Integration of the Web Monitoring software into Internet Archive's Wayback Machine

The development of wayback-diff and wayback-discover-diff

By

Fotios Tsalampounis

Supervisors: S. Ougiaroglou, D. Dervos

Department of Information and Electronic Engineering
International Hellenic University

A dissertation submitted to the International Hellenic University in accordance with the requirements of the degree of Bachelor of Informatics Engineering.

September 2019

Word count: 20877

Ενσωμάτωση λογισμικού Web Monitoring στο Wayback Machine του Internet Archive

Η υλοποίηση του wayback-diff και του wayback-discover-diff

του

Φώτιου Τσαλαμπούνη

ΑΜ: 133996

Επιβλέποντες Σ. Ουγιάρογλου, Δ. Δέρβος

ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων
Διεθνές Πανεπιστήμιο της Ελλάδος

Η πτυχιακή εργασία κατατέθηκε στο Διεθνές Πανεπιστήμιο της Ελλάδος σύμφωνα με τις απαιτήσεις για το πτυχίο του Μηχανικού Πληροφορικής ΤΕ.

Σεπτέμβριος 2019

Λέξεις: 20877

Abstract

This Thesis presents the work the author did during the Google Summer of Code 2018 for the Internet Archive. To be precise, the development of a ReactJS frontend application called Wayback-diff and a Python backend application called Wayback-discover-diff is discussed. Wayback-diff is an application that allows users to compare two snapshots of the Wayback Machine side-by-side. Wayback-discover-diff calculates the simhash values of snapshots of webpages offering information that acts as a measure to how much a webpage has changed over time. Information about the tools, frameworks and libraries that were used for their development as well as an algorithmic explanation of the applications' code can be found in chapters two and three. Chapter three also contains an analysis of the Wayback-discover-diff's runtime, the actions that were taken to improve it and the statistical analysis of those improvements. Finally, chapter four provides an insight as to how the applications were integrated into the Wayback Machine and chapter five discusses future improvements.

Αυτή η πτυχιακή παρουσιάζει την δουλειά του συγγραφέα στα πλαίσια του Google Summer of Code 2018. Πιο συγκεκριμένα παρουσιάζεται η ανάπτυξη της frontend εφαρμογής wayback-diff σε ReactJS και της Python backend εφαρμογής Wayback-discover-diff. Το wayback-diff είναι μια εφαρμογή που επιτρέπει στους χρήστες να συγκρίνουν δύο snapshots μιας ιστοσελίδας δίπλα-δίπλα. Το Wayback-discover-diff υπολογίζει την Simhash τιμή των snapshot των ιστοσελίδων παρέχοντας ουσιαστικά μια νέα πληροφορία που μπορεί να χρησιμοποιηθεί ως μέσο μέτρησης των αλλαγών στην ιστοσελίδα με τον πάροδο του χρόνου. Πληροφορίες σχετικά με τα εργαλεία, τα frameworks και τις βιβλιοθήκες που χρησιμοποιήθηκαν στην ανάπτυξη των εφαρμογών, όπως και η αλγοριθμική εξήγηση των εφαρμογών, μπορούν να βρεθούν στα κεφάλαια δύο και τρία. Το κεφάλαιο τρία επίσης περιέχει την ανάλυση του χρόνου εκτέλεσης του wayback-discover-diff, τις βελτιώσεις που έγιναν σύμφωνα με αυτή και την στατιστική ανάλυση αυτών των βελτιώσεων. Τέλος, το κεφάλαιο τέσσερα περιέχει μια περιγραφή σχετικά με τον τρόπο ενσωμάτωσης των εφαρμογών στο Wayback Machine και στο κεφάλαιο πέντε γίνονται προτάσεις για μελλοντικές βελτιώσεις.

Dedication and acknowledgements

I would first like to thank my Google Summer of Code 2018 mentor, Vangelis Banos for offering abundant support throughout and after the GSoC. I am very grateful for his willingness to help me learn. He consistently steered me towards the right direction and ignited my interest towards radical solutions but at the same time did not interfere, proving his interest to help me grow as a developer instead of just getting the work done.

I would also like to express my gratitude towards Google and the Google Open Source program. They help organizations like the Internet Archive carry out important work while at the same time promoting the Open Source Software culture and offering young developers the chance to cultivate important skills and kick-start their careers.

Lastly but most importantly, I would like to express my profound gratitude towards my mother, my brother, my sister and my close friends. With their kind words they have been supporting me throughout my studies, giving me confidence and fortitude to keep going. This accomplishment would not have been possible without them. It stands as a testament to their unconditional love and encouragement. Thank you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter the main concepts and driving forces that played a profound role in the devision of this Thesis are described. The author's fervor towards Open Source software, the principles behind it and, its significant role to shaping the modern software's development patterns are just some of the reasons that inspired him.

Also the importance of Open source software and its licenses are explained among with a definition of what the Google Summer of Code, the Internet Archive and EDGI are. Last but not least, the scope of this Thesis is explored.

1.1 Open Source software

Open source means literally that the source code is open. It is open for everyone to see, learn from, contribute to and, in some cases, copy and modify. Open source also means that the compiled source code is free for everyone to use. It is widely accepted that information needs to be easily accessible to everyone. Technology's rapid moving forward and information's becoming more widely available every moment is giving software development an explosive growth. An important constituent to achieve the same rate of accessibility is to make the right tools available.

The open source model is a decentralized software development model that relies on peer production, with the source code and documentation as well as, sometimes, the blueprints freely available to everyone. [56] Open-source projects are meant to be a collaborative effort, where developers among the community improve upon the project's source code and share those improvements with the rest of the community. Even though the need for the open-source movement was born as a rejoinder to the limitations of proprietary code, open-source code is again released under the terms of an open-source software license. Of course, the terms of such licence are different as to those of a proprietary project.

1.1.1 Open source licenses

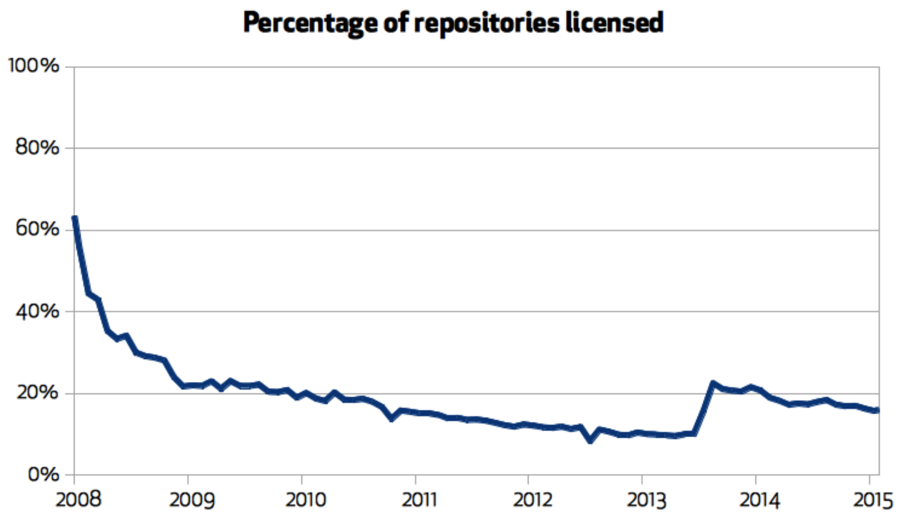
Open source licenses comply with the Open Source Definition, protect the community and the involved developers while sharing knowledge and the code. In brief, they allow software to be freely used, modified, and shared. A variety of licenses exist to cover different needs. For a new license to be approved by the Open Source Initiative (also known as the OSI), it must go through a license review process.

OSI has approved 83 open source licenses to date.[55] However about 80% of the projects available on Github have no license.

> Open source simply isn't open source without a proper license. Unless you've explicitly told others that they can modify and reuse your work, you've only showed others your code; you haven't shared it. [54]

To be more precise, when it comes to creative work, including code, in the absence of a license the fail-safe default is that the work is under exclusive copyright. [27]

Figure 1.1: Percentage of repositories licensed on GitHub [54]



This is what most developers do not know and the reason why including an open source license in a project is vital.

Looking into all of the open source licenses is beyond the scope of this Thesis. For the sake of completion, the top 10 open source licenses used in projects on GitHub are shown below.

Rank	License	% of projects
1	MIT	44.69%
2	Other	15.68%
3	GPLv2	12.96%
4	Apache	11.19%
5	GPLv3	8.88%
6	BSD 3-clause	4.53%
7	Unlicense	1.87%
8	BSD 2-clause	1.70%
9	LGPLv3	1.30%
10	AGPLv3	1.05%

Table 1.1: Open source license usage on GitHub.com March 9, 2015 [54]

1.1.2 Google Open Source

Google has made sure to actively participate in the open source software scene with various ways over the last couple of decades. A lot of their projects, and their code, are released under an open source licence. Whether it is because they think it might help push the industry forward, share best practices they developed, or even because they think some code is interesting to share with the community, they go ahead and release it. The number of these projects today adds up to almost seventy. Another way Google is contributing to the Open Source community is through developer engagement programs such as Google Code-in and the Google Summer of Code.

1.1.2.1 Google Code-in

Pre-university students, aged from 13 to 17 years old, are given the chance to participate in Google Code-in.

Google Code-in has been endeavouring, for eight consecutive years, to engage young students into open source software development. Since the program begun, almost 8000 students from 107 countries have participated in the contest. [13] Choosing from a wide variety of tasks, students can work on coding, documentation, training, outreach, research, quality assurance, and design making this an excellent opportunity to interact with a very wide spectrum of open source.

Every year Google chooses more than a dozen Open Source organizations to participate. The organizations submit a list of tasks that students should work on for three to five hours. Each organization has dedicated mentors that guide students through these tasks. Mentors provide feedback on the students' submitted work and ask for changes if they find it is necessary. To make the program more enticing, Google offers prizes for the students including digital certificates, t-shirts, and to the finalists a trip to Google's headquarters in California.

1.1.2.2 Google Summer of Code

For university students at the age of eighteen or older, Google runs the Google Summer of Code program. Having the same main objective as Google Code-in, Google Summer of Code offers a well-rounded coding experience. Since its debut in 2005, almost 15.000 students, over 24.000 mentors and 118 countries worldwide have produced over 35 million lines of code for 651 open source organizations. [14]

In order to participate, organizations submit an *Organization application*. If they get accepted, they submit their *Ideas List*, a document in which organizations describe in detail the projects they would like to collaborate with students during the summer on. This in turn means that, instead of working on distinct tasks from an organization's list, student developers can choose and apply to up to three projects from the same or different organizations' *Idea List*.

The goal of the program is to provide students with real life coding experience while monitoring and guiding their progress. Each student is assigned a mentor from their participating organization. The mentor helps them stick to their schedule, solves any questions they might have and last but not least inspires them to contribute to open source development even after the summer ends.

The program spans over the course of 16 weeks. Starting from May it is separated into four periods. [18] The first period, called "**Community Bonding**", added in the program in 2007 [17], is a three week era in which the students have plenty of time to familiarize themselves with the community, set up their development environment, read the required documentation and work out the details of their project implementation with their mentor. After the Community Bonding period at the end of May, the coding begins.

The coding is split into three month long periods. During those thirty days students are required to spend at least forty hours per week working on their project and communicating with their mentor. The project's guidelines make it clear that the GSoC should be considered to be a full-time commitment. Over the last four days of each month, students and mentors are required to submit their evaluation. For students, this should be an evaluation of their mentor and their organization. They will be asked about the frequency and the quality of their communication, how much time they spent working on the project, to point out any difficulties they faced and any suggestions for their mentor and organization or the program in general. Mentors, on the other hand, evaluate the students' work during the same period of time. Mentors must make a pass or fail decision for the student based on their work. If the mentor passes the student, the student will be receive a part of their stipend and will be able to continue with the program. In the unfortunate case that the mentor has to submit a failing evaluation of the student, the student does not receive any kind of stipend and is immediately removed from the GSoC.

Stipend

As mentioned above, students are paid a stipend for their work during the Google Summer of Code. The stipend is not to be considered a payment, as Google does not employ the students but, aims to cover their expenses during the program. The amount of this stipend is dependent of the country each student studies in. If the student is enrolled in an online university, their home location is used instead.

To determent the amount of the stipend, Google uses the Purchasing Power Parity (PPP) theory. In this way, using 6000 USD as the base, 2400 USD as the minimum and 6600 USD as the maximum amount, Google makes sure to cover every student's living expenses fairly.

Organizations

The entry requirements for organizations are much simpler. In order for an organization to be eligible to participate in the GSoC it should; 1) be a group running an active free/open source software project. The project does not need to be a legally incorporated entity. 2) have already produced and released software under an Open Source Initiative approved license. 3) have at least two contributors to serve as organization administrators and/or mentors for the entire program.

After submitting their project ideas and the minimum and maximum number of student they wish to receive, they wait for the Google Open Source Program Office to reach a decision.

1.1.3 Internet Archive

The Internet Archive is a Non Profit organization founded in 1996 in San Francisco, aiming to provide "Universal access to knowledge". One way it aims to achieve that is by creating a free and open library on the web. Among other initiatives, all aiming towards creating a free and open library, it is archiving webpages in various moments throughout time with the Wayback Machine. Users are able to search for webpages in the Wayback Machine and see how they looked in a specific, already archived, moment in time. They can also request for a webpage to be immediately archived. This enables the users to visit this created archive in the future. Apart from webpages, the Internet Archive's digital library also contains archives of sound files, video files, training material and other cultural artifacts.

The need behind archiving the Web becomes apparent when one compares webpages to newspapers. Both are very powerful tools which carry the role of providing information to the general public. That information could concern important matters, like governmental and local administration issues but, also aim to entertain the public with information about cultural and heritage events. In addition, through newspapers, journalists have a long history of uncovering interesting stories and making sure to hold the appropriate person responsible when they uncover a wrongdoing.

