen at the end of the pseudocode on line 17, the function traverseVirtualDOM will recursively call itself on each child for a given node. The function traverseVirtualDOM will then use the child as an input thereby making it a new parent and this continues until the function runs out of children. This is how it recursively calls until the whole virtual DOM has been searched. The full functionality of the function will be explained in more detail later. The next sections will go through the implementation of the solution.

7.3 State Management in Chrome Extension

When developing a Chrome extension, the need for a seamless way to manage and update the application's state across various components arises. This becomes particularly crucial when dealing with information spread across different scripts, such as Content Scripts, Background Scripts, and Popup Scripts. In the context of our extension, essential information like the Content Identifier (CID) and the Gateway URL need to be accessed and updated from different parts of the extension. Moreover, it helps to streamline the management of the extension's state. The code snippet is from the global file "store.ts"

```typescript
import { proxy, subscribe } from "valtio"

type Cid = {
  seedUrl?: string
  websiteUrl?: string
}

const defaultStore = {
  gatewayUrl: "https://cloudflare-ipfs.com/ipfs/",
  cids: [] as Cid[]
}

const loadAndCombineStore = () => {
  const store = JSON.parse(localStorage.getItem("p2")) || defaultStore
  return { ...defaultStore, ...store }
}

export const store = proxy(loadAndCombineStore())

subscribe(store, () => {
  localStorage.setItem("p2", JSON.stringify(store))
})

```

Listing 2: Code snippet showing the Valtio store

Explanation:

- **Line 1**: Imports *proxy* and *subscribe* from the Valtio library. *proxy* is used to create a reactive state object, while *subscribe* is used to listen for changes to the state.
- **Line 3 - 5**: Defines type *Cid*, which is an object with properties *seedUrl* and *websiteUrl*.
- **Line 8 - 10**: Defines *defaultStore* object with properties *gatewayUrl* set to a default IPFS gateaway URL and *cids* initialized as an empty array of *Cid* objects
- **Line 13 - 15**: Defines the **loadAndCombineStore** function, that loads the state from the *localStorage*. If the stored state is not found, it falls back to the *defaultStore*. It returns an object combining the default store with the loaded store, to ensure any missing properties in the loaded store are filled with default values
- **Line 18**: Creates a reactive state object using Valtio's *proxy* function. The proxy is initialized with the result of the function **loadAndCombineStore**. The store object holds the application's state and automatically tracks changes.
- **Line 20 - 21**: Is a subscription mechanism, that subscribes to changes in *store* object. Whenever the *store* object changes, the callback function is invoked. Inside the callback function, the *store* object is serialised into a JSON string and stored in the local storage under the key "p2".

7.4 Click Upload Page Button

In order to package the web content, the background scripts need to know when a user clicks on the button "upload page" on the home page. Here is a code snippet from "pages/home.ts".

```
1  const Home = ({ setPath }: { setPath: (string) => void }) => {                React
2     [...] // set path for page
3     const [isError, setIsError] = useState(false)
4     const [error, setError] = useState("")
5     const [cid, setCid] = useState("")
6     const [ipfsLink, setIpfsLink] = useState("")
7
8     const triggerUploadInBackgroundScript = () => {
9       chrome.runtime.sendMessage({ message: "upload" }, (res) => {
10        const { cid: _cid, websiteUrl, error } = res;
11        setError(typeof error === "object" ? error.message : error)
12        setIsError(!!error)
13        if (!error && _cid) {
14          const cid = {
15            seedUrl: _cid,
16            websiteUrl: websiteUrl
17          }
18          setCid(_cid)
19          store.cids.push(cid)
20          setIpfsLink(`${store.gatewayUrl}${_cid}`)
21        } else {
22          setIpfsLink("")
23        }
24     })
25  }
26    return (
27      <div className="extension">
28          [...] // h1 tag
29        <button onClick={triggerUploadInBackgroundScript} type="button">
30          Upload this page
31          <LuHardDriveUpload id="upload-icon" />
32        </button>
33          [...] // Buttons to seed page and settings page
34        {(ipfsLink && !isError) && (
35          <a href={ipfsLink} target="_blank" rel="noreferrer">
36            {cid}
37          </a>
38        )}
39        {isError && <p>{error}</p>}
40      </div>
41    ) }
```

Listing 3: Code snippet of the requests sent from the pop-up script.

Explanation:

- **Line 1**: Defines the **Home** functional component, which takes an object as its parameter. The component has the property **setPath**, which is a function that accepts a string argument and returns void.
- **Line 2**: Defines a function that changes the path using the **setPath** function passed down as a property.
- **Line 3 - 6**: Declares state variables using React's useState hook:
  - isError: Tracks error occurrence.
  - error: Stores error messages.
  - cid: Stores content identifiers (cid).
  - ipfsLink: Stores the link for IPFS uploading.
- **Line 8 - 9**: The **triggerUploadInBackgroundScript** function sends a message to the background script of the Chrome extension. This message contains a property **message** set to "upload". The second argument is a callback function **res** that handles the response received from the background script.
- **Line 10**: Extract properties from response object **res** and assign to variables *_cid*, *websiteUrl* and *error*.
- **Line 11**: Sets the error state variable. It checks if the error is an object, and if so, it sets the error message, otherwise, it sets the error directly.
- **Line 12**: Sets the **isError** state variable to true if an error exists (!!error converts the truthiness of error to a boolean value), otherwise, it sets it to false.
- **Line 13 - 17**: This conditional block checks if there is no error and _cid" exists and creates an object "cid" with properties.
- **Line 18 - 19**: Sets the cid state variable to "_cid" and pushes *cid* object into an array named *cids* within *store* object.
- **Line 20 - 22**: Sets the *ipfsLink* state variable by concatenating the *store.gatewayUrl* with *_cid* or else sets it to an empty string.
- **Line 27 - 41**: The HTML content is rendered.
- **Line 29**: Defines a button with an onClick event, set to the **triggerUploadInBackgroundScript** function. When clicked, this button initiates the upload process by sending a message to the background script.
- **Line 34 - 39**: Checks if *ipfsLink* exists and there is no error, an anchor is rendered with an IPFS link, or else an error message is rendered

The background script has a function with a listener within a Chrome extension, which responds to the message "upload" from the **triggerUploadInBackgroundScript** function. Whereafter an asynchronous function, **backupAndUploadToIpfs** is executed. The function will be elaborated in the next section.

7.4.1 Retrieve DOM

In order to comprehensively archive web content, our solution requires the preservation of not only the HTML file but also the accompanying images, links and CSS stylesheets. This fulfills requirements 6 and 7. These links will be stored in an array **dowloadQueue**, whereafter they will be processed into a function which will download the links, thereby fulfilling requirement 3. First of all, The background script needs to retrieve the DOM from the currently active tab, by communicating with the content script. The following code is the **backupAndUpload-ToIpfs** function from background/index.ts, which initiates the backup and upload process of a webpage's HTML content to the IPFS network.

```TypeScript
const backupAndUploadToIpfs = async () => {
  let error = ""
  let cid = ""
  let websiteUrl = ""
  let isError = false
  chrome.tabs.query({ active: true, currentWindow: true }, (tabs) => {
    chrome.tabs.sendMessage(
      tabs[0].id,
      { message: "fetch-dom" },
      async (response) => {
        try {
          downloadQueue.length = 0
          const dom = response.dom
          const origin = response.host
          websiteUrl = response.url.replace(/^https?:\/\//, "")

          await addExternalLinksToQueueAndRefactorVirtualDom(dom, origin)
          await downloadEntireQueue()

          const size = downloadQueue.reduce((acc, curr) => acc + curr.blob?.size,

          if (size > 1024 * 500) {
            [...] // File size too large
          } else {
            cid = await uploadToIpfs(dom)
          }
        } catch (e) {
          [...] // error
        } } ) })

  while (cid === "" && !isError) {
    await sleep(200)
  }
  return { cid, error, websiteUrl }
}
```

Listing 4: Code snippet of the background script function backupAndUploadToIPFS which fetches the website and adds the links to be downloaded to a queue

Explanation:

- **Line 1**: defines function **backupAndUploadToIpfs**
- **Line 2 - 5**: initialize variables
- **Line 6**: utilizes the Chrome extension API to query for active tabs in the current window. The parameters ensure that the only active tab in the current window is queried. The method accepts a callback function **tabs**
- **Line 7 - 9**: Send a message to the active tab to fetch the DOM
- **Line 10**: asynchronous function that takes the object response as a parameter.
- **Line 11 - 15**: Try block, where the download queue is reset, and the *dom*, *origin* and *websiteUrl* are extracted from the DOM.
- **Line 17**: await **addExternalLinksToQueueAndRefactorVirtualDom** function, which prepares the download queue by traversing the virtual DOM and extracting external links. The function will be elaborated in the next section.
- **Line 18**: await **downloadEntireQueue** function will download links from the global array **dowloadQueue**.
- **Line 20 - 27**: Check if the file size exceeds the limit else Upload DOM to IPFS and get content identifier (CID). The catch block will handle errors.
- **Line 31 - 34**: Wait until *cid* is populated or an error occurs and return results.

7.4.2 Download links from html

The process of traversing the DOM to extract the links is encapsulated in the function **addExternalLinksToQueueAndRefactorVirtualDom**.

```TypeScript
1  const addExternalLinksToQueueAndRefactorVirtualDom = async (
2    domString: string,
3    origin: string
4  ) => {
5    const dom = makeVirtualDOM(domString)
6
7    const promises = dom.children.map((child) => {
8      return traverseVirtualDOM(child, origin)
9    })
10
11   await Promise.all(promises)
12 }
```

Listing 5: Code snippet of the background script function addExternalLinksToQueueAndRefactorVirtualDom which does as the name follows

Explanation:

- **Line 1 - 3**: Defines async **addExternalLinksToQueueAndRefactorVirtualDom** function with parameters *domString* and *origin*.
- **Line 5**: since the DOM is retrieved as a string from the content script, the DOM will be made into an object by the function **makeVirtualDOM**

- **Line 7 - 8**: maps over the children of the parsed document and creates an array of promises by calling **traverseVirtualDOM** function for each child
- **Line 11**: Awaits for all promises in the array, this ensures all the promises are resolved before proceeding further.

7.5 Packaging of Web Content

The implementation will incorporate an algorithm to navigate through the Document Object Model (DOM). Given that the DOM exhibits a hierarchical tree-like structure, leveraging recursion presents a natural choice for optimization. The recursive algorithm harmonizes with the inherent tree structure of the DOM, offering an efficient and intuitive approach to traversal that optimizes both performance and code clarity. The function TraverseVirtualDom asynchronously traverses a virtual DOM tree and identifies specific attributes within the tag of nodes.

```
1                                                                    TypeScript
2  const traverseVirtualDOM = async (node: ParsedChildNode, origin = "") => {
3    if (node.type === "tag") {
4      const attributeToReplace =
5        node.name === "img" ? "src" : node.name === "link" ? "href" : null;
6
7      if (attributeToReplace) {
8        for (let index = 0; index < node.attributes.length; index++) {
9          const attribute = node.attributes[index];
10         if (attribute.name === attributeToReplace) {
11           let url = attribute.value;
12
13           [...]// Check wether or not its a valid URL using the URL constructor
14
15           const fileType = url.split(".").pop();
16           const hashedUrl = await sha256HashString(url);
17           const newFileName = hashedUrl; //`${hashedUrl}.${fileType}`
18
19           downloadQueue.push({
20             url,
21             newFileName,
22             originalUrl: attribute.value,
23           });
24         } }
25       }
26   }
27   if (node.children) {
28     const promises = node.children.map((child) =>
29       traverseVirtualDOM(child, origin),
30     );
31     await Promise.all(promises);
32   } };
```

Listing 6: Code snippet of the background script function traverseVirtualDOM which finds all links and replaces them with Hashed paths

Explanation:

- **line 2**: Defines the function **TraverseVirtualDom**
- **Line 3 - 4**: Defines the constant attributeToReplace. This checks if the node type is a tag and checks whether the attribute is an image. If so it sets the attributeToReplace equal to src. If the attribute is not an image, it checks if the attribute is a link and if so it sets the attributeToReplace to href. Otherwise, there is no attribute to replace and the value is set to null.
- **Line 6 - 12**: Checks if there is an attribute to replace. The for loop iterates over the number of attributes for a node, and checks if the attribute needs to be replaced.
- **Line 13**: Checks if the URL of the attribute represents a valid URL. If not valid a new URL is constructed by appending the origin onto the relative path.
- **Line 15**: Extracts the file type from the URL by splitting it at the "." and taking the last part.
- **16 - 17**: After processing the URL, the hash value of the URL is calculated based on the hashing algorithm SHA-256. The folders each of the URL will be removed. Therefore, the hashed value serves as a safeguard against files having the same file names but different paths associated with them. This hash value is utilized to create a new path name for the corresponding attribute value, facilitating efficient replacement when all the files get downloaded.
- **Line 21 - 22**: Finally, it pushes an object into a **downloadQueue** array containing information about the URL, the new file name, and the original URL.
- **Line 27 - 30**: Check if current node have children, and then recursively calls the **traverVirtualDOM**.
- **Line 31**: Promise waits for the traversal of the whole DOM

7.6 Upload Web Content

Centralized servers can limit content accessibility and face outages. IPFS, a decentralized storage solution, tackles this by replicating data across a network. This offers resilience, censorship resistance, data integrity, and efficient distribution.