Retaining an archive of published newspapers, has two main advantages. First and foremost, it creates ripe conditions for journalistic accountability. Published articles remain available for everyone to read, criticize and use as a reference. This encourages journalists to be more responsible in their publications and as a result leads to higher quality and credibility content. Moreover, these archives constitute an excellent source of cultural information. Researchers use these data to study the social and cultural aspects and phenomena of a region during a specific period of time in the past. Newspapers publish what people would want to read, so it is an excellent source to investigate what was the public interested in.

The web, because of its nature, makes information easily accessible to a much broader audience than a newspaper. It also easily provides a platform for everyone to speak their mind and publish content. With the ease of access to the Web that is available today, virtually anyone can go online and and post information about anything. That information can be spread around the world in a few seconds. Also, the technology behind the web allows the webpage publishers to edit their content anytime they think it is appropriate without leaving a trace. Keeping archives of webpages help mitigate the danger of not being able to trace malicious changes to information and also, offers a great opportunity to gather data about how the Web changes over time.

Since 1996, the Internet Archive has archived:

> 330 billion web pages 20 million books and texts 4.5 million audio recordings (including 180,000 live concerts) 4 million videos (including 1.6 million Television News programs) 3 million images 200,000 software programs

The Wayback Machine

The Wayback Machine is the Internet Archive's project for archiving the web. Today it contains more than 2 petabytes of compressed data, translating to over 345 billion web captures, including content from every top-level domain, over 200 million web sites, which are in over 40 languages.

1.1.4 The EDGI

The Environmental Data & Governance Initiative, or EDGI for short is an international network of academics and non-profits addressing potential threats to federal environmental and energy policy, and to the scientific research infrastructure built to investigate, inform, and enforce them.

Website Monitoring is an EDGI project aspiring to build tools and community around monitoring changes to government websites, both environment-related and otherwise.

Project Goals

The purpose of the system is to enable analysts to quickly review monitored government websites in order to report on meaningful changes. The Website Monitoring automated system a.k.a. Scanner aims to make these changes easy to track, review, and report on. [11]

1.2 Scope of Thesis

This Thesis will describe in detail the work the author has done for the Internet Archive as a student developer for the Google Summer of Code 2018. All of the worked he carried out during the GSoC is released under an open source licence.

His project was **Idea 5 - Integrate the "Scanner" software into the Wayback Machine**. The aim of this project was to identify changes in archives for a given URL and highlight those changes by integrating the web-monitoring software into the Wayback Machine and help to further advance it. This further translates into improving the web-monitoring software, recognising which components are useful to the Internet Archive's implementation, how they can be best used and extracting and using them in a new project.

The web-monitoring project consists of four components:

1. **web-monitoring-db** A Ruby on Rails app that serves database data via a REST API, serves diffs and collects human-entered annotations.

2. **web-monitoring-ui** A React front-end that provides useful views of the diffs. It communicates with the Rails app via JSON.

3. **web-monitoring-processing** A Python backend that ingests new captured HTML, computes diffs, performs prioritization/filtering, and populates databases for the Rails app.

4. **web-monitoring-versionista-scraper** A set of Node.js scripts used to extract data from Versionista and load it into the database.

After carefully studying these components it was apparent that only component number 3 and part of component number 2 were required for our implementation. Part of the second component was used one of the applications developed for this project called wayback-diff. Wayback-diff is further explained later on in this Thesis.

The web-monitoring-processing component was used without the need of any major modifications. For the purposes of this project and in the of collaboration between the Internet Archive and the EDGI some new features were added. Since web-monitoring-processing is not in the scope of this Thesis, the new features and improvements made to it are going to be briefly mentioned for completion purposes.

1.3 Structure of this Thesis

This thesis is structured in a manner that would allow even people with a shallow background in software development understand how this project was carried out and how the software works.
For this purpose, Chapter 2 contains an extensive enumeration and explanation of the technologies used in wayback-diff and wayback-discover–diff.
Following Chapter 2, Chapter 3 acts as an explanation handbook for the two applications providing an insight as to how they were developed, their algorithmic and structural importance and how wayback-discover-diff was analyzed and improved.
Chapter 4 showcases how the applications were integrated into the Wayback Machine, which was the main goal of the project. Screenshots from the Wayback Machine are used to support this showcase.

Lastly, in Chapter 5 this Thesis ends with a few words about the project and work that can be done on future improvements.

Chapter 2

Technologies used

In this chapter it is described how during the authors participation in the Google Summer of Code 2018, two applications for the Internet Archive's project were developed. In more detail the technologies and methodologies used throughout the project are explained. Last but not least definitions of the terms used in the project are given in this chapter.

2.1 wayback-diff

Wayback-Diff is the React component which I created for the Google Summer of Code 2018 and will be integrated into the Wayback Machine. It queries the wm-diffing-server (part of the web-monitoring-processing app) for the differences between two captures of a webpage. wm-diffing-server in turn fetches the two captures from the Wayback Machine, calculates their differences and returns a JSON response to the component. After that, wayback-diff renders the snapshots with their differences marked in the user's browser. This component is exported as an ES module and can be used in any React project.

2.1.1 React

ReactJS is an open-source JavaScript library which is used for building user interfaces specifically for single page websites or applications. Single page web applications translate into applications that change their views and data without reloading the webpage. React is used for handling the view layer for web applications and mobile apps. It also allows developers to create reusable UI components.

One of the main objectives of React's creating was to create a fast, scalable and simple framework. It only handles the user interface and corresponds to the MVC (Model–view–controller) template. It can cooperate with many JavaScript libraries and frameworks. The most important features of React are:

- JSX: a simple JavaScript template which allows HTML quoting and uses the HTML tag syntax to render React components.
- Single-Way data flow: when rendering child components, a set of immutable values, the props, are passed to them as properties in its HTML tags. The children components can not modify this properties but they can call upon a callback function that would handle such modifications. This is known as “properties flow down; actions flow up”.
- Virtual Document Object Model: instead of updating directly the Document Object Model (DOM), React offers the opportunity to manipulate and update a virtual representation of the browser’s DOM in the memory. Then, dependent on the changes made, the framework calculates the updates that need to be made to the DOM.
- React Native: React has native libraries that enable the React framework architecture to be used in order to develop native mobile (iOS and Android) applications.

React was created by Jordan Walke, a software engineer working for Facebook. React was first deployed on Facebook’s newsfeed in 2011 and on Instagram.com in 2012. Facebook maintains React to date.

2.1.2 JSX

JSX is an XML/HTML-like syntax used by React that extends ECMAScript so that XML/HTML-like text can co-exist with JavaScript/React code. This means that JSX provides developers with an easier way to develop code which allows them to pull HTML to JavaScript code instead of the other way around.

Many developers find JSX exceedingly easy to use. It is also easier to read and be understood even by novice developers. JSX is not part of React and does not attempt to comply with any HTML or XML specifications. It is designed as an ECMAScript feature and the similarities with HTML and XML are superficial. The advantages of the JSX summed up in four points are:

1. Less technical people can understand and modify the code. CSS developers and designers will find JSX more familiar than JavaScript alone.
2. One can leverage the full power of JavaScript in HTML and avoid learning or using a templating language. JSX is not a templating solution, instead is a declarative syntax used to express a tree structure of UI components.
3. Adding a JSX transformation step will unearth errors in your HTML code that might otherwise be missed.

4. JSX promotes the idea of inline styles.

As mentioned above, to transform the JSX code to JavaScript Babel is required. Babel automatically processes the JavaScript files during development using Babel CLI tool.

2.1.3 Yarn

No matter which language is used, it is often that pieces of code or features of an application are reused in the same or different projects. A solution to easy code reuse would be one that allows the storage of code blocks in a manner that its reuse would be effortless and allows developers to focus on the rest of their projects and testing the code.

To solve this issue, package managers were created. Basic components, libraries, or frameworks are stored in a way that every developer can access or upload new ones. The primary function of a package manager is to install packages from a global registry to a developer's machine. Many programs use more than one package, leaning, in some cases, to thousands of packages which have tree-like dependencies. These dependencies are versioned and installed based on semantic versioning (semver).

Semver defines a versioning scheme that reflects the types of changes in each new version, whether a change breaks an API, adds a new feature, or fixes a bug. However, semver relies on package developers not making mistakes. When the dependencies are not clear, errors, bugs and inconsistencies appear. These inconsistencies will be passed on if the code is not checked and the developer is not aware of them.

One popular package manager for JavaScript is npm. The npm client accesses many online repositories to view, retrieve and install packages that a developer might find useful. It also has a registry can be used to access and search, view, download projects or upload their own.

The npm registry was found to be inadequate across Facebook's needs. To overcome its shortcomings they decided to create a new package manager, yarn. Yarn is the result of the collaboration of Facebook with Exponent, Google, and Tilde. Yarn still accesses the packages on the npm repository but achieves faster installation and manages dependencies consistently across remote machines or machines in secure offline environments.

Yarn replaces the existing workflow of the npm client or other package managers while retaining compatibility with the npm registry. It is faster, more secure and more reliable than other package managers with the same feature set. It resolves the issues around versioning and non-determinism by using lockfiles and an install algorithm that is deterministic and reliable. When a dependency is made, each lockfile locks the dependencies on a specific version so as to avoid the npm client's corresponding way of handling the dependencies, which was the node.

Node creates a dependency tree but its structure may differ from machine to machine. This happens because node has the tendency to merge the same nodes together. So, while an application might run on the developer's machine without any issues, it might not run on other machines. The reason for this is that when the code is distributed, the dependency tree has the final form where some dependencies might be merged.

Lockfiles on the other hand lock the dependency nodes on the version used to create the program and travel with it across every machine. They use a concise format with ordered keys to ensure that changes are minimal and review is simple. The install process of dependencies in yarn can be broken down into three steps:

1. Resolution: Yarn starts resolving dependencies by making requests to the registry and looking up each dependency.

2. Fetching: It looks in a global cache directory to see if the package needed has already been downloaded. If it hasn't, Yarn fetches the tarball for the package and places it in the global cache so it can work offline and won't need to download this dependency if it is used in the future. Dependencies can also be placed in source control as tarballs for full offline installs.

3. Linking: Finally, yarn links everything together by copying all the files needed from the global cache into the local node_modules directory.

Throughout this entire process, Yarn imposes strict guarantees around package installation. You have control over which lifecycle scripts are executed for which packages. Package checksums are also stored in the lockfile to ensure that you get the same package every single time.

2.1.4 Callback functions

In JavaScript, functions are objects. Because of this, functions can take functions as arguments, and can be returned by other functions. Functions that do this are called higher-order functions. Any function that is passed as an argument is called a callback function. Functions, that don't take a function as an argument or return a function as output are called first-order functions. Handling functions just like any other bits of data in programs, makes abstraction a lot easier.

Callback functions are easy to make. They are easy to work with, their code does not repeat itself, it is more abstract, easier to maintain, easily readable and can have more specialized functions. They can be passed as an argument to another function, exactly as a variable would be. The function is passed without the trailing pair of executing parenthesis () like when we are executing a function. The reason for this is because adding the trailing parentheses would cause the callback function to be executed immediately. Instead it is "called back" (hence the name) at some specified point inside the containing function's body.

The higher-order function works as a closure to the callback function that calls it. This means that, when the higher-order function calls the callback function, the code of the callback function starts running and whatever it was written after the call of the callback function will not be run until the callback finishes running. Though the callback function basically works as a normal function, there are some noteworthy principals for their implementation:

- Named or Anonymous functions can be used as Callbacks
- Parameters can be passed to Callback functions
- The Callback should be made sure it is a function before executing

In the case of asynchronous code execution, many callback functions can make the code complicated and chaotic. This phenomenon is called "Callback hell" because it makes the code difficult to completely comprehend. There are two solutions to "callback hell":

1. Naming the functions: Declaring them and passing just the name of the function as the callback, instead of defining an anonymous function in the parameter of the main function.
2. Modularity: Separate the code into modules, so it is possible to export a section of code that does a particular job. Then import that module into the larger application.

2.1.5 React Router

It is often found that a web application can be composed of applications implemented both on the frontend and on the backend. These applications can aim to service different needs but when used together offer a new added-value service. This could mean that they do not necessarily have the same routing conventions. This is the case where the backend service functions just as an API, and the user does not interact with it at all. The routes that used to manage the user experience and the routes that used to manage queries to the database are not the same.

An API is any place where a piece of code talks to another piece of code. It is often used to refer to an external resource that gives values, or the code's internal database resource(s).

React Router is an Application Programming Interface (API) for React applications. React Router uses dynamic routing and it allows the application, dependent on the URL, to display new information without reloading the webpage. React Router uses component structure to call components, which display the appropriate information. The use of a router in React prevents the flash of white screens when navigating through the application, during the page loading time, making the user experience smoother. In a sequence of parent-child programming, when the Router is not used, the highest parent component is a React component. Otherwise, Router needs to be the highest parent so it can render different components and pass props to their children and thus the entire application.

2.1.6 Components

Components are the building blocks of any React application. Simply put, a component is a JavaScript class or function that optionally accepts input i.e. properties (props) and returns a React DOM element that describes how a section of the User Interface should appear. There are two types of components in React:

- Functional components

- Class components

Functional components are purely presentational and are represented by a function that optionally takes props and returns a React element to be rendered in the DOM tree. Generally, it is preferred to use functional components whenever possible because of their predictability and conciseness. Since, they are purely presentational, their output is always the same given the same props. Functional components are sometimes referred to as stateless, dumb or presentational in other literature. All these names are derived from the simple nature that functional components take on.

- Functional because they are basically functions
- Stateless because they do not hold and/or manage state
- Presentational because all they do is output UI elements

Class components on the other hand are created using ES6 class syntax. They have some additional features such as the ability to contain logic (for example methods that handle onClick events), local state, and other capabilities. From other resources, class components are referred to as smart, container or stateful components.

- Class because they contain classes
- Smart because they can contain logic
- Stateful because they can hold, manage and depend on local state
- Container because they usually contain other components

In order to choose the component that most suits ones needs, they should determine the circumstances that each of the components are used for. The use of a class component is advised if there is a need to:

- Manage local state
- Add lifecycle methods to the component
- Add logic for event handlers

Otherwise, one should always use a functional component.

2.1.7 Props

One of React's benefits is the implementation and use of reusable components. Depending on the story case and data of the application, the components are usually rendered dynamically in order to maximize their usefulness. Upon creation, components can be customized using parameters. The parameters passed to a component when creating it are called properties (props) of the component.

In functional components, components can pass properties and data down from one component to another, typically in a parent-child formation. Props are read-only variables and cannot and should not be modified by any component.

In class components the idea of props is the same as in functional components, but there are two notable distinctions:

1. Props are not passed as an argument to the class
2. The name attribute is accessed using this.props.name instead of props.name

Finally, there is one last way of adding props to a component. Default props are used by a component as default attributes in case no props are explicitly passed to that component. As a fallback, default props are helpful in enabling the developer to offer a better user experience throughout the application.

2.1.8 Validating Proptypes

As an application grows, many bugs might found during typechecking. JavaScript has some tools to typecheck projects, like Flow or TypeScript. React also has some built-in typechecking utilities. To enable typechecking on the props for a component, the special propTypes property can be used. PropTypes not only offers vital help towards catching bugs, but also serves as handy documentation on how a component csn be used properly in terms of expected props.

PropTypes define the type of each prop. Each time, a value is passed through a prop, it gets validated against its type. If the type of a prop value is different data type than what is specified in PropTypes, an error message will be printed in the browser console. To be more specific, PropTypes help:

- Draw attention to bugs caused by using the wrong data type in a prop (e.g.: object instead of a string)
- Show all the available props including their required data, in one place.
- In some code editors that support code completion for props, the available props are visible while typing in the component in a tooltip.

Overall, PropTypes is a great way to validate the component properties and it also provides a quick overview of all the available props and the expected data types. Thus, the use of PropTypes is considered to be a best practice in respect to avoiding some unpalatable bugs and improving the reusability of the component.

2.1.9 Even components can be passed as props

Just like object oriented programming supports the composition of classes, React enables the composition of components. These composed components receive their attribute components as props. This concept of component composition is quite powerful as it enables you to write highly modular and reusable components.

This type of architectural flexibility allows a few important features that greatly add upon React's adaptability. To be more precise, it allows the up-front initialization of a component and its props before the application even runs and the dynamic initialization of a component that will be rendered in a different screen of the application's case story or inside a child component that would not have the correct context and data to correctly render it otherwise.

2.1.10 React State

The "state" is what allows developers to create components that are dynamic and interactive. "State" is an object that components rely on to determine how they will be rendered and behave. React's "state" is one of the more complex concepts to apprehend. Not just as to what should be saved in state, but what it actually is and how React works with it.

State is monitored throughout the application and it is used to calculate whether a component should be re-rendered. Each React component is bound to the state variables it uses to be rendered (state variables passed as props) and the state variables used to control its appearance (state variables in control statements).

A change in the data saved in state might be caused, upon a series of different events. Namely, on a button fire event, if data is received from a server, either from a websocket message, or from a response to a previous request or a timer event. This list of course is not exhaustive. In all those cases, the state is updated through code. As the state object is immutable and should not for any reason be updated directly, React's API offers the "this.setState" function for this purpose. When the state is updated, React compares the old state with the new one to determine which components need to be updated. It does not re-render all the visible components if there is no need to do so. In addition, though the "shouldComponentUpdate" function components can implement a specific update behaviour on props and state updates.

2.1.11 Components Lifecycle

React web applications are actually a collection of independent components which run according to the interactions made with them. Every React Component has a lifecycle of its own. The lifecycle of a component can be defined as the series of functions that are invoked in different stages of the component's existence. The functions containing the "Will" naming convention take place before some specific phase and "Did" after the completion of that phase. A React Component can go through four stages of its life as follows:

1. Initialization: This is the stage where the component is constructed with the given props and default state. This is done in the constructor of a Component Class. In this phase the developer has to define the props and initial state of the component this is generally done in the constructor of the component.

2. Mounting: is the stage of rendering the DOM elements returned by the render method. It is when the component has finished initializing and is being mounted on the DOM and rendered for the first time in the webpage. At this stage the lifecycle functions called, can be easily be understood because of their Naming Conventions. Hence, it becomes clear that the mounting phase consists of two phases, implemented on their respective functions as described below:

   - UNSAFE_componentWillMount(): this function is invoked right before the component is mounted on the DOM. It gets invoked once before the render() function is executed for the first time. Thus, calling this.setState() would not trigger a new render. This function was used to initialize the state of the component but has been marked as legacy and should not be used in new projects. Instead any state initialization should happen in the constructor.
   - componentDidMount(): this function is invoked right after the component is mounted on the DOM. This function gets invoked once after the render() function is executed for the first time.Initialization that requires DOM nodes should go here as well as any network requests.

3. Updating: is the stage when the state or props of a component are updated and it needs to be re-rendered. The following are the descriptions of functions that are invoked at different points of this phase:

   - setState(): Even though this function is a React component lifecycle function, It wouldn't be accurate to call a part of the update phase of a component. It is what triggers the update phase.
   - shouldComponentUpdate(): By default, every state or props update of a component would re-render the page. This may not always be the desired outcome. This function lets React know whether the component should be re-rendered depending on the specific update of the props and date. shouldComponentUpdate() is invoked before rendering an already mounted component when new props or state are being received. It receives the new props and state as arguments and returns the appropriate value indicating whether the component should be re-rendered or not. If it returns false, the subsequent steps of rendering will not be executed. This function won't run if the forceUpdate() function has triggered the re-render of the component.
   - UNSAFE_componentWillUpdate(): As the name suggests, this function is invoked before the component is re-rendered i.e. this function gets invoked once before the render() function is executed after the update of state or props and is where any preparation of the

component's data should take place. This function is deprecated and replaced by componentDidUpdate().

- componentDidUpdate(): Similarly this function is invoked after the component is re-rendered. This function gets invoked once after the render() function is executed. This function offers the opportunity to operate on the DOM after the component has been updated. It is also where the developer should make network requests if they depend on the change that has happened to the props.
- UNSAFE_componentWillReceiveProps(): This function is invoked before a mounted component gets its props updated. The new props are passed as parameters and logic can be implemented that would update the state accordingly. This function was deprecated because it was easy to create cases that would lead to inconsistencies and bugs.

4. Unmounting: As the name suggests unmounting is the final step of the component lifecycle where the component is removed from the page. The following function is the sole member of this phase:

    - componentWillUnmount(): This function is invoked before the component is finally unmounted from the DOM. This offers the opportunity to developers to perform any clean-up operations such as invalidating timed events, canceling network requests and destroying any objects.

2.1.12 Pure Components

The pure component is one of the most significant ways to optimize React applications. The usage of Pure Component gives a considerable increase in performance because it reduces the number of render operations in the application. A Pure component can replace a component which has only the render function implemented and rely on the default implementations of the rest of the lifecycle functions for the rest of its functionality. Instead of making a full-blown component just to render some content to the screen, the pure component can substitute it. Pure components are the simplest, fastest components one can write. They are easy to write, simple to reason about, and the quickest to implement.

Pure components do not have to be simple though. They can also incarnate the implementation of an, otherwise, very resource hungry component. Using a Pure component instead of a simple component in this scenario would be preferred because relying on the developer's implementation of the lifecycle functions and not having any boilerplate code overhead from the framework would make complex components run faster. Advantages to using functional components in React are:

- Do away with the heavy lifting of components, no constructor, state, life-cycle madness, etc.
- No this keyword (i.e. no need to bind).

- Presentational components (also known as dumb components) that emphasize UI over business logic are smaller, faster and, simpler.
- Encourages building smaller, self-contained components.
- Highlights badly written code (for better refactoring).
- Their architecture helps them be very fast and easy to reuse

2.1.13 iFrame

An iFrame (Inline Frame) is an HTML document embedded inside another HTML document on a webpage. Although an iFrame behaves like an inline image, it can be configured with its own scrollbar independent of the surrounding page's scrollbar. The iFrame is a very convenient tool if one needs to decouple some content from the main content or load external content.

iFrames can display dynamic content without requiring the user to reload the surrounding page. This capacity is enabled through JavaScript or the target attribute of an HTML anchor. There are several reasons a developer would want to render to an iFrame but the main benefit is style encapsulation. The iFrame HTML element is also often used to insert content from another source, such as multimedia or advertisements, in the web page such as YouTube Videos, Google Maps etc.

2.1.14 Lodash

Lodash is a JavaScript productivity kit. It helps deal with all types of objects and save developers time by not having to code generic functions. The code will also be cleaner with less lines and it will work on all browsers. Lodash's modular methods are great for:

- Iterating arrays, objects and strings
- Manipulating and testing values
- Creating composite functions

Lodash can function as a template engine too. The base of lodash is pulled from underscore.js, yet another powerful library. Lodash uses embedded JavaScript format to render templates.

2.2 Wayback-discover-diff

Wayback-discover-diff in a Python based web server application which uses Flask and Celery to run and, Redis as its database. It's primary task it to calculate the Simhash value of snapshots of webpages. It is developed as an added value service upon the Wayback Machine and wayback-diff. The WBM has a substantial number of webpages archived throughout different points in time. Wayback-diff was created to help users also compare them instead of just being able to see on snapshot at a time and identify their changes. The problem Wayback-discover-diff aims to solve is identifying which snapshots of a webpage makes sense to compare. If the webpage has radically changed from one point in time to another or if it remained the same, comparing two snapshots would not produce any useful information.

In order to overcome this issue, wayback-discover-diff calculates the difference between too snapshots. Thus, it provides an accurate and reliable way to guide the users towards meaningful snapshot comparisons. To do so, wayback-discover-diff is implemented in Python and uses the Simhash algorithm to hash and provide a measurable value to the snapshots.

2.2.1 Python programming language

As mentioned above, the wayback-discover-diff application is written using Python. Python is an interpreted, high level, general-purpose, programming language. It was released in 1991 and created by Cuido van Rossum. It's an object-oriented language whose design philosophy emphasizes code readability. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

Python was created as a successor to the ABC language. Python 2.0, which was released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, which was released in 2008, was a major revision of the language that is not completely backward-compatible, meaning that Python 2.0 code needs to be modified to run on Python 3.0. Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation.

Python is very popular amongst programmers due to its readability, structure and the fact that is an interpreted language. Interpretation allows the programmer to debug a program faster and avoid every segmentation fault that may exist. A source debugger, also written in Python, reads the program line by line, after the interpreter raises an exception, and prints a stack trace, inspects local and global variables, evaluates arbitrary expressions, sets breakpoints and so on. In addition to this, Python has some advantages that make it preferable to other languages, such as:

- Simplicity: Python is a minimalistic language. It is very easy to write and read.
- It's free: Python is free and open source. This means that the developers don't have to pay for anything. They can share, copy and change it.
- Compatibility: Python is compatible with numerous platforms, so developers don't have to worry about supporting different CPU architectures.
- Object-oriented: Python supports procedure-oriented as well as object-oriented programming. In the procedure-oriented programming, a developer applies reusable pieces of code. The object-oriented programming uses objects which are based on data and functionality.

- Libraries: The Python community has created a wide variety of libraries for Python. These libraries allows developers to manage documentation, perform unit testing, query databases, mock web browsers, send emails, provide cryptography functionality, present graphical user interfaces and so on.

2.2.2 Python Virtual Environments

Python, like most of the modern programming languages, has its own way of downloading, storing and resolving packages and modules. While this has its advantages, some problems might occur, especially with how and where the packages are stored.

The main purpose of Python's virtual environments is to create an isolated environment for Python projects to be created and executed in. This means that each project has its own set of dependencies discrete from any other project. As follows, multiple projects can use their own virtual environment using as many resources as needed and still be independent of one another. Therefore, there is no limit to the amount of environments one can create since they essentially are just directories containing scripts.

One important issue that virtual environments aim to solve is breaking package changes. It is not very rare that one of the packages becomes incompatible with the other parts of the application. A viable solution is to set up a new environment for this application, that contains the Python version and the packages versions that are compatible with each other and isolate them from the rest of the operating system. Another issue is the mitigation of the varying parameters that might cause the application to fail to run properly on different systems.

When developers collaborate to develop an application and upon delivering the application to a client it is cardinal that the different configuration of those systems not to hamper the applications ability to run. Virtual environments can grant the convenience of running the application in a sealed environment where it is known to run properly.

A Python environment consists of a certain Python version and some packages. The two most popular package manager tools that assist in managing them are PIP and Conda. PIP, is only a Python package manager used to install and manage software packages. It can be used for Python version 2.7.9 and later. Conda on the other hand, is a package and environment manager. It is a multipurpose, more flexible and easier to use.

From the tools mentioned above, Conda is the manager used for the applications developed during the Google Summer of Code for Internet Archive's project and described in this thesis.

2.2.3 Conda environment manager

Conda is an open source, cross platform, language-agnostic package manager and environment management system for Python. It is released under the Berkeley Software Distribution License by Continuum Analytics. It allows users to easily install different versions of binary software packages and required libraries so that they can meet the requirements of their project policy. Moreover, it allows them to switch between package versions and update the packages from known software repositories.

Although Conda is a program written in Python it has the flexibility needed to manage projects in different languages, like the language R which is a programming language for statistical computing. Conda, in contrast with similar Python-based cross-platform package managers, can also install Python. Conda manager has many versions that can be installed in any operating system and favor different use cases. Namely the following versions of Conda exist:

- Miniconda, a free, open source distribution of the Python and R programming languages. It provides the user with the flexibility to use only the packages needed for their project by downloading them one by one. Because of this, it is faster but suitable for more experienced users.
- Anaconda, is a free, open source distribution of the Python and R programming languages. It comes with some pre-installed packages and thus is friendlier for a new user of Conda or Python.
- Anaconda Enterprise platform, it is a commercial product that allows organizations to apply Python and R in enterprise environments

In order to choose between the version of Conda that would better fit the needs of the project, one needs to consider that Anaconda is a version that is more user friendly, especially for inexperienced programmers. It has almost every implementation they might need, but it is larger. This means that on environments that memory is a limited resource, user friendliness might come at a high cost. Miniconda, on the other hand, is addressed to more experienced users who know exactly the packages needed for each of their projects. So, it offers users the perfect tool to build their environment from scratch and configure it to their exact needs. The extra time spent setting the environment up is paid back in speed and in a lower memory footprint.