The function **uploadToIpfs** implements the functionality to upload web content, specifically an HTML file and its associated resources to the IPFS network. This fulfills our third *Must have* requirement for ensuring content distribution and accessibility across a distributed network, independent of any centralized server.

```typescript
const uploadToIpfs = async (dom: string) => {
  try {
    [...] Logs the contents of the downloadQueue array.

    dom = replaceLinksInDom(dom)
    dom = removeScriptTags(dom)

    const folderName = await sha256HashString(dom)

    const form = new FormData()

    for (const file of downloadQueue) {
      if (!file.blob) continue
      form.append(
        `${folderName}/${file.newFileName}`,
        file.blob,
        `${folderName}/${file.newFileName}`
      )
    }

    form.append(
      `${folderName}/index.html`,
      new Blob([dom], { type: "text/html" }),
      `${folderName}/index.html`
    )

    const fileRes = await fetch("http://127.0.0.1:5001/api/v0/add?pin=false", {
      method: "POST",
      body: form
    })

    const data = await fileRes.text()
    const fixedData = fixJsonFormat(data)
    const parsedData: ResponseData[] = JSON.parse(fixedData)

    return parsedData[parsedData.length - 1].Hash
  } catch (e) {
    [...] print out a error message and the error
  }
}

```

Listing 7: Code snippet of the background script function uploadToIpfs which takes all the files and uploads them to IPFS

Explanation:

- **Line 1**: Defines the asynchronous function uploadToIpfs, which takes a string dom representing the contents of a DOM (Document Object Model).
- **Line 5-6**: Modifies the DOM by replacing links and removing script tags using helper functions **replaceLinksInDom** and **removeScriptTags** respectively.
- **Line 8**: Generates a hash value (folderName) for the modified DOM using the SHA-256 hashing algorithm.
- **Line 10**: Creates a new instance of **FormData**, which is used to construct a set of key/value pairs representing form fields and their values.
- **Line 12-24**: Iterates through each item in the **downloadQueue** array and appends the corresponding files to the **FormData** object. Each file is appended with a unique key derived from the folderName and the file's new filename. Additionally, the modified DOM itself (index.html) is appended to the form data.
- **Line 27-29**: Sends a POST request to an **IPFS** node running locally with the constructed form data. Upon receiving a response, it parses the JSON data and logs the IPFS links for each uploaded file.
- **Line 32 - 34**:Process the response from the IPFS node and uses an helper function **fixJsonFormat()** to fix potential JSON formatting issues.
- **Line 36**: Returns the IPFS hash of the uploaded files, which is the hash of the last file uploaded.
- **Line 37-38**: Catches any errors that occur during the upload process, logs a failure message, and prints the error details.

7.7 Settings

The pivotal functionality of the Chrome extension is its communication between the different scripts, and updating the states of the components. The feature on the settings page demonstrates the seamless communication and synchronization of state management between different scripts. The Chrome extension has a feature that allows user to choose their own IPFS gateway, which was described in section 6.4. When the user adjusts the IPFS gateway setting, the application's store undergoes an update to reflect this change. Subsequently, this update store information is utilized across various scripts within the application architecture.

```
1  export const Settings = ({ setPath }: { setPath: (string) => void }) => {
2    const snap = useSnapshot(store);
3    const [saved, setSaved] = useState(false);
4    const [timer, setTimer] = useState<number | null>(null);
5
6    [...] // back to home page
7
8    const changeGatewayUrl = (e) => {
9      store.gatewayUrl = (e.target.value);
10     setSaved(false);
11       [...] // timer to save in input field
12   };
13
14   useEffect(() => {
15     return () => {
16       if (timer) {
17         clearTimeout(timer);
18       }
19     };
20   }, [timer]);
21   return (
22       [...] // rendered HTML
23   );
24 };
25
```

Listing 8: Code snippet of the settings page found in settings.tsx

Explanation:

- **Line 1**: React component, takes prop *setPath* which is function for navigation
- **Line 2**: Initialize snapshot of the store, using the *useSnapshot* hook imported from Valtio. This allows the component to access the state stored in *store*.
- **Line 3 - 4**: Initilize two state using *useState* hook imported from React. The state variables are:
  - saved: boolean to keep track of changes that have been saved
  - timer: manage a timer for displaying a "Saved" message after a delay
- **Line 7**: function helps to navigate back to the homepage

- **Line 8 - 10**: defines function **changeGatewayURL** that updates *gatewayUrl* property in the *store* object, when input value changes. It sets *saved* state to false, which is used to manage a timer
- **Line 11**: Conditional statement to check if the timer is active
- **Line 14 - 20**: *useEffect* hook sets up to clean up function, which is responsible for clearing the timer, whenever a component is removed from the UI, and its DOM elements are deleted from the document or the *timer* state changes. It checks if the *timer* is currently active, if so the **clearTimeout** function is called to clear the timer.
- **Line 21 - 22**: render a form with an input field for editing the gateway URL and display a "Saved" message

8 Testing

To ensure that the proposed solution is viable and works as intended, tests had to be done.

The group have used various techniques to ensure that the solution works as intended such as unit tests. To ensure that uploading IPFS data works correctly a testing software named "Insomnia" is used.

8.1 Unit tests

Testing utility function used throughout the code base is a vital step in insuring program integrity. Normally one would do unit tests for near all functions in a code base. For testing 3 different utility functions where chosen for testing to make sure their output is as suggested. The code was also written using TypeScript leading to type-safety throughout the program.

fixJsonFormat

This function is supposed to fix a broken JSON format the code recieves as a response from IPFS.

```
{
          "Name":   "1fcfb3eb5ce21ff321af7d30e2a44a95ff249ca4762f5fcee87c8d9fd37354ba/
ed915bf9134ffc344a2eb1bba99768bf63f04ed1ec3e9a7b5c51114825ac5a90",
    "Hash": "QmZq7uRL9rGhpzbagLo45ufDw3RsPSmR2AEW5WvtnAfhYY",
    "Size": "2426"
}{
    "Name": "1fcfb3eb5ce21ff321af7d30e2a44a95ff249ca4762f5fcee87c8d9fd37354ba/71d72662bb4170ef930a4
    "Hash": "QmZmGE6q4HoS1yj67d4QN9gFRDaygcJxsQmX3yhyR4GnV8",
    "Size": "1695"
}
```

Listing 9: Example of broken JSON format

In Listing 9 an example of the broken JSON format received from IPFS is presented. The fault of this JSON format is that it does not start and end with square brackets [], furthermore there is no comma (**,**) between the curly brackets {}.

To test **fixJsonFormat** it was put through different cases of broken JSON strings. The result was then checked for JSON compatibility with different test cases listed further below in Table 4

```
[
  {
           "Name":  "1fcfb3eb5ce21ff321af7d30e2a44a95ff249ca4762f5fcee87c8d9fd37354ba/
ed915bf9134ffc344a2eb1bba99768bf63f04ed1ec3e9a7b5c51114825ac5a90",
      "Hash": "QmZq7uRL9rGhpzbagLo45ufDw3RsPSmR2AEW5WvtnAfhYY",
      "Size": "2426"
  },{
      "Name": "1fcfb3eb5ce21ff321af7d30e2a44a95ff249ca4762f5fcee87c8d9fd37354ba/71d72662bb4170ef930
      "Hash": "QmZmGE6q4HoS1yj67d4QN9gFRDaygcJxsQmX3yhyR4GnV8",
      "Size": "1695"
  }
]
```

Listing 10: Example of correct JSON format

To test the function different unit tests where made.

Test name	Passed
Multiple elements file	Yes
Single element file	Yes
Check if Length is preserved	Yes
Correct elements returned	Yes
Can handle empty strings	Yes

Table 4: Unit tests for **fixJsonFormat**

In Table 4 the different tests for **fxJsonFormat** are displayed and the result. To conclude **fixJsonFormat** is successfully able to format the respond from IPFS into correct JSON format.

removeScriptTags Adding reactive elements to the copied website was out of scoped so the program needed an easy way to remove all reactive and script components. For this **removeScriptTags** was created. Testing this function includes sending a mock HTML file/ string through the function and check if any scripts reside.

The function succesfully passes the test that was made for it, but sometimes it results in an error. This error is due to whitespace between element tags not matching between the result and the expected result. These errors will be overlooked since whitespace between element tags does nothing in HTML.

sha256HashString

Testing this function was done alongside using a online webtool, which could also hash string into sha256. Running the same string through the online tool and comparing that to the output of the function should lead to an equal output.

The function was tested through multiple strings of different lengths and compared to the webtools results. These tests all concluded with the hash being an exact match.

8.2 Insomnia

Insomnia is a software that can be used for testing, debugging, and mocking API's. [81]

In this project, Insomnia was used to test the interaction between IPFS and the extension. ´

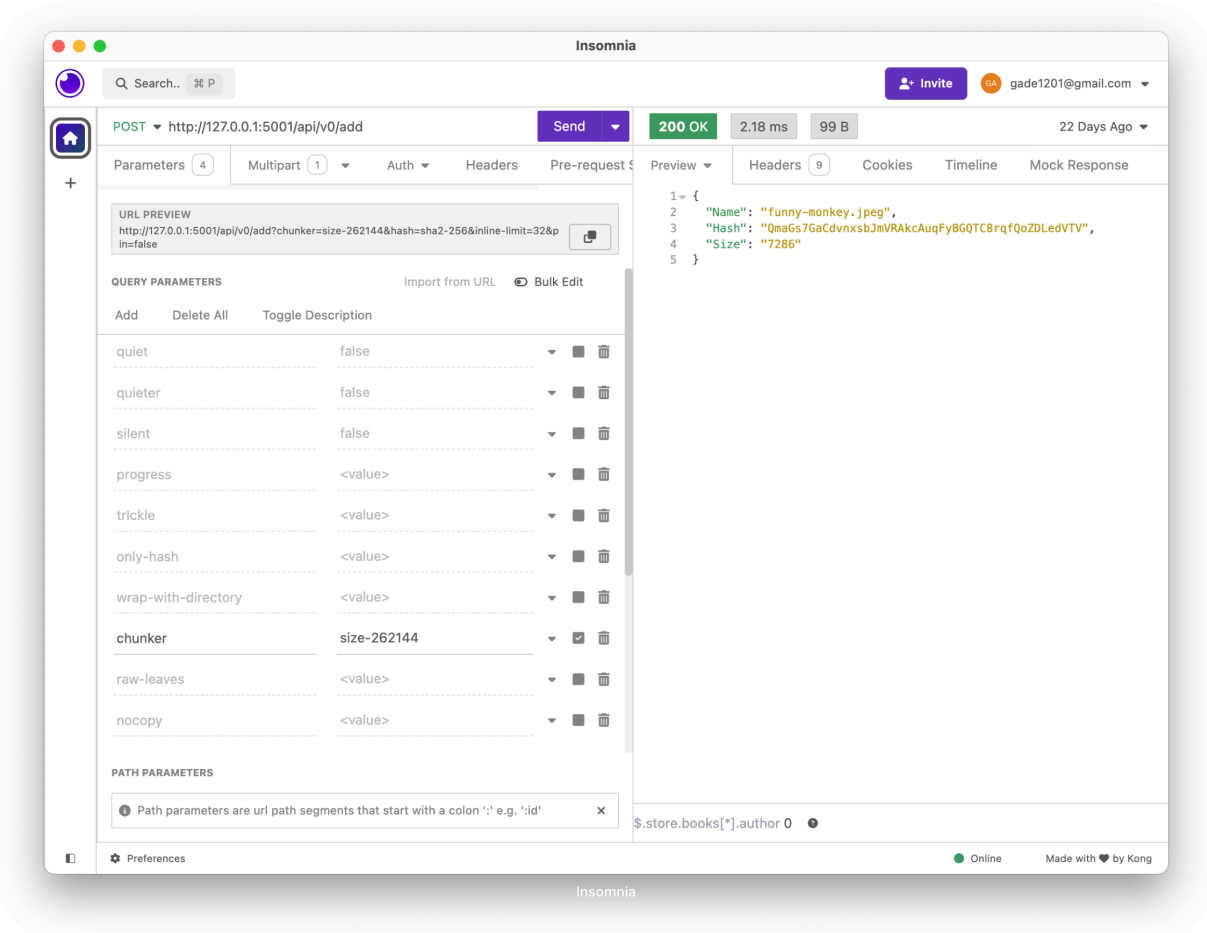

Figure 14: Insomnia IPFS testing

In Figure 14, a screenshot of Insomnia can be seen. In the screenshot, a sample picture was uploaded to IPFS using Insomnia. After the image was uploaded, IPFS gave a response with a hash. The hash could then later be looked up and confirmed to be uploaded correctly.