2.2.4 Flask Framework

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require any particular tools or libraries to work. It offers the creator of the application full creative control. They can choose the components of the application or even write their own and Flask will still be able to run them without any issues.

Flask is designed to be extended. It has a robust core that includes basic functionality that all web applications need and expects everything else to be implemented by the developer or third-party extensions. Such extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools and are updated constantly.

The Flask framework was created by Armin Ronacher of Pocoo, an international group of Python enthusiasts who conceived the idea as an April fool's joke that grew popular and was made into an application. Despite the lack of major release, Flask is very popular among Python enthusiasts, especially after 2016 when it became the most popular development framework on GitHub.

The fact that Flask Framework is a microframework and its not dependent on any particular tools or libraries has its positives, as mentioned above, but also has some negative features. One that stands out is that the developer will have to do more work to develop or increase the list of dependencies by adding third party extensions to be able to use Flask in a modern web application.

A prime indicator that Flask core has a strictly need-to-operate codebase, is its dependencies. Flask core has only two dependencies, Werkzeug, a WSGI utility library, and jinja2 which is its template engine.

The Web Server Gateway Interface (WSGI) is a simple calling convention for web servers to forward requests to web applications or frameworks written in Python. The WSGI has two sides, the server/gateway side and the application/framework side. The server side is often a full web server such as Apache or Nginx, or a lightweight application server that can communicate with a web server, such as flup. The the application side is a Python callable, supplied by the Python program or framework.

Between the server and the application, there may be one or more WSGI middleware components, which implement both sides of the API, typically written in Python. WSGI does not specify how the Python interpreter should be started, nor how the application object should be loaded or configured, and different frameworks and web servers achieve this in different ways.

Jinja is a web template engine for the Python programming language and is licensed under the BSD software license. It is a text-based template language and thus can be used to generate any markup code as well as source code. Jinja2 is one of the most used template engines for Python. It is inspired by Django's templating system but extends it with an expressive language that gives template authors a powerful set of tools. On top of that it adds sandboxed execution and optional automatic escaping for applications where security is important. It is internally based on Unicode and runs on a wide range of Python versions from 2.4 to current versions including Python 3. In more detail Jinja's features include:

- Sandboxed execution mode. Every aspect of the template execution is monitored and explicitly whitelisted or blacklisted.
- Powerful automatic HTML escaping system for cross site scripting prevention.
- Template inheritance that makes it possible to use the same or a similar layout for more templates without having to duplicate code.
- High performance with just in time compilation to Python bytecode.
- Optional ahead-of-time compilation.
- Debug system that makes it easy to debug integrating template, compile and runtime errors into the standard Python traceback system.

- Configurable syntax. One can reconfigure Jinja2 to better fit different output formats such as LaTeX or JavaScript.
- Template designer helpers. Jinja2 ships with a wide range of useful little helpers that assist in solving common tasks in templates such as breaking up sequences of items into multiple columns and more.

Overall, even though the Flask framework is not intended for inexperienced users, it provides its users the significant creative freedom to create projects that would architecturally fit their exact needs and not dictate specific restrictions, architectural or otherwise.

2.2.5 Celery Framework

Celery is a Python Framework that, following the Object-Oriented Middleware approach, is used for managing and distributing tasks. Its main feature consists of handling many small tasks and distributing them in a large number of computational nodes using multiprocessing. These tasks are executed concurrently either asynchronously or synchronously.

Celery is written in Python but its task and job queue protocol can be implemented in any programming language. It can also operate with other languages using webhooks to exchange data with the applications. For some languages, like Ruby, PHP, Go, and Node.js, different, dedicated clients exist.

Some features that make the Celery Framework stand out are:

- Monitoring, a stream of monitoring events is emitted by workers and is used by built-in and external tools to tell you what your cluster is doing – in real-time.
- Workflows, simple and complex workflows can be composed using a set of powerful primitives we call the "canvas", including grouping, chaining, chunking, and more.
- Time and Rate Limits, control how many tasks can be executed per second/minute/hour, or how long a task can be allowed to run, and this can be set as a default, for a specific worker or individually for each task type.
- Scheduling, specify the time to run a task in seconds or a datetime, or you can use periodic tasks for recurring events based on a simple interval, or Crontab expressions supporting minute, hour, day of the week, day of the month, and month of the year.
- Resource Leak Protection, the max-tasks-per-child option is used for user tasks leaking resources, like memory or file descriptors, that are simply out of the developer's control.
- User Components, each worker component can be customized, and additional components can be defined by the user. The worker is built up using "bootsteps" — a dependency graph enabling fine grained control of the worker's internals.

To work with Celery, the following components are needed:

- A Celery module
- A message broker. This is an independent component used to send and receive messages to the distributed task workers. Essentially, it is a message oriented middleware that deals with the message exchange.

Each Celery module needs to establish a connection with the message broker. To achieve that, the task function and a broker keyboard argument are required. The broker keyboard argument is used to indicate the URL used to connect to the broker and be registered in the appropriate queue.

As mentioned above, Celery can use multiprocessing either asynchronous or synchronous. When creating a new task, the Celery module can be instructed to either run the tasks altogether or use a delay to the tasks and either wait for the previous task to be finished running or to have the module wait for a specific amount of time before sending them to the proper worker. Usually, the delay function is used because it is simpler.

Moreover, Celery can be used to inspect and manage worker nodes and tasks upon some degree. There are some inspect and control commands that implement the tasks. Some of them are:

- shell: Drop into a Python shell. The locals will include the celery variable: this is the current app. Also all known tasks will be automatically added to locals (unless the –without-tasks flag is set). Uses Ipython, bpython, or regular python in that order if installed. You can force an implementation using –ipython, –bpython, or –python.
- status: List active nodes in this cluster.
- result: Show the result of a task. The name of the task can only be omitted as long as the task doesn't use a custom result backend.
- purge: Purge messages from all configured task queues. This command will remove all messages from queues configured in the CELERY_QUEUES setting. It can also specify the queues to purge and exclude queues from being purged.
- inspect active: List active tasks.
- inspect scheduled: List scheduled ETA tasks. These are tasks reserved by the worker when they have an eta or countdownargument set.
- inspect reserved: List reserved tasks. This will list all the tasks that have been prefetched by the worker, and is currently waiting to be executed (doesn't include tasks with an ETA value set).
- inspect revoked: List history of revoked tasks.
- inspect registered: List registered tasks.

- inspect stats: Show worker statistics.
- inspect query_task: Show information about task(s) by id. Any worker having a task in this set of ids reserved/active will respond with status and information and also query for information about multiple tasks.
- control enable_events: Enable events
- control disable_events: Disable events
- migrate: Migrate tasks from one broker to another. This command will migrate all the tasks on one broker to another. As this command is new and experimental you should be sure to have a backup of the data before proceeding.

All these inspect and control commands support a –timeout argument. This argument is the number of seconds that the broker waits for the responses of the worker. Due to latency sustained under workload, this number can be increased so that every worker has the opportunity to reply.

Every task in Celery is recorded in a log file. This file is customizable depending on the developer's needs. It can be projected on the screen, written in a file and/or to a log management service. For these logs to be generated one can use the native Python logger or the built-in logger called celery.task logger. The Celery logger can be used with its default configuration or it can be customized. Because the Celery documentation is a bit sparse the custom logging handlers can be a bit trivial to use. The underlying Python logging system does not support all the concurrency settings Celery supports like eventlets, greenlets, prefork (subprocessing), threads and so forth. That is why Celery includes the celery.task logger class.

2.2.6 YAML

YALM (”YAML Ain't Markup Language”) is a human-readable data-serialization language that works well with modern programming languages for common everyday tasks. YAML was designed to be user friendly and useful for people working with data. It is used for configuring files and also for data storage and transmission. It uses Unicode printable characters, some of which provide structural information and the rest containing the data itself. YAML targets many of the same communications applications as XML but has a minimal syntax which intentionally breaks compatibility with SGML.

With regard to formatting, YAML uses a very user friendly way of writing the context. It uses similar indentations with Python and has some features that users uses in everyday life such us value pairs and dashes for lists. Many different custom data structures are allowed, but the YALM natively encodes three:

1. mappings (hashes/dictionaries)
2. sequences (arrays/lists)

3. scalars (strings/numbers)

These data types are based on the Pearl programming language, though other features have been inspired by other resources. For instance: the colon-centered syntax, used to express key-value pairs, is inspired by electronic mail headers as defined in RFC 0822, and the document separator "—" is borrowed from MIME (RFC 2046). Escape sequences are reused from C, and whitespace wrapping for multi-line strings is inspired from HTML. Lists and hashes can contain nested lists and hashes, forming a tree structure; arbitrary graphs can be represented using YAML aliases (similar to XML in SOAP). YALM leverages these data structures and creates an environment which includes a simple typing system and aliasing mechanism to form a complete language for serialization.

Although many languages use YAML for data serialization, YALM itself excels when used with languages fundamentally build with the same basic primitives. Some of these languages are Perl, Python, PHP, Ruby, and Javascript. YAML directly supports both collections (mappings, sequences) and scalars. Support for these common types enables programmers to use their language's native data structures for YAML manipulation, instead of requiring a special document object model (DOM). Even though YAML's potential is virtually boundless, it was specifically created to work well for common use cases such as: configuration files, log files, interprocess messaging, cross-language data sharing, object persistence, and debugging of complex data structures.

As every other language, YAML was created to meet some goals. The design goals for YALM are:

1. Be easily readable by humans.
2. YAML data to be portable between programming languages.
3. To match the native data structures of agile languages.
4. Have a consistent model to support generic tools.
5. Support one-pass processing.
6. Be expressive and extensible.
7. Be easy to implement and use.

There are hundreds of different languages for programming, but only a handful of languages for storing and transferring data. Some of the most known are XML and JSON. These languages have some things in common with YAML but there are some differences that assist us in choosing the right one for the type of structure we want to create.

Comparing YAML to XML one can observe that YAML lacks the notion of tag attributes that are found in XML. Instead, YAML has extensible type declarations (including class types for objects). YAML itself does not have XML's language-defined document schema descriptors that allow, for example, a document to self-validate. However, there are several externally defined schema descriptor languages for YAML (e.g. Doctrine, Kwalify and Rx) that fulfill that role. Moreover, the semantics provided by YAML's language-defined type declarations in the YAML document itself frequently relieve the need for a validator in simple, common situations. Additionally, YAXML, which represents YAML data structures in XML, allows XML schema importers and output mechanisms like XSLT to be applied to YAML.

Then, when compared to JSON it becomes apparent that the JSON syntax is the basis of YAML version 1.2. This version was promulgated with the express purpose of bringing YAML "into compliance with JSON as an official subset". Though prior versions of YAML were not strictly compatible, the incobatibilities were rarely noticeable, and most JSON documents can be parsed by some YAML parsers such as Syck. This is because JSON's semantic structure is equivalent to the optional "inline-style" of YAML. While extended hierarchies can be written in inline-style like JSON, this is not a recommended YAML style except when it aids clarity. YAML has many additional features lacking in JSON, including comments, extensible data types, relational anchors, strings without quotation marks, and mapping types preserving key order.

In the case of YALM's terminology, this specification uses keywords based on RFC2119 to indicate requirement level. In particular, the following words are used to describe the actions of a YAML processor:

- May: The word may, or the adjective optional, mean that conforming YAML processors are permitted to, but need not behave as described.
- Should: The word should, or the adjective recommended, mean that there could be reasons for a YAML processor to deviate from the behavior described, but that such deviation could hurt interoperability and should therefore be advertised with appropriate notice.
- Must: The word must, or the term required or shall, mean that the behavior described is an absolute requirement of the specification. The rest of this document is arranged as follows.

2.2.7 Green Unicorn Web Server Gateway Interface server (WSGI)

Green Unicorn, commonly shortened to "Gunicorn", is a Web Server Gateway Interface (WSGI) server implementation that is commonly used to run Python web applications.

A WSGI server is an interface for running Python web applications. When Python was a relatively new language, traditional servers in existence could not comprehend the code and assist the application to run on the web. In the 1990's a developer named Grisha Trubetskoy developed a module called mod_python for Apache server, a well-known windows server created from volunteers all around the world. The module created was to execute arbitrary Python code. After 1995 when the first big update of Apache server happened, Apache server was configured with mod_python and could run most of the Python web applications.

However, mod_python was not stable and the Apache community was stalling the development to fix the issues. It was a mere extension to the existing server. This is when the need to create a new server to cater to the needs of every Python-based application became prominent and the Python community then created the WSGI, Web Server Gateway Interfaces.

WSGI is a simple and standard interface to run Python code that runs in a different port than the web server. The web server passes incoming requests to WSGI and the WSGI responds with HTML code to the request. Such interface provides developers the flexibility to swap between different WSGIs without changing any part of the code and allows them to use any framework or server they prefer. In addition, it makes scaling the web application effortless. Once the requests arrive to the WSGI, it decides how to communicate each request to a framework's process. The segregation of responsibilities is important for efficiently scaling web traffic.

Gunicorn is a WSGI server. It is built in a manner that aims to allow as many web servers as possible to be able to interact with it. No matter which framework the application is built upon, as long as the application is compatible with the WSGI Gunicorn will be able to run it. For WSGI to have a smooth cooperation with the framework, the need for a universal interface arose. This interface was named PEP 333 and its latest update PEP 3333. PEP 3333 makes sure that the WSGI and the framework can communicate, meaning that the server side invokes a callable object that is provided by the application side. Gunicorn has three main responsibilities. To check every aspect of the application to make sure that the application is healthy and operates correctly, to distribute requests across multiple instances of the application that it has already run beforehand and to communicate with the server. The described functionality operates off a central master process that distributes the requests to the worker processes which handle every responsibility. There are many types of workers:

- Sync Workers: the default worker type, it processes one request at a time and does not support maintaining a persistent connection with the application.