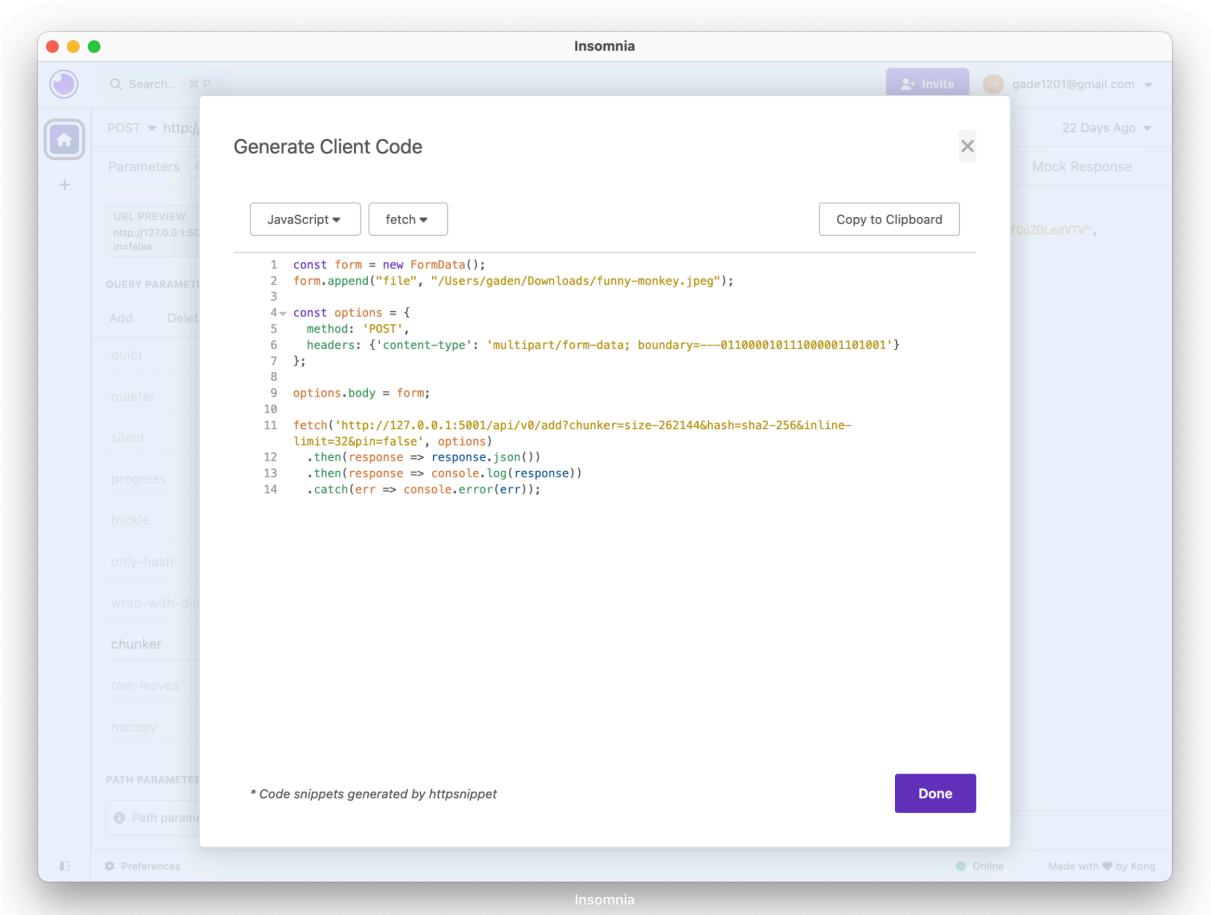

Figure 15: Insomnia code generation

After the upload was confirmed to be uploaded correctly, Insomnia was used to generate the sample code. In picture Figure 15, the generated code can be seen.

This way of development allows for testing the code before implementing it and gives certainty that the API is implemented correctly and working as intended.

8.3 Manual testing with developer tools

After the API calls were tested and implemented, the solution itself had to be tested. Multiple techniques were used for that.

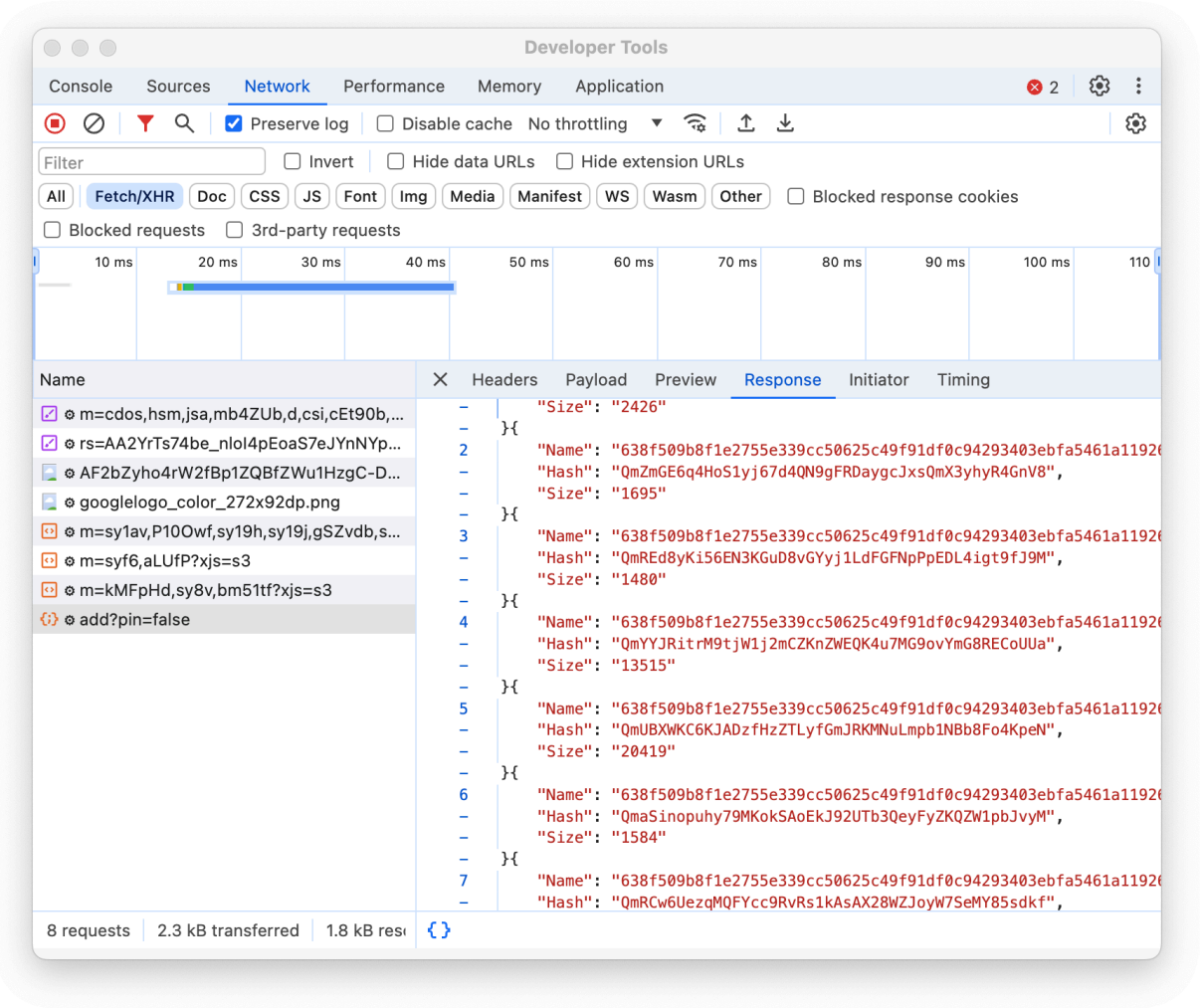

Figure 16: Chrome Developer Tools

In image Figure 16 the solution can be seen tested using chrome developer tools. After clicking the “backup” button on any arbitrary website, the network requests within the extension are inspected. The extension can be seen downloading all the resources on the site, followed by a single upload request to IPFS.

The extension shows the ability to download and upload most webpages. However, during testing, some sites were found to be too large to be uploaded to IPFS in a single request.

This proved the testing useful and showed improvements to be implemented in the future.

9 Discussion

The proposed solution fulfills a lot of the requirements listed in the requirements section. However, there are downsides to the solution.

9.1 Peers

In a decentralized architecture, no single entity is responsible for storing the data, which poses a new problem: if all peers stop seeding, the backup will become unavailable. One potential solution is to have an organization seed all the links created through our extension, although this might turn the system into a more centralized one. However, clients who wish to back up a link forever have the option to take responsibility for seeding themselves. This hybrid approach allows for both decentralized and centralized elements, potentially balancing reliability with user autonomy.

9.2 Helia vs Kubo

The initial idea was to have a Chrome extension with all the functionality built-in. For the IPFS uploading process, the group decided to use Helia since it can run in a web environment. Unfortunately, Helia was throwing many errors that could not be solved within the project's scope. Instead, the group settled on using Kubo. Kubo does not run in a development environment and runs as an executable on the user's machine. The Chrome extension then connects using an HTTP API to Kubo.

While this solution works, it requires users to undertake a cumbersome setup process, which is not ideal. Simplifying this setup would have been a significant improvement if more time had been available.

9.3 Assessing the requirements

To determine whether or not the solution has fulfilled the goals of the project, the requirements from section 4 will be assessed.

Identifier	MoSCoW	Requirement	Status
1	Must have	A web-based front end	Fulfilled
2	Must have	Download the HTML of a webpage to main memory	Fulfilled
3	Must have	Upload the webpage to a decentralized network	Partially Fulfilled
4	Should have	Allow other user to share uploaded webpages	Fulfilled
5	Should have	Share the HTML file as a static webpage on a decentralized network	Fulfilled
6	Should have	Include images from the webpage in the uploaded content	Fulfilled
7	Should have	Include CSS from the website in the uploaded content	Fulfilled
8	Should have	Make it possible to stop sharing specific uploaded content	Not Fulfilled
9	Could have	Include JavaScript in the uploaded content.	Not Fulfilled
10	Could have	Include a progress bar for downloading content	Not Fulfilled
11	Could have	Upload a digital representation of the webpage	Not Fulfilled
12	Will not have	Allow an organization to easily share all uploaded webpages	Not Fulfilled
13	Will not have	Website interface with list of all previously generated links	Not Fulfilled
14	Will not have	Automatically upload and create links for visited webpages	Not Fulfilled
15	Will not have	Compression of webpage content	Not Fulfilled
16	Will not have	Upload all webpage links referenced on the uploaded webpage	Not Fulfilled

Table 5: Reassessing MoSCoW requirements. Green means implemented, yellow means partially implemented, red means not implemented.

Must have
All three “Must have” requirements for the solution have been fulfilled. The extension has a web-based front end in the form of a Chrome extension, thereby fulfilling requirement 1. In addition, by using the chrome extension it is possible to download the HTML of a webpage to main memory (requirement 2), and upload the webpage to a decentralized network (requirement 3).

Requirement 3 is only partially fulfilled since not all websites can be uploaded to IPFS due to the max file size limit on the protocol.

Should have
Most “Should have” requirements have been fulfilled. The extension can successfully download a webpage along with its styling and images and share it using a decentralized network (requirements 4, 5, 6, and 7).

Could have
None of the proposed features in the category have been fulfilled.

Will not have
As the category implies, none of the features in this category have been implemented although as will be mentioned in the future works section implementing some of them might make for a better product.

Although the most important requirements were fulfilled, the implementation can still be improved upon. The requirement with the highest need for improvements is the backup and upload algorithm. It works on some sites, but sites that go beyond the block limit of IPFS do not work. This makes it a high priority for future work.

9.4 Future Work

The final product fulfilled many of the requirements mentioned in the MoSCoW requirements but not all. Additionally, throughout the project, multiple possible improvements were discovered but did not get implemented. Some of these improvements are listed down below. These improvements vary wildly in what they improve on, being either UI, UX, or simply code optimization.

One of the biggest thresholds of the solution is that currently only a file size of 500 kB or less can be uploaded. This is due to the IPFS system having a block maximum capacity of 500 kB. This is not optimal since many webpages take up more space than 500 kB and can therefore not be uploaded as a complete webpage. In this case, only the HTML file is uploaded and the CSS and images are discarded. To improve upon this would be splitting the webpage sources into different CID's and referencing them through an IPFS gateway. This would be a huge update to the solution since many webpages have a file size of more than 500 kB. One limiting factor of this solution would be the webpage losing its sources/media if the gateway ever closed. Implementing this would also require a larger refactorization of the codebase but it is definitely a great improvement compared to the current solution.

Another improvement for the current solution is to implement Helia correctly to allow the extension to work independently. The state of the solution is currently that you can not pin your upload without downloading IPFS desktop resulting in your uploads to be deleted from IPFS. This makes for the extension to be rather impractical since you have to download and configure IPFS desktop with the correct settings to be able to pin your upload. Implementing Helia will therefore be a great improvement since it removes the need to download and set up IPFS desktop.

Lastly, having a centralized server seeding all content uploaded through the extension would help ensure that links would not die as long as the centralized solution is in operation. This would be an addition to the current system and would therefore act as a buffer for users' own seeding of their content. This would be particularly helpful in the extension's early phases where it has a small user base.

10 Conclusion

The primary focus of this project, as identified in Chapter 2, stems from the inherent challenges of preserving digital content. Link-rot is a phenomenon, where links stop working and disrupt the flow of information searching as well as undermining the credibility of digital content. This chapter led to the formulation of the following problem statement:

> "*How can we develop a resilient webpage preservation system to mitigate the risks of link-rot, ensuring the sustained accessibility and integrity of the linked content?*"

Based on the problem statement MoSCoW requirements were set up and based on these and further investigations into the potential technologies available, it was decided that the solution to the problem statement would be a Chrome extension. Implementing the solution as a Chrome extension enables seamless preservation of web content directly from the user's browser. By allowing users to upload and manage their digital content independently of centralized servers, the application promotes a more resilient digital ecosystem.