- Async Workers: available only through Greenlets, an implementation for Python, can be used without changes in the code of the application.

- Tornado Workers: only available for the tornado framework, there are not a recommended configuration.

- AnsyncIO Workers: they are compatible only with Python 3. There are two types of AnsyncIO Workers:

  1. gthread, which accept connections and remain open when requested, executing every request in the thread.
  2. gaiohttp, a full AsyncIO worker which uses aiohttp, an asynchronous HTTP Client/Server.

Gunicorn is a very fast and reliable WSGI which is constantly updated, can be scaled with the use of more threads and gets constantly extended to make sure that as many frameworks as possible will be supported without any issues.

2.2.8 Urllib3

Urllib3 is a powerful, sanity-friendly HTTP client for Python. Urllib3 is widely used in many Python-based applications and implements many critical features for network requests, such as:

- Thread safety.
- Connection pooling.
- Client-side SSL/TLS verification.
- File uploads with multipart encoding.
- Helpers for retrying requests and dealing with HTTP redirects.
- Support for gzip and deflate encoding.
- Proxy support for HTTP and SOCKS.

2.2.9 Multithreading

More often than not, an application needs to break down complex calculations and data processing into smaller tasks and run them concurrently so as to take advantage of the processing power of the CPU. In order to succeed in such an endeavor, threads and multithreading become prominent. Threads create the derived code execution and multithreading assigns them to different processors.

A thread represents a flow of execution. To be more precise, when a thread is created, a new code execution path that allows the program or application to run is generated. In Python 3 just creating a thread does not mean it will run concurrently to other threads of the same application but instead they will run one at the time.

Multithreading handles the threads in a manner where they will only appear as running concurrently without actually doing so. Multiprocessing on the other hand, would create different processes that would run concurrently, in different CPUs. The threads will only appear to run concurrently because in reality multithreading does not create different processes for the code to be executed (as would be useful in CPU intensive work) different CPUs but instead used only one CPU and switches it doing work between the threads in a fitting manner. Whenever the code reaches upon any Input/Output operation, instead of keeping the CPU idle until such operation is completed, the multithreading framework in Python switches the active thread to another one. The new thread again is executed in the CPU until it either finishes or it reaches an I/O operation, where it would be switched out with another one.

2.2.10 Python Logging

Both during development and when an application is run by the user, logging contributes to the better understanding of how the application works and its current state. In Python, the logging feature is embedded in the standard library and is easy to incorporate into an application. Logging can provide insight on the applications functions, variables, state and errors for debugging the code. Also it can be used to show progress to the user as the application is running, interact with them and guide them on how the application can be used correctly.

The default Python logger supports 5 logging levels, to be used depending on the severity of the event. To log each level, there is a different method that can be called. The defined levels, in order of increasing severity, are the following:

- DEBUG
- INFO
- WARNING
- ERROR
- CRITICAL

Apart from the default logger, Python offers the option to create a custom logger by creating an object of the class Logger. Beside the Logger class, the Python logging module contains a few more useful classes. Namely they are:

1. Logger: This class's objects are used in the application's code to call the functions.

2. LogRecord: Loggers automatically create LogRecord objects that have all the information related to the event being logged, like the name of the logger, the function, the line number, the message, and more.

3. Handler: Handlers send the LogRecord to the required output destination, e.g. the console or a file. The Handler class is a base for subclasses like StreamHandler, FileHandler, SMTPHandler and, HTTPHandler. These subclasses hand off the output stream to their respective output channel. Also, handlers are used to define the directory in which the log is going to be saved in. Using multiple handlers, the logger's output can be forwarded towards multiple destinations. For example saved in a file and in an output stream

4. Formatter: The format of the log can be configured through the Formatter class. It allows a format string list the attributes and define the template with which the logger's output will be generated with.

In conclusion, logging plays a vital role in better understanding the code, the execution of the application and any errors that might occur during runtime. It is a way to communicate with users but also a great versatile debugging mechanism.

2.2.11 SURT

SURT stands for Sort-friendly URI Reordering Transform. The Uniform Resource Identifier (URI), is the way the worldwide web's identification system allows everyone to share information with the rest of the world. There are substantial benefits to participating in the existing network of URIs, including linking, bookmarking, caching, and indexing by search engines, and there are substantial costs to creating a new identification system that has the same properties as URIs.

SURT is a URI Reordering Transform library which makes their left-to-right representation better match the natural hierarchy of domain names. Conversion to SURT form also involves making all characters lowercase and changing the 'https' scheme to 'http'. Further, the '/' after a URI authority component – for example, the third slash in a regular HTTP URI – will only appear in the SURT form if it appeared in the plain URI form.

SURT form URIs are typically not used to specify exact URIs for fetching. Rather, SURT form is useful when comparing or sorting URIs. For example, URIs in SURT format sort into natural groups – a, All 'archive.org' URIs will be adjacent, regardless of what subdomains like 'books.archive.org' or 'movies.archive.org' are used. Most importantly, a SURT form URI, or a truncated version of a SURT form URI, can be used as a SURT prefix. A SURT prefix will often correspond to all URIs within a common 'area' of interest for crawling.

In order a URI to apply to the SURT form, it must use a "SURT prefix". A collection of sorted SURT prefixes is an efficient way to specify a desired crawl scope: any URI whose SURT form starts with any of the prefixes should be included. A small set of conventions can be also be used to calculate an "implied SURT prefix" from a regular URI, such as a URI supplied as a crawl seed. These conventions are:

1. Convert the URI to its SURT form.

2. If there are at least 3 slashes ('/') in the SURT form, remove everything after the last slash. As examples:

    - <http://(org,example,www,)/main/subsection/> is unchanged;
    - <http://(org,example,www,)/main/subsection> is truncated to <http://(org,example,www,)/main/>;
    - <http://(org,example,www,)/> is unchanged; and
    - <http://(org,example,www,)> is unchanged.

3. If the resulting form ends in an off-parenthesis (')'), remove the off-parenthesis. So each of the above examples except for the last is unchanged, while the last <http://(org,example,www,)> becomes <http://(org,example,www,>.

2.2.12 Redis

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

Redis can be used also for atomic operations. It has an in-memory dataset. This dataset can be either abdicated on the hard drive or logged. The database can also be dropped if there is a need for a feature-rich, networked, in-memory cache. Redis has some features that make it stand out. Some of them are:

1. Speed. It is written in C, which makes it fast.
2. It is a NoSql Database.
3. It is currently being used by tech-giants like GitHub, Weibo, Pinterest, Snapchat, Craigslist, Digg, StackOverflow and, Flickr.
4. It is open source and stable.
5. Being open source, it is developer friendly. Redis can be used in most of the programming languages like JavaScript, Java, Go, C, C++, C #, Python, Objective-C, PHP and almost every famous language out there.
6. It works in most POSIX systems like Linux, BSD and, OS X without any external dependencies. Linux and OS X are the two operating systems where Redis is more widely developed and tested. Redis may also work in Solaris-derived systems like SmartOS, but the support to such systems is best effort. There is no official support for Windows builds, but Microsoft develops and maintains a Win-64 port of Redis.

2.2.13 BeautifulSoup

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It musters a parser used for providing idiomatic ways of navigating, searching, and modifying the parse tree.

The advantages of BeautifulSoup is that it can parse HTML like a simple XML and return the values required with much ado. This is particularly helpful in web scraping. Many websites have large collections of pages generated dynamically from an underlying structured source like a database. The data of a category are typically encoded into similar pages by a common script or template and parsing them with BeatifulSoup is trivial.

In recent years, some value-added services, such as comparison shopping and vertical search in a specific domain, have motivated the research of extraction technologies with high accuracy. BeautifulSoup can collect all the information given from some Web pages and can recreate them into a more readable and structured data form. That data form can be used to extract conclusions about the data faster and modify the parts that are not as efficient as it need be.

2.3 web-monitoring-processing

For completion purposes, since the web-monitoring-processing project is important to this Thesis but not inside of its scope, the work done on web-monitoring-processing is going to be briefly mentioned.

Since web-monitoring-processing was going to be used as a whole, it constituted the best place to begin work for the Google Summer of Code. The aim was to identify issues and ways to improve this application. The wm-diffing-server was the most important component of the application and thus the main point of insterestinterest for further work.

In total, 3 Pull Requests (see table 2) that consisted of 12 commits (see table 3) were made on this project.

Pull Request	Link
Diffing server exception handling #185	PR #185
Add tornado debug env value #187	PR #187
Cors #188	PR #188

Table 2.1: Pull requests made to EDGI's web-monitoring-processing's repository.

Commit & Link
Returning some exceptions in JSON format
Code cleanup, used send_error method
Handling more errors
Diffing server unit tests, typo correction
Allocating a new port for each test, all tests working
Implementation of requested changes
Implementation of requested changes 2
Implementation of requested changes 3
Add tornado debug env value
Read env at top, fix string boolean value
Change DEBUG env value name
Handle env values better

Table 2.2: Commits made to EDGI's web-monitoring-processing's repository.

2.4 Common Technologies

This chapter is an introduction to the technologies that are used by both wayback-diff and wayback-discover diff.

2.4.1 JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write and easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language. JSON is built on two structures:

- A collection of name - value pairs. In various languages, this may be realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this may be realized as an array, vector, list, or sequence.

The above mentioned are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with most programming languages can also be based on these structures. In JSON, they take on these forms:

- An object is an unordered set of name/value pairs.
- An array is an ordered collection of values.
- A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.
- A string is a sequence of zero or more Unicode characters. A character is represented as a single character string. A string is very much like a C or Java string.
- A number is very much like a C or Java number, except that the octal and hexadecimal formats are not used.
- Whitespace can be inserted between any pair of tokens.

2.4.2 CORS

CORS (cross-origin resource sharing) manages cross-origin requests. Managing cross-origin requests means providing the server the ability to handle requests originating outside of their own resources. CORS, among many other things, allows servers to specify which domains can access their assets. Allowing cross-origin requests might be helpful when using different servers to load resources such as stylesheets, scripts and, images.

The default configuration in most web servers might allow Cross-Origin requests. However, HTTP requests methods like PATCH, PUT, or DELETE may be denied to prevent malicious behavior. This happens in order to enhance the security of the web. Handling such requests from a different server might prove to be malicious. As a result, many modern browsers follow security policies to mitigate such risks. Even if a server allows CORS requests, the same-origin policy in browsers is very restrictive. Under this policy, a document hosted on server can only interact with other documents that are also on the same server. In short, the same-origin policy enforces that documents that interact with each other have the same origin. An origin is made up of the following three parts:

- The protocol which defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
- The host the server that hosts the document we can load on our browser.
- The port number.

Not having a security policy would be risky, but a security policy like same-origin might be found restrictive in some scenarios. To control which level of Cross-Origin requests are allowed, specific HTTP headers are used. An HTTP header is a piece of information associated with a request or a response. The following headers are passed back and forth between the client and a server when the web page wants to use resources hosted on a different server. The CORS standard manages cross-origin requests by adding new HTTP headers to the standard list of headers, that are used to describe requests and responses. The new HTTP headers added by the CORS standard are:

1. Access-Control-Allow-Origin: Indicates whether the response can be shared.
2. Access-Control-Allow-Credentials: Indicates whether the response to the request can be exposed when the credentials flag is true.
3. Access-Control-Allow-Headers: Used in response to a preflight request to indicate which HTTP headers can be used when making the actual request.
4. Access-Control-Allow-Methods: Specifies the method or methods allowed when accessing the resource in response to a preflight request.
5. Access-Control-Expose-Headers: Indicates which headers can be exposed as part of the response by listing their names.
6. Access-Control-Max-Age: Indicates how long the results of a preflight request can be cached.
7. Access-Control-Request-Headers: Used when issuing a preflight request to let the server know which HTTP headers will be used when the actual request is made.
8. Access-Control-Request-Method: Used when issuing a preflight request to let the server know which HTTP method will be used when the actual request is made.
9. Origin: Indicates where a fetch originates from.

There are many resource sharing solutions for all web technologies. The security concept, however, will always be similar so that larger risk can be avoided. By understanding security policies like CORS, it is easy to understand how risky behavior is abated.

2.4.3 Simhash

Most hashing algorithms, when they hash data, produce a unique output. This will be completely different no mater the percentage of change made on the hashed data. Even if the data has changed only by a bit, the output of the hashing algorithm will be greatly different. But what might also be important information is how much the data has changed. When it comes to figuring out how similar two pieces of data are to one another traditional hashing algorithms are inadequate by design. For a person, it is very easy to understand the differences and the similarities between some data. But when there are hundreds of thousands of data, the need of a fast algorithm that can calculate the similarity of data is vital.

The Simhash algorithm with its simple and brilliant approach, helps mitigate this issue. Other hashing algorithms such as the MD5 or the SHA, aim to very quickly create a unique signature (hash) of the data. While Simhash still aims to have unique signatures for documents, also attempts to make sure that documents with nearly identical characteristics get very similar hashes. In this way, its output can be used to figure out if two pieces of data are related and to extract a quantitative value as to their similarity level. It's a statistical tool to assist in the search of near-duplicates.

2.4.4 xxhash

xxHash is a hashing algorithm designed from the ground up to be as fast as possible on modern CPUs. It is not a strong cryptographic hash, such as the SHA family, but still passes the SMHasher test set with 10 points.

Most simple hashes, such as FNV, step through the input data byte-by-byte. Working on byte at position 1 requires that all work on the previous byte (at position 0) has finished. Therefore, a dependency which causes the CPU to potentially wait (stalling) exists. Slightly better algorithms process a block of bytes at once, e.g. CRC Slicing-by-4/8/16 consumes 4/8/16 bytes per step instead just one, offering a substantial speed-up. However, have the same issue persists. Working on block 1 requires that all work on block 0 has finished.

xxHash subdivides the input data into four independent streams. Each stream processes a block of 4 bytes per step and stores a temporary state. Only the final step combines these four states into a single one. The major advantage is that the code generator has lots of opportunities to re-order opcodes to avoid latencies.

2.4.5 base64

Base64 is a group of similar binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation. Base64 was used to encode arbitrary octet sequences into a form that satisfies the rules of 7bit. It is also designed to be efficient for non-text 8 bit and binary data. Sometimes it is used for text data that frequently uses non-US-ASCII characters. In addition it is suitable for use with SMTP servers that support the 8BITMIME SMTP extension. Base64 is commonly used in a number of applications including email via MIME, and storing complex data in XML. Base64 is particularly prevalent on the Web where its offers the ability to embed image files or other binary assets inside textual assets such as HTML and CSS files.

In JavaScript there are two functions respectively for decoding and encoding base64 strings only:

- atob(): A function that decodes a string of data which has been encoded using base-64 encoding.
- btoa(): A function that creates a base-64 encoded ASCII string from a ”string” of binary data.

Base64 is a reliable encoding algorithm for data, but, has some disadvantages:

- base64 encoding makes file sizes roughly 33% larger than their original binary representations
- Data URIs aren’t supported on IE6 or IE7
- base64 encoded data may possibly take longer to process than binary data

Even though encoding data to base64 will increase of the size of the data, there are some cases this is useful. Base64, even though requiring more memory for the same data, will encode them to a shorter string length. Thus, in applications where a lot of data are transferred and their length might cause issues, base64 is useful.

Chapter 3

Deliverables

This chapter is purposed to be an explanation handbook for the two applications developed for the Internet Archive during the Google Summer of Code 2018. More specifically, the algorithm and structure of the applications is explained along with flow diagrams which further assist the reader to gain a tight grasp on the project's inner workings. Last but not least, benchmark tests on wayback-discover-diff to identify and mitigate areas where the application's performance was lacking were executed. Their results and actions taken because of them are demonstrated as well.

3.1 wayback-diff

The entry point of the application is located in the *source/index.js* file. The architecture of the application is purposed so as to generate and export the application into a single Javascript file. Exporting it as a library makes using it in another project as simple as typing *"import wayback-diff"*. This is enables wayback-diff to run both as a standalone application but also be included in a bigger platform such as the Wayback Machine.

React's render function is the function that the frameworks expects a DOM element to be returned for each component. In wayback-diff's entrypoint file the element which will be rendered is a React Router. Router, as mentioned above, is a component which depending on the browser's path, manages what will be rendered and with what parameters. This is accomplished with Routes. The last route in Router, diffgraph. is outside the scope of this thesis and hence it will not be discussed further. The rest of the routes represent the various story-cases of wayback-diff.

wayback-diff supports two URL schemas.

http://localhost:port/diff/WEBSITE

and

http://localhost:port/diff/TIMESTAMP_A/TIMESTAMP_B/WEBSITE

The main, top level component of wayback-diff is DiffContainer which receives 6 parameters as props.

The **fetchCDXCallback** prop, a callback function used to fetch the snapshots available from the CDX server.

The **fetchSnapshotCallback** prop, a callback function that is used to fetch the snapshots from the Wayback Machine.

- If null is passed to either one of the fetchCallback props a default fallback method is going to be used instead.

- The callback functions should return a fetch Promise.

If this prop is used, the limit configuration option will not have any effect.

The **loader** prop, a React Component that is displayed when loading. If this is not set or it is null, a default loader will be used instead.

The **timestampA** and **timestampB** props are the timestamps extracted from the URL.

The **url** prop is the webpage for which the snapshots are requested.

The **noTimestamps** prop which should only be set to true in the */diff///WEBPAGE* path schema.

The timestamps are extracted from the variables *"match"* and *"location"* which are part of the Router's context. The configuration file is imported with the command *"require"*.

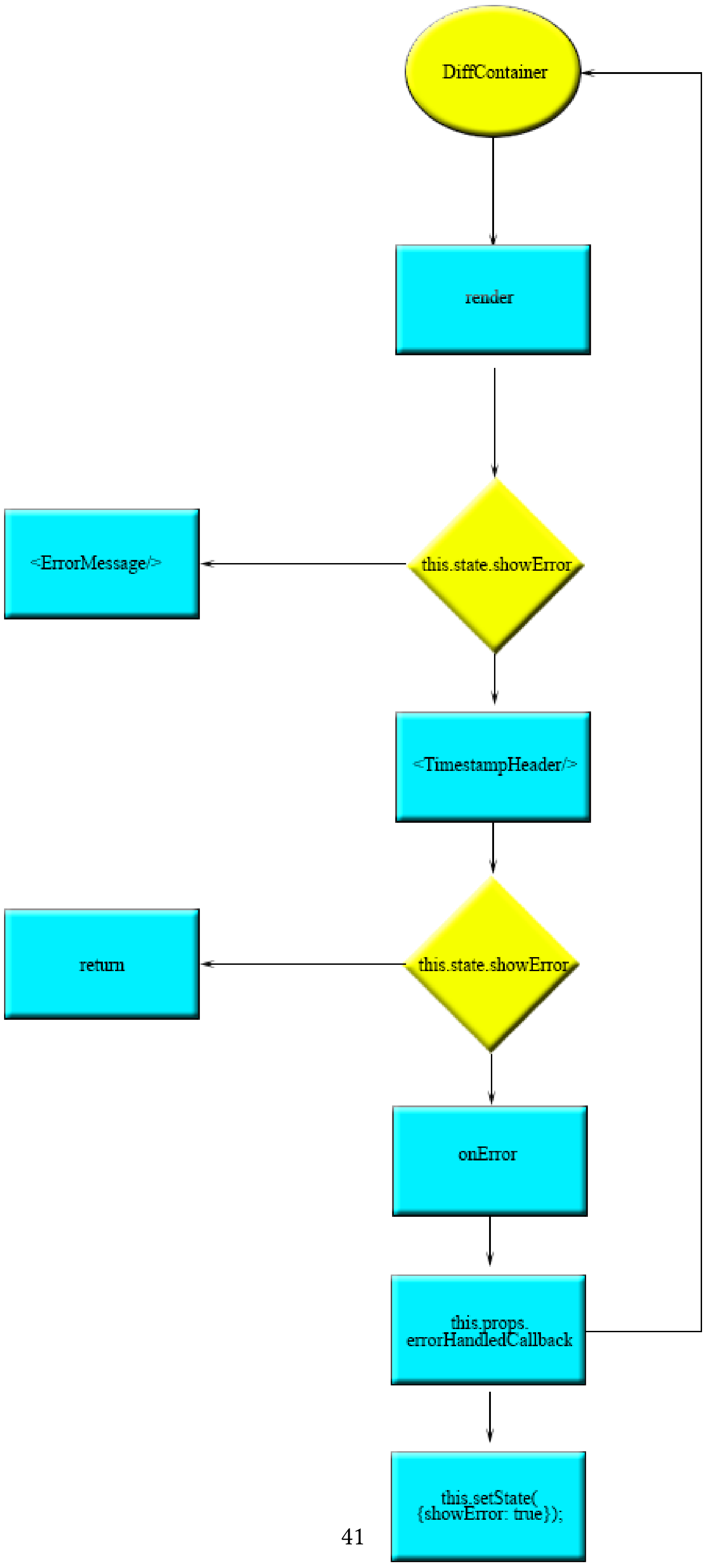
3.1.1 DiffContainer component

The most of the work for genesis of the DIffContainer component is done in the render function. At first, the URL is verified using a utility function. If it is not, a Bootstrap Alert element is returned displaying the corresponding error message.

At this point it is important to mention that the state is generally used to control which component, if any, needs to be rendered and to be able to re-render it in case its data is updated. In other words, it is used as a control variable. If the state indicates that an error needs has occurred like so, the showError state variable has a value, then render function returns an Error component. The Error component returns the corresponding error message. In general, if an error occurs anywhere in the wayback-diff application, only the DiffContainer component handles it. All the other components stop rendering and pass the error code up the hierarchy until it reaches DiffContainer using callbacks.

A simplified flow chart that describes the error handling in wayback-diff's DiffContainer and TimestampHeader components follows.

Figure 3.1: Wayback-diff error handling simplified flow chart



If the URL is valid and there is no error to show, the Timestamps received in the component's Props are checked. If neither TimestampA nor TimestampB are included and the noTimestamps variable is true, the timestamp header with a div which contains a message saying that the user needs to choose some timestamps is rendered.

If both of the timestamps are present in props then three components are rendered. The TimestampHeader, which in this case needs to fetch and select the requested timestamps in its select elements, the DiffView component, which contains the comparison iFrames and the DiffFooter component.

The comparison iFrames are initialized from the **prepareDiffView** function. This function is a middleware function for calling a modified version of the **DiffView** component of the *web-monitoring-processing-ui* project by EDGI. The **DiffView** component was modified for the purposes of this project so its **loadDiffData** function can fetch the snapshots from the *web-monitoring-processing* backend and render their changes. Also, minor modifications were made to the **SideBySideRenderedDiff** component, returned from the **DiffView component**, so it can receive and render the loader that is set in the *index.js* entry file of the React application. These two components were forked from the *web-monitoring-processing-ui* project and were not developed by the author of this thesis. For this reason their inner workings are outside the scope of this thesis and they are mentioned for completion purposes only.

If only one of the timestamps is present in the URL then the **DiffContainer** component renders the *TimestampHeader* component along with two iFrames. Depending on which timestamp is missing, one iFrame contains a message informing the user that a snapshot was not selected and the other one renders the snapshot as is. This means that the snapshot does not go under any processing before being displayed since there is no other snapshot to compare it to.

3.1.2 TimestampHeader Component

The **TimestampHeader** component receives the loader component from Props and initializes it. The component then checks if it has the required data available so it can be drawn in the browser or if it should request it. If CDX data are not found, the download process starts in the **fetchCDXData** function. During this, the loader component is shown.

The **fetchCDXData** function builds the request that will be sent to the CDX endpoint and initializes it along with an ABORT signal. The request's result is handled by the **handleFetch** function. This function handles the request's response. If it receives an error message, it informs the error handler with the appropriate error code, otherwise it sends the data to the **prepareData** function and, if the URL has timestamps, it makes sure the correct values are marked in the User Interface.

ADD EXAMPLE CDX RESPONSE

The CDX server has a wide range of data available for each snapshot of a webpage. The information from the CDX server that is useful for wayback-diff is the timestamps and each of the timestamps' digest. The timestamp's digest is a unique value, not unlike a hash value of an encryption algorithm, which describes each snapshot. This value is used to filter identical snapshots and prevent a user from comparing snapshots with the same content against each other.

The **prepareData** function updates the state with the **cdxData**, **leftSnaps** and **rightSnaps** and, after the necessary processing, the **leftSnapElements**, the **rightSnapElements** and **headerinfo** variables. **cdxData**, **leftSnaps** and **rightSnaps** contain the reply of the CDX query. The **leftSnaps** and **rightSnaps** variables are proposed to contain the filtered versions of the CDX data. The **leftSnapElements**, the **rightSnapElements** variables contain the option DOM elements that are going to be displayed in the select DOM elements of the component. Finally, the header info contains the paragraph DOM element that provides information about the webpage.

After this point, either because of a value change of the **showError** state variable or a change of the state's cdxData variable, a re-render will be caused. A change in these two variables would cause a re-render because two conditions are met.

1. They are variables saved in state.
2. These variables are used in flow control operators in the component's render function.

If no error has occurred so far, a validation check is performed for the requested timestamps. This validation ensures that both timestamps, or just the one if the second one was not provided, added are valid. If they are not, they are corrected. This is done by requesting the snapshots that correspond to the entered timestamps from Wayback Machine. More specifically, the request object is set to follow possible redirects which are done from the server's side. If a requested snapshot does not exist, the Wayback Machine redirects the browser to the closest one. As follows, when the request has finished the timestamp in the response URL is checked against the one provided. If it is different from the ones provided, it is amended, the state timestamps variables are updated and the **DiffContainer** component is notified using callbacks. Through this callback the **DiffContainer** component updates the URL of the browser to match the returned timestamps.

When all the above mentioned procedures are complete, the Timestamp-header component is ready to be rendered. Its user interface is composed by three functions:

1. **showInfo**, which returns the **headerInfo** state variable's value as it was calculated after the processing of the cdxData.
2. **showOpenLinks**, which, depending on how many timestamps were inserted in the URL, shows above the corresponding right or left iframe an "Open in new window" link which takes the user to the Snapshot of the page that it is comparing. This offers users the opportunity to view it without the additional markup that was added during the wayback-diff comparison.
3. **timestampSelect**, which returns a select element on the left, 2 buttons in the middle and another select element on the right. The option elements for each select element are derived from state, and more specifically, from the variables **leftSnapElements** and **rightSnapElements**

which were calculated in the **prepareData** function earlier. The **onChange** event of these options is set. When the chosen index of either of the two select elements is changed, the other's data are filtered so that timestamps with the same digest, and thus the same content, are not shown. The two buttons in the middle are the Restart button which resets all data in the select elements unfiltered and the Show Differences button which, with the corresponding callback, updates the **DiffContainer** component that the chosen timestamps have been changed by the user and need to be shown.

All fetch requests that are made are shown by **fetch_with_timeout**. This function makes sure that the requests on all endpoints expire if no answer is received after a set amount of time and that they can be aborted with a global ABORT signal. The application's structure is such that it attempts to handle errors in a graceful manner. In the case that they can't be handled, they are critical and the application should be stopped. The global abort signal guarantees that if somewhere in the wayback-diff container an error is thrown, the application, and thus all the requests running in the background, need to stop and the appropriate error message to be shown.

The error handling is done in the **DiffContainer** component. If at any point of the code or in any component an error is thrown, the codes and the error conditions are hierarchically raised up to the **DiffContainer**, with the use of callbacks, so that they can be handled there exclusively. The default behavior of the other components in case of an error is to not be rendered.

3.2 wayback-discover-diff

This app has a few endpoints. One of them causes the app to calculate data in an asynchronous manner, abstract from the interaction with the client that made the request and the other three cause the app to return data to the requester.

/calculate-simhash?url=URL&year=YEAR

The calculate-simhash endpoint receives two parameters, the URL and year. These represent the webpage and the year that the application is going to request the snapshots and calculate their simhash value for. It runs a background task to calculate simhash for all captures of target URL in the specified year and it returns a JSON file stating the Job ID of the initiated task and its status.

Return JSON “status”: “started”, “job_id”: “XXYYZZ (uuid)”

/simhash?url=URL&timestamp=timestamp

The simhash endpoint receives two parameters that define for which URL and timestamp pair should the application return the simhash value for. In this endpoint, in retrospect with the timestamp validation and handling that happens on wayback-diff, it is important for the application to receive the exact and correct timestamp. In the event that the timestamp may not be correct, it will not be verified and the application will return "None" to the requestee.

The format of the JSON response generated for this request is the following:
Returns JSON "simhash": "XXXX"

if the capture simhash has already been calculated or None if is not found in the Redis database.
A URL and timestamp combination might not be found on the Redis database for two reasons.

1. The requested snapshot has not be processed and thus no simhash value exists for it.
2. The requested timestamp does not correspond to a snapshot for this URL.

/simhash?url=URL&year=YEAR

The simhash with URL and YEAR as a parameter instead of the above mentioned timestamp parameter returns all the timestamps for which a simhash value exists in the DB for that specific URL and year. The response has the following format:

[["TIMESTAMP_VALUE", "SIMHASH_VALUE"]]

/simhash?url=URL&year=YEAR&page=PAGE_NUMBER

Since the data returned from the above mentioned request can be a lot, it is useful to be able to receive the response in pages. Adding a page parameter would allow that. Depending on how the application is configured, the results are paginated with a preset length. The response schema also changes a bit and has the following format :

[["pages","NUMBER_OF_PAGES"],["TIMESTAMP_VALUE", "SIMHASH_VALUE"]]

The SIMHASH_VALUE that is returned from both endpoints is base64 encoded.

/job?job_id=<job_Id>

After initiating a background job from the calculate-simhash endpoint, clients need to check on that job's status and result. The JOB endpoint queries the database to see if the job has finished. The returned JSON has the following format:

Returns JSON “status”: “pending”, “job_Id”: “XXYYZZ”, “info”: “X out of Y captures have been processed”

This application is made up by three main components.

1. The Redis server, which acts as a database and message broker for the Python application.

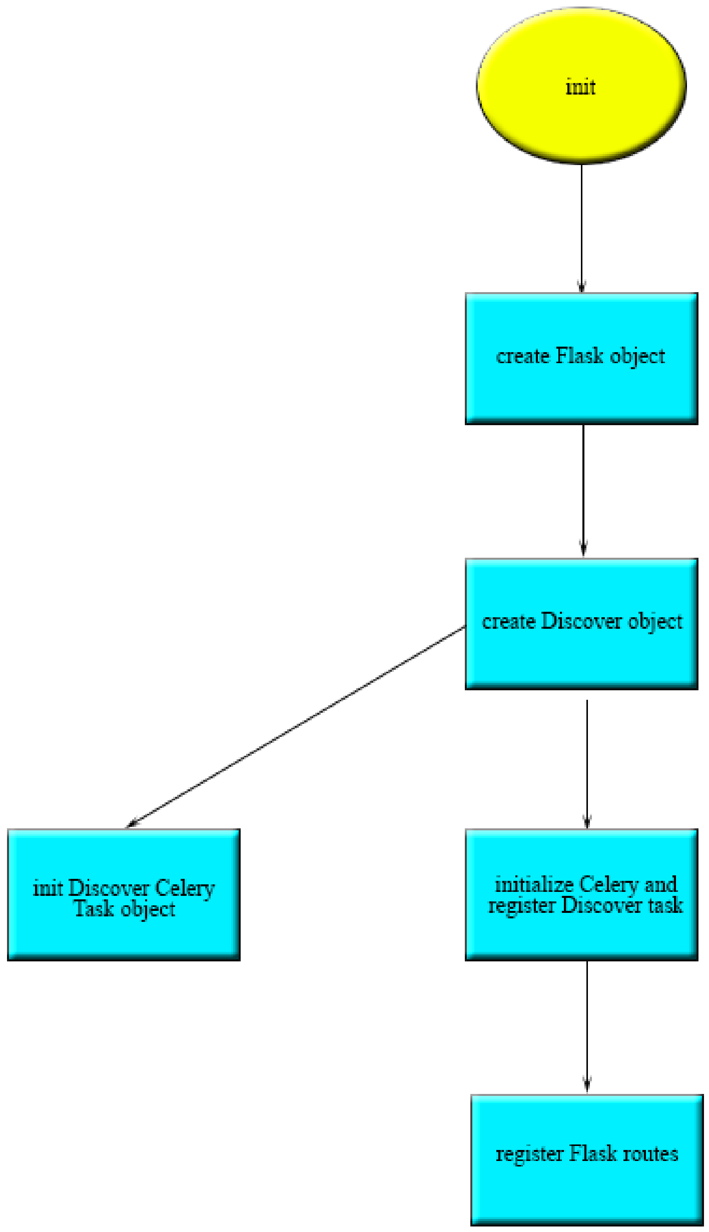
2. The gunicorn web server that together with Flask serve the incoming HTTP requests.

3. The Celery worker which sends Tasks to the workers as needed, in harmony with the incoming requests and the configuration of the application.

The starting point of this application and all of its components is the ***run_gunicorn.sh*** file. This file initiates all three components in the background with the configuration as is laid out in a ***conf.yaml*** file.

The application module that is served though gunicorn and Flask is initialized in the *__init__.py* file. The reason why gunicorn was selected to support Flask is performance. Although Flask includes a web server, it is not designed to live up to the requirements of a real life, live server. Gunicorn on the other hand is optimized for such cases and its easy integration with Flask makes it an excellent partner.

Figure 3.2: __init__.py simplified flow chart.


init
create Flask object
create Discover object
init Discover Celery Task object
initialize Celery and register Discover task
register Flask routes

It also creates a Discover object, initializes Celery and registers the Discover task. In addition it reads the configuration file to retrieve the domains from which CORS requests are going to be allowed, if any.

All the endpoints except for the **CALCULATE_SIMHASH** one, are run in in the gunicorn workers. Celery workers are not used to execute this part of the code. This decision was made because the work done for these endpoints is not intensive enough to justify the performance overhead of running in an asynchronous Celery worker.

Gunicorn receives the incoming requests and its workers hand them over to the Flask framework.

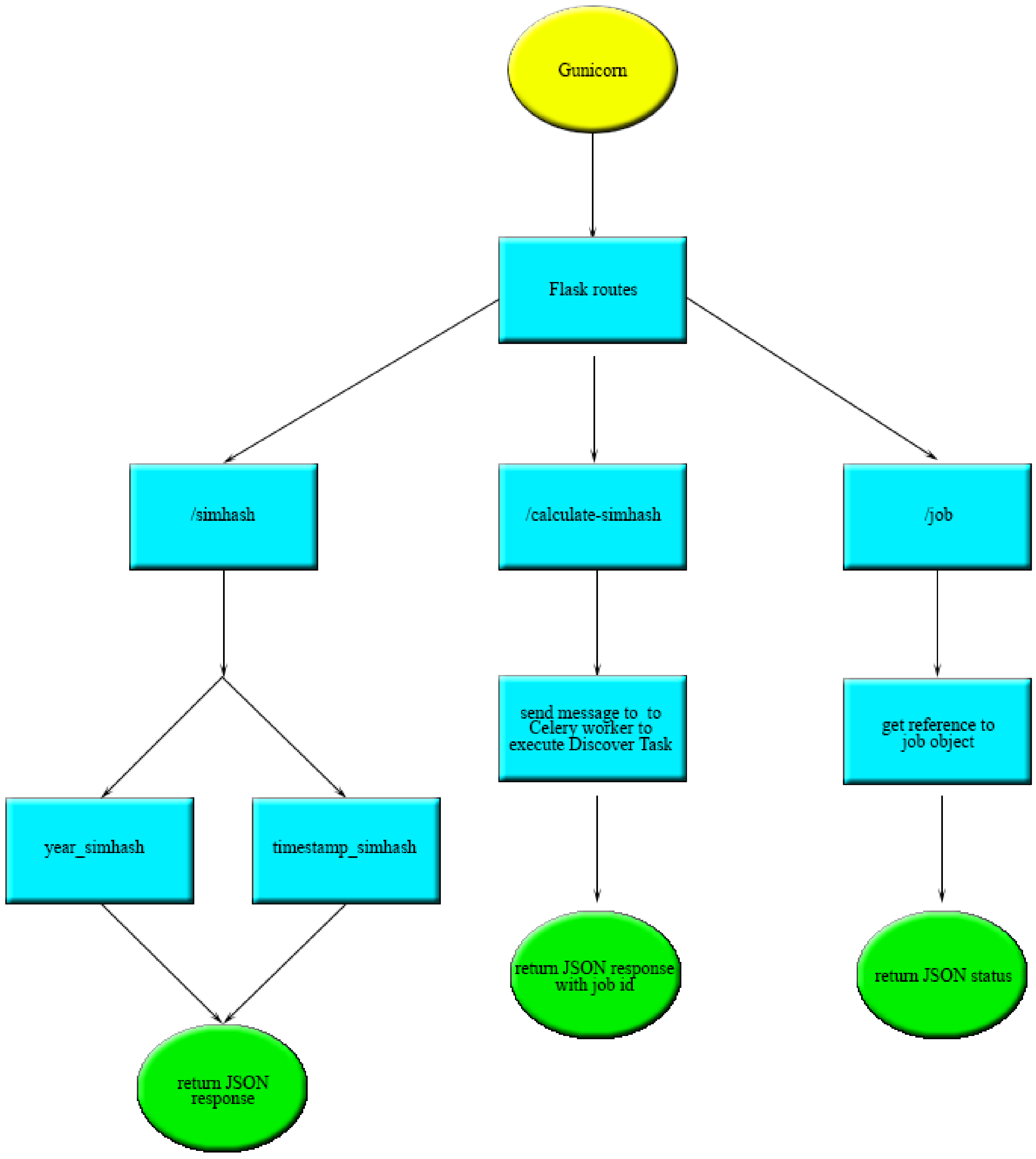
Flask resolves the request's route and follows the necessary work flow.

The **JOB** route extracts the submitted job_id from the request and extends to obtain a reference to that task. Depending on that task's state, whether it is pending, stopped running due to an error or has completed it constructs and returns the appropriate JSON response.

The **SIMHASH** path, validates the request's parameters and depending on which they are, it executes the **year_simhash**, or **timestamp_simhash** function of the Discover object. Even though these two functions are included in the Discover object which inherits the Celery Task class, they are run in the gunicorn worker and not in a Celery worker. They are included in the Discover class for cohesion because of the nature of the work they do.

The **CALCULATE_SIMHASH** route validates the parameters it received and proceeds to send a message to the Celery worker to execute the Discover Task with the appropriate parameters. Then it returns a JSON response with the job status as started and the job_id of the just initiated Celery task.

Figure 3.3: Flow chart showing how each request is handled.



The Discover class is a subclass of the Task class from the Celery framework. In its **init** function the configuration file is extracted into object variables and a logger among with a HTTP request pool manager and a connection to the Redis database is initialized. The run function is the code that is executed from the Celery worker when the message to execute the Discover task is sent to it. The algorithm that guides the process of calculating the simhash values of the snapshots of a webpage has six steps:

1. Fetch all the available timestamps for snapshots of the selected year for this particular webpage.

2. Detect which snapshots have the same digest value and split them into a **duplicate** and **non duplicate** list.

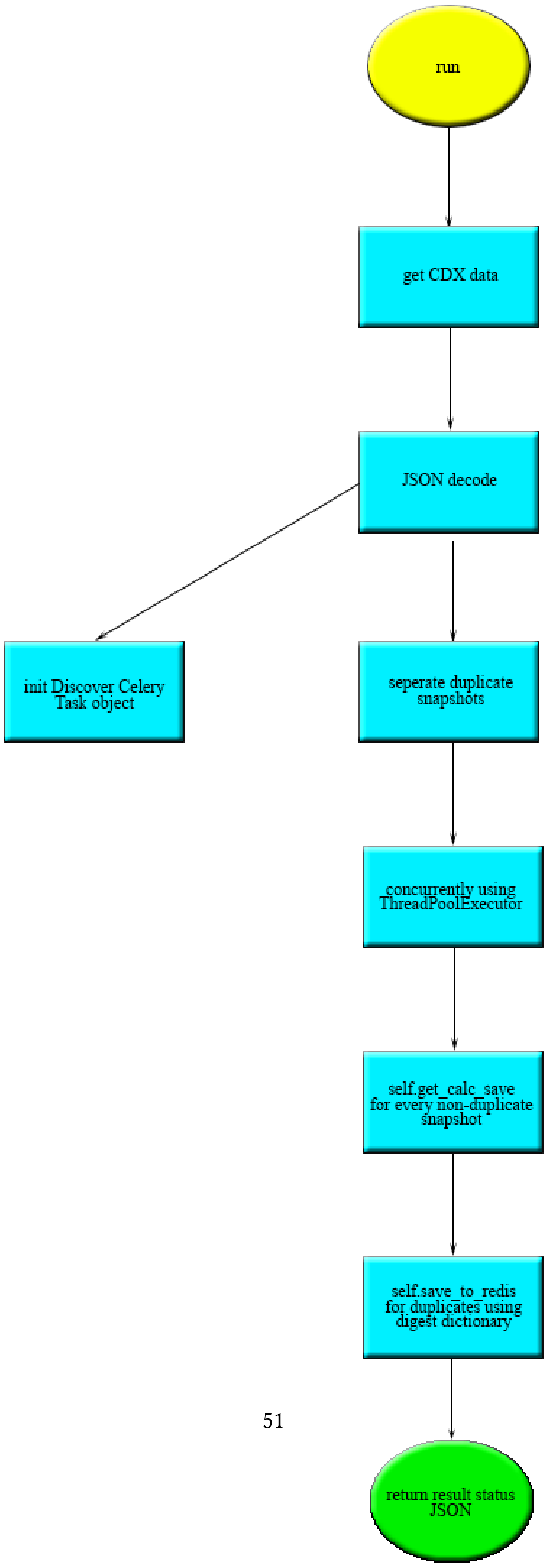
3. Using Python's **ThreadPoolExcecutor**, for every distinct simhash value execute the **get_calc_save** function.

4. For every non distinct (**duplicate**) element, find the already calculated simhash value using the digest value as a key and using that value save to Redis the Timestamp - Simhash value pair.

5. Set these database entries to expire after a period of time set in the configuration file. The reason why the entries of the database need to expire is purely because of system limitations. It would not be practical, after a while, to have a database of a few gigabytes or terabytes since this information is not going to be frequently accessed. Instead it is more cost-effective to purge the entries of the database and calculate them again if and when needed.

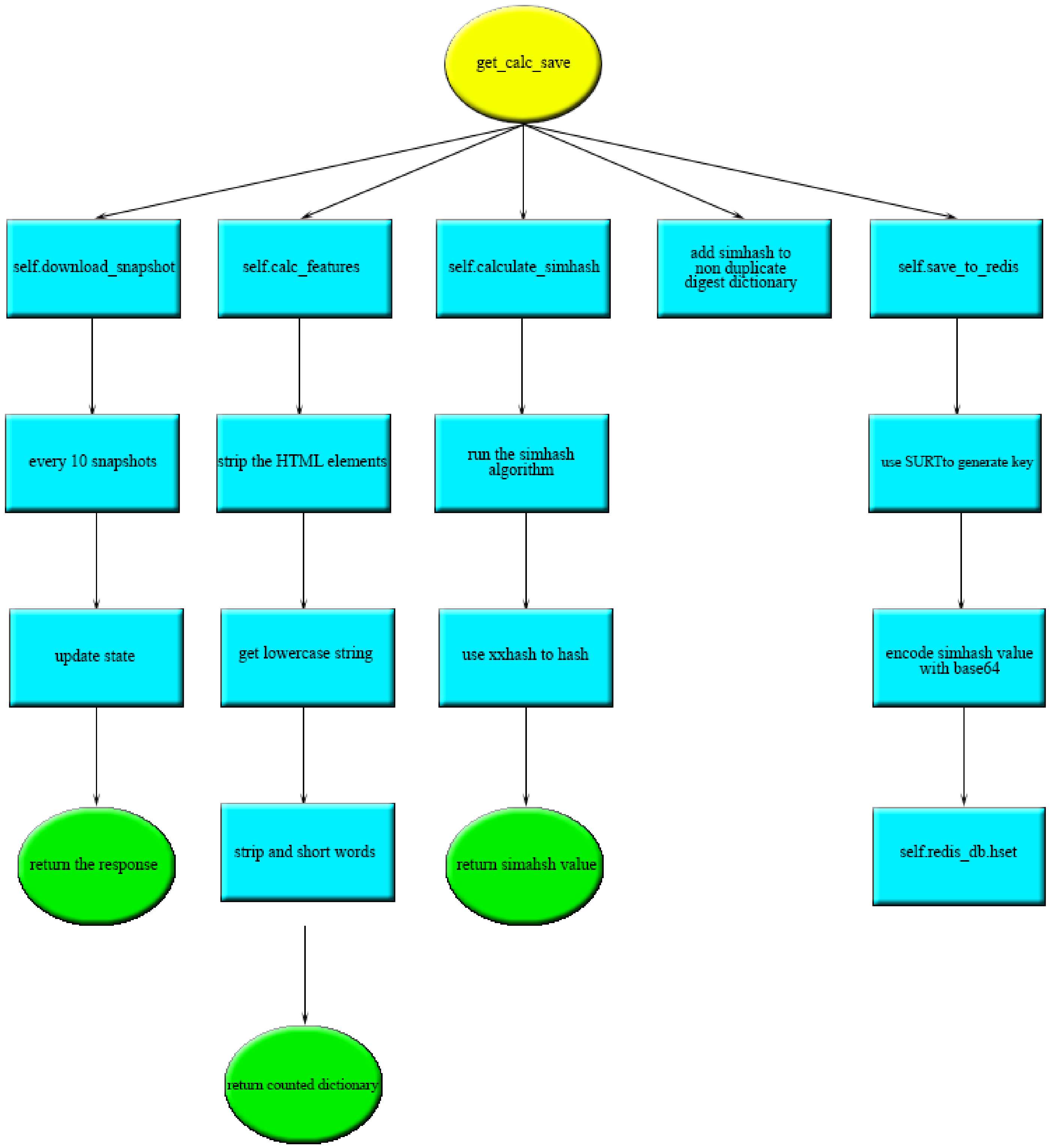
6. Return a status, error or complete, JSON response .

Figure 3.4: Discover Celery task flow chart.


run
get CDX data
JSON decode
init Discover Celery Task object
seperate duplicate snapshots
concurrently using ThreadPoolExecutor
self.get_calc_save for every non-duplicate snapshot
self.save_to_redis for duplicates using digest dictionary
return result status JSON

The **get_calc_save** function uses the **download_snapshot** function to download the snapshot constructed from the URL and timestamp it received as parameters and calculates the features of the downloaded snapshot. After significant research it was apparent that the features that are important to use as data to calculate the simhash value of the snapshot from, are not the HTML document as a whole, but its text. User interface or other changes are not as important as changes to the webpage's content at this time. Keeping this in mind, the downloaded HTML document is stripped and transformed into a dictionary of the words included in its content. This dictionary shows which words and how many times occurred in the snapshot. In turn, this dictionary is passed as input to the **calculate_simhash** function which, using xxhash as a hashing algorithm, returns a simhash value. When the Simahsh value for a snapshot is calculated, it is appended in the digest dictionary so it can be later reused for other snapshots with the same digest value. By doing this, less computing power and time is spent. This is possible because if two or more snapshots have an identical digest value, it means that their content is exactly the same. For this reason it is safe to only once calculate the simhash value for this data and reuse it. The last step of the **get_calc_save** function is to save the calculated simahsh value to the Redis database. The key which the data are inserted under in the database is the SURT value of the webpage. The simhash value is encoded into a base64 value before being saved in the database.

Figure 3.5: The get_calc_save function's flow chart.



The **year_simhash** function, first of all, checks the validity of its parameters. Then it queries the Redis database for the available keys with the SURT value of the requested webpage. If no results are found, the appropriate JSON response is returned. If results are found, the code iterates over the list of results, which in this case are the available timestamps and checks which year they describe. If the year of the timestamp matches the requested year then this timestamp is appended in an array. After iterating through the results, if timestamps that match the requested year were found, the **handle_results** function is called. As parameters it receives the array of the matching timestamps, the requested URL and the number of the requested page, if any.

The **handle_results** function, if pagination is required, breaks the array of timestamps whose simhash values are going to be requested from the database into smaller chunks. For each timestamp in the processed chunk, it fetches its simhash value from the database and it appends the timestamp-simahash pair into an array. Finally, this array is turned into a JSON object and is returned.

The **timestamp_simahsh** function is simpler. After validating its parameters, it queries the database for the value of the SURT value of the webpage and timestamp pair and returns it in JSON format. If no value is found in the database, the appropriate error message is returned.

3.3 Improving the performance of wayback-discover-diff

wayback-discover-diff is designed to handle a lot of requests and as a result a lot of data asynchronously. Doing all of this work in an asynchronous manner does not mean though that the server's resources are unlimited or that a colossal amount of wait time is acceptable. So, the need to improve and optimize the code is immersive. Most of the resource hungry work is done in the applications raised threads. The threads do three things that require a significant amount of time to be executed.

1. They download the snapshot.
2. They calculate the simhash value from the downloaded data.
3. They save the simhash value to Redis.

3.3.1 Downloading the Snapshot

In the first implementation of an algorithm that does those three things the call graph indicated that most of the run-time was spent calculating the simhash (59.73%) value and the rest downloading the data (40.27%) in large (3.1MB) webpages (https://terrytao.wordpress.com). For smaller (1.5MB) webpages (iskme.org) 90.01% of the run-time was spent downloading the data and only 9.97% calculating the simhash.

Figure 3.6: Runtime spent for terrytao.wordpress.com

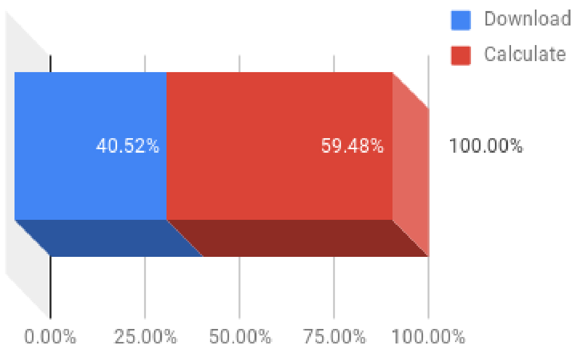


Figure 3.7: Runtime spent for iskme.org

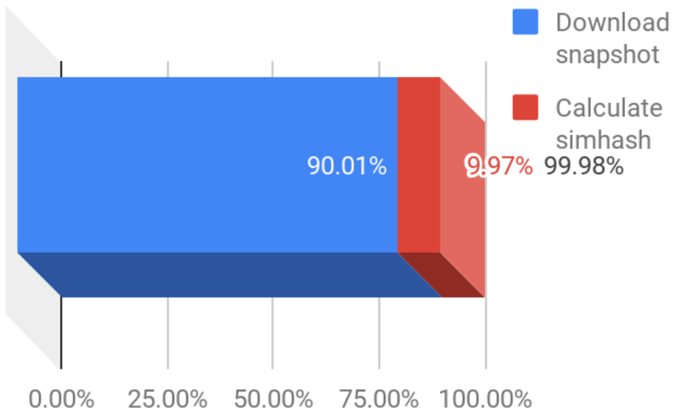


Figure 3.8: Callgraph of wayback-discover-diff for terrytao.wordpress.com

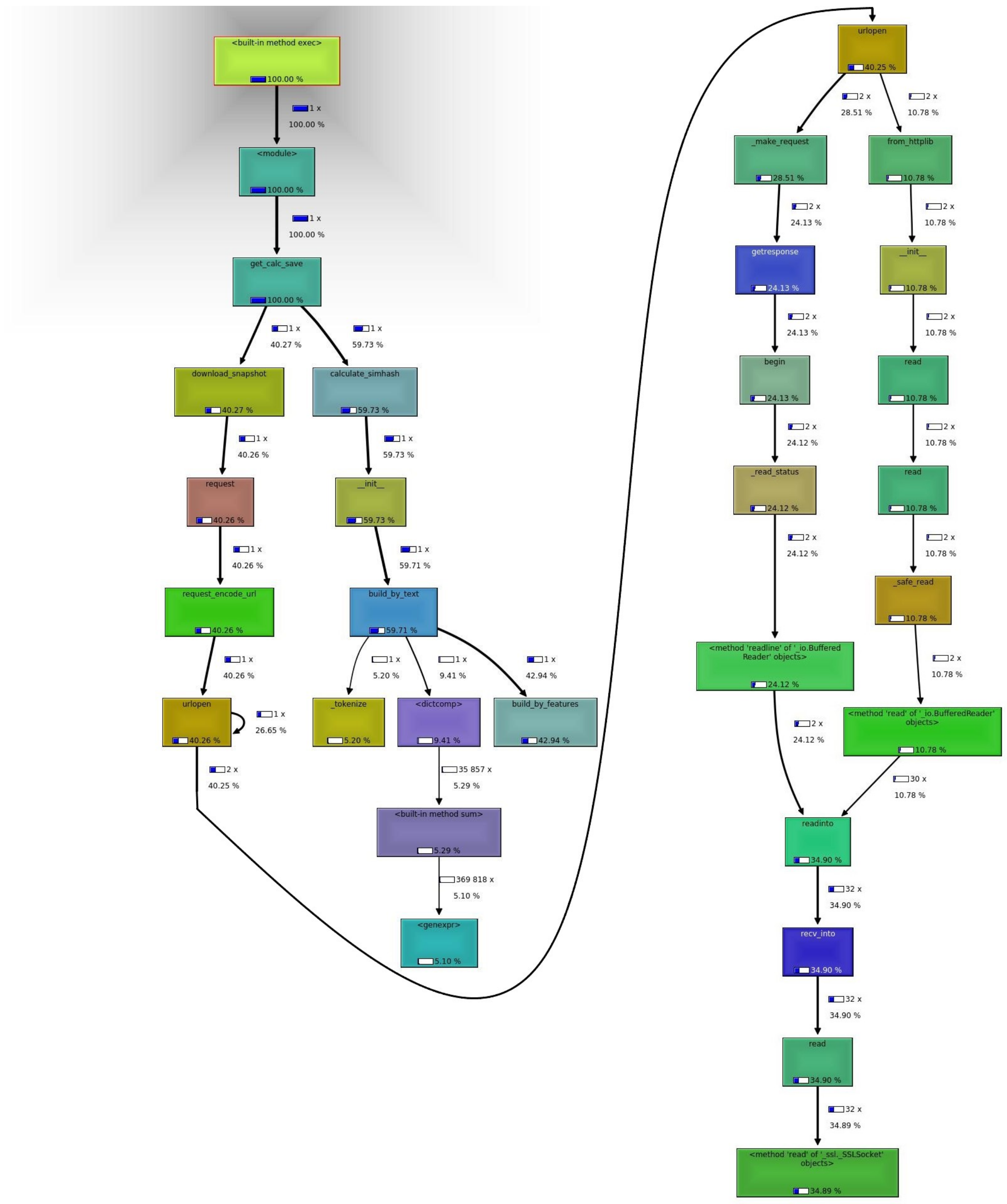