The solution was put through several tests that confirmed its ability to reliably store and retrieve web content across decentralized nodes. However, the tests also identified potential challenges, such as dependency on network activity and the need for sustained community support to maintain data availability in the long term.

As the project progresses, future developments may include enhancing the user interface and extending browser compatibility. These improvements will ensure that the solution addresses more than just the technical aspects of link-rot but also aligns with users' needs and practical usability.

In conclusion, the problem statement have been addressed, by tackling the critical challenges of preserving digital information through the development of a Chrome extension that leverages a decentralized storage solution (IPFS). This approach combats link-rot, a major threat to long-term web accessibility. By empowering users to independently archive web content, the extension promotes a more resilient digital ecosystem and ensures the continued availability of valuable information for future generations.

Bibliography

[1] K. Król and D. Zdonek, "Peculiarity of the bit rot and link rot phenomena," [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/GKMC-06-2019-0067/full/html

[2] M. Castells, "The Impact of the Internet on Society: A Global Perspective," [Online]. Available: https://www.bbvaopenmind.com/en/articles/the-impact-of-the-internet-on-society-a-global-perspective/

[3] Britannica, "Internet," [Online]. Available: https://www.britannica.com/technology/Internet

[4] Mozzilla.org, "How does the internet work," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/How_does_the_Internet_work

[5] Mozzilla.org, "An overview of HTTP," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[6] Mozzilla.org, "HTTPS," [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/HTTPS

[7] Britannica, "World Wide Web," [Online]. Available: https://www.britannica.com/topic/World-Wide-Web

[8] Mozzilla.org, "What are Hyperlinks," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_are_hyperlinks

[9] Cloudflare, "What is a domain name? | Domain name vs. URL," [Online]. Available: https://www.cloudflare.com/learning/dns/glossary/what-is-a-domain-name/

[10] Mozzilla.org, "What is a Domain Name," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_domain_name

[11] Cloudflare.com, "What is DNS? | How DNS works," [Online]. Available: https://www.cloudflare.com/learning/dns/what-is-dns/

[12] P. Stox, "At Least 66.5% of Links to Sites in the Last 9 Years Are Dead," [Online]. Available: https://ahrefs.com/blog/link-rot-study/

[13] K. Yasar, "Link rot explained: Everything you need to know," [Online]. Available: https://www.techtarget.com/whatis/feature/Link-rot-explained-Everything-you-need-to-know

[14] A. Steele, "What is Link Rot? The Hidden Threat to Your Website's Health," [Online]. Available: https://loganix.com/what-is-link-rot/

[15] "Linkody Link Rot Study & Link Building Statistics," [Online]. Available: https://blog.linkody.com/case-studies/link-rot

[16] A. Liptak, "In Supreme Court Opinions, Web Links to Nowhere," [Online]. Available: https://www.nytimes.com/2013/09/24/us/politics/in-supreme-court-opinions-clicks-that-lead-nowhere.html

[17] J. Zittrain, K. Albert, and L. Lessig, "Perma: Scoping and addressing the problem of link and reference rot in legal citations," *Legal Information Management*, vol. 14, no. 2, pp. 88–99, 2014.

[18] J. M. Perkel, "The trouble with reference rot." [Online]. Available: https://www.nature.com/articles/521111a

[19] Elsevier, "About Elsevier." [Online]. Available: https://www.elsevier.com/about

[20] Clare, "Link Rot Isn't Unique to the Legal Field," *Perma.cc*, 2019.

[21] K. Zhou, C. Grover, M. Klein, and R. Tobin, "No more 404s: predicting referenced link rot in scholarly articles for pro-active archiving," in *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries*, 2015, pp. 233–236.

[22] Brett, "So how bad a problem is link-rot?," 2019, [Online]. Available: https://archive.blogs.harvard.edu/perma/2019/02/14/so-how-bad-a-problem-is-link-rot/

[23] J. P. B. A. U. Rehman Rui L. Aguiar, "Fault-Tolerance in the Scope of Cloud Computing," [Online]. Available: https://www.researchgate.net/publication/361237225_Fault-Tolerance_in_the_Scope_of_Cloud_Computing

[24] A. S. T. Maarten van Steen, "A brief introduction to distributed systems," [Online]. Available: https://link.springer.com/article/10.1007/s00607-016-0508-7

[25] S. Soller, "On centralized and decentralized distributed systems," [Online]. Available: http://arkanis.de/projects/2014-01%20On%20centralized%20and%20decentralized%20distributed%20systems/On%20centralized%20and%20decentralized%20distributed%20systems.pdf

[26] T.-W. U. G. M. L. Nguyen Binh Truong Upul Jayasinghe, "A Survey on Trust Computation in the Internet of Things," [Online]. Available: https://www.researchgate.net/publication/316042146_A_Survey_on_Trust_Computation_in_the_Internet_of_Things

[27] W. Machine, "About the Internet Archive," [Online]. Available: https://archive.org/about/

[28] Archive.today, "Faq - Archive.today," [Online]. Available: https://archive.ph/faq

[29] Perma.cc, "Perma.cc user guide," [Online]. Available: https://perma.cc/docs

[30] Stilio, "Stilio website," [Online]. Available: https://www.stillio.com/

[31] Archivebox, "Archivebox website," [Online]. Available: https://archivebox.io/

[32] Cyotek, "Cyotek website," [Online]. Available: https://www.cyotek.com/cyotek-webcopy

[33] Pocket, "Save to Pocket," [Online]. Available: https://chromewebstore.google.com/detail/save-to-pocket/niloccemoadcdkdjlinkgdfekeahmflj

[34] W. M. by Internet Archive, "Wayback Machine by Internet Archive," [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/wayback-machine_new/

[35] Webcite, "Faq - Webcite," [Online]. Available: https://webcitation.org/

[36] J. of Medical Internet Research, "JMIR publications," [Online]. Available: https://support.jmir.org/hc/en-us/articles/115002044888-What-is-WebCite

[37] Time.com, "Internet Archive Loses Lawsuit Over E-Book Copyright Infringement. Here's What to Know," [Online]. Available: https://time.com/6266147/internet-archive-copyright-infringement-books-lawsuit/

[38] Archive.org, “Impacts of the temporary National Emergency Library and controlled digital lending,” [Online]. Available: https://blog.archive.org/2020/06/11/impacts-of-the-temporary-national-emergency-library/

[39] Google, “Managing owners, users, and permissions - Google,” [Online]. Available: https://support.google.com/webmasters/answer/7687615?hl=en

[40] Stilio, “Stilio - Faq,” [Online]. Available: https://www.stillio.com/faq

[41] Perma.cc, “Perma.cc,” [Online]. Available: https://chromewebstore.google.com/detail/permacc/bigjakhahgnccheaompmgebkncglllel

[42] Perma.cc, “Perma.cc Contingency Plan,” [Online]. Available: https://perma.cc/contingency-plan

[43] Perma.cc, “Link rot explained: Everything you need to know,” [Online]. Available: https://https//perma.cc/terms-of-service#tos-7

[44] D. Peter Judge, “Fire destroys OVHCloud's SBG2 data center in Strasbourg,” [Online]. Available: https://www.datacenterdynamics.com/en/news/fire-destroys-ovhclouds-sbg2-data-center-strasbourg/

[45] OVH, “OVH - Faq,” [Online]. Available: https://www.ovhcloud.com/en/about-us/

[46] S. E. j. Roger Montti, “OVH Data Center Fire Darkens Popular Sites Worldwide,” [Online]. Available: https://www.searchenginejournal.com/ovh-data-center-fire-darkens-thousands-of-sites-worldwide/398485/?ref=hackernoon.com

[47] D. Peter Judge, “51 join class action against OVHcloud over fire data loss,” [Online]. Available: https://www.datacenterdynamics.com/en/news/51-join-class-action-against-ovhcloud-over-fire-data-loss/

[48] Heimdal, “Danish Cloud Hosting Companies Ravaged by Ransomware Attacks,” [Online]. Available: https://heimdalsecurity.com/blog/danish-companies-ransomware-attacks/

[49] C. Glover, “Devastating ransomware attack hits Danish cloud hosting companies CloudNordic and AzeroCloud,” [Online]. Available: https://techmonitor.ai/technology/cybersecurity/ransomware-attack-on-cloudnordic-azerocloud-loses-all-data

[50] Oxford, “Definition of the Word Webpage,” [Online]. Available: https://www.oed.com/search/dictionary/?scope=Entries&q=WebPage

[51] Oxford, “Definition of the Word Website,” [Online]. Available: https://www.oed.com/search/dictionary/?scope=Entries&q=WebSite

[52] Oxford, “Definition of the Word URL,” [Online]. Available: https://www.oed.com/search/dictionary/?scope=Entries&q=URL

[53] ProductPlan, “MoSCoW Prioritization.”

[54] W3Schools, “TypeScript Introduction,” [Online]. Available: https://www.w3schools.com/typescript/typescript_intro.php

[55] N. Biswas, “TypeScript Basics,” [Online]. Available: https://link-springer-com.zorac.aub.aau.dk/book/10.1007/978-1-4842-9523-6

[56] O. Adepoju, “State Management In React With Valtio,” [Online]. Available: https://blog.openreplay.com/state-management-in-react-with-valtio/

[57] Simplilearn, "React vs Javascript: Key Differences [2024]," [Online]. Available: https://www.simplilearn.com/react-vs-javascript-article

[58] D. Mallick, "Virtual DOM vs Browser DOM in React.js," [Online]. Available: https://medium.com/@darshana_18428/virtual-dom-vs-browser-dom-in-react-js-eccf466d6e8b

[59] Plasmo, "Introduction to Plasmo." [Online]. Available: https://docs.plasmo.com/

[60] I. docs, "What is IPFS," [Online]. Available: https://docs.ipfs.tech/concepts/what-is-ipfs/#defining-ipfs

[61] I. docs, "Welcome to the IPFS docs," [Online]. Available: https://docs.ipfs.tech/

[62] Cloudflare, "Interplanetary File System (IPFS) · Cloudflare Web3 docs." [Online]. Available: https://developers.cloudflare.com/web3/ipfs-gateway/concepts/ipfs/

[63] I. Docs, "Nodes," [Online]. Available: https://docs.ipfs.tech/concepts/nodes/#types

[64] I. docs, "How IPFS works," [Online]. Available: https://docs.ipfs.tech/concepts/how-ipfs-works/#subsystems-overview

[65] I. docs, "Content Identifiers (CIDs)," [Online]. Available: https://docs.ipfs.tech/concepts/content-addressing/#what-is-a-cid

[66] I. docs, "Distributed Hash Tables (DHTs)," [Online]. Available: https://docs.ipfs.tech/concepts/dht/#kademlia

[67] I. docs, "IPIP-0337: Delegated Content Routing HTTP API," [Online]. Available: https://specs.ipfs.tech/ipips/ipip-0337/

[68] S. R. S. A. Shantanu Kumar Rahut Razwan Ahmed Tanvir, "Scientific Paper Peer-Reviewing System With Blockchain, IPFS, and Smart Contract." [Online]. Available: https://www.researchgate.net/publication/333328471_Scientific_Paper_Peer-Reviewing_System_With_Blockchain_IPFS_and_Smart_Contract

[69] I. Docs, "The lifecycle of data in IPFS," [Online]. Available: https://docs.ipfs.tech/concepts/lifecycle/#_1-content-addressable-representation

[70] I. docs, "Bitswap," [Online]. Available: https://docs.ipfs.tech/concepts/bitswap/

[71] I. docs, "HTTP Gateways," [Online]. Available: https://specs.ipfs.tech/http-gateways/

[72] I. Docs, "Persistence, permanence, and pinning," [Online]. Available: https://docs.ipfs.tech/concepts/persistence/#persistence-versus-permanence

[73] L. Tapasweni Pathak, "Guide to IPFS garbage collection," [Online]. Available: https://blog.logrocket.com/guide-ipfs-garbage-collection/

[74] dremio, "What are Binary Large Objects?," [Online]. Available: https://www.dremio.com/wiki/blobs/

[75] React, "Using the Effect Hook," [Online]. Available: https://legacy.reactjs.org/docs/hooks-effect.html

[76] React, "Using the state Hook," [Online]. Available: https://legacy.reactjs.org/docs/hooks-state.html

[77] W. schools, "React useEffect Hooks," [Online]. Available: https://www.w3schools.com/react/react_useeffect.asp

[78] C. Innocent, “Understanding React’s useEffect cleanup function,” [Online]. Available: https://blog.logrocket.com/understanding-react-useeffect-cleanup-function/

[79] M. Chamin Jayasooriya, “How to Use Valtio: A Simple and Powerful State Management Library for React (Part 1),” [Online]. Available: https://medium.com/@chamin.njay/how-to-use-valtio-a-simple-and-powerful-state-management-library-for-react-part-1-61f20f53820d

[80] M. Usetech, “ State management in React using Valtio,” [Online]. Available: https://medium.com/@usetech/state-management-in-react-using-valtio-8609fbcf0ad7

[81] Kong, “The Collaborative API Development Platform - Insomnia.” [Online]. Available: https://insomnia.rest/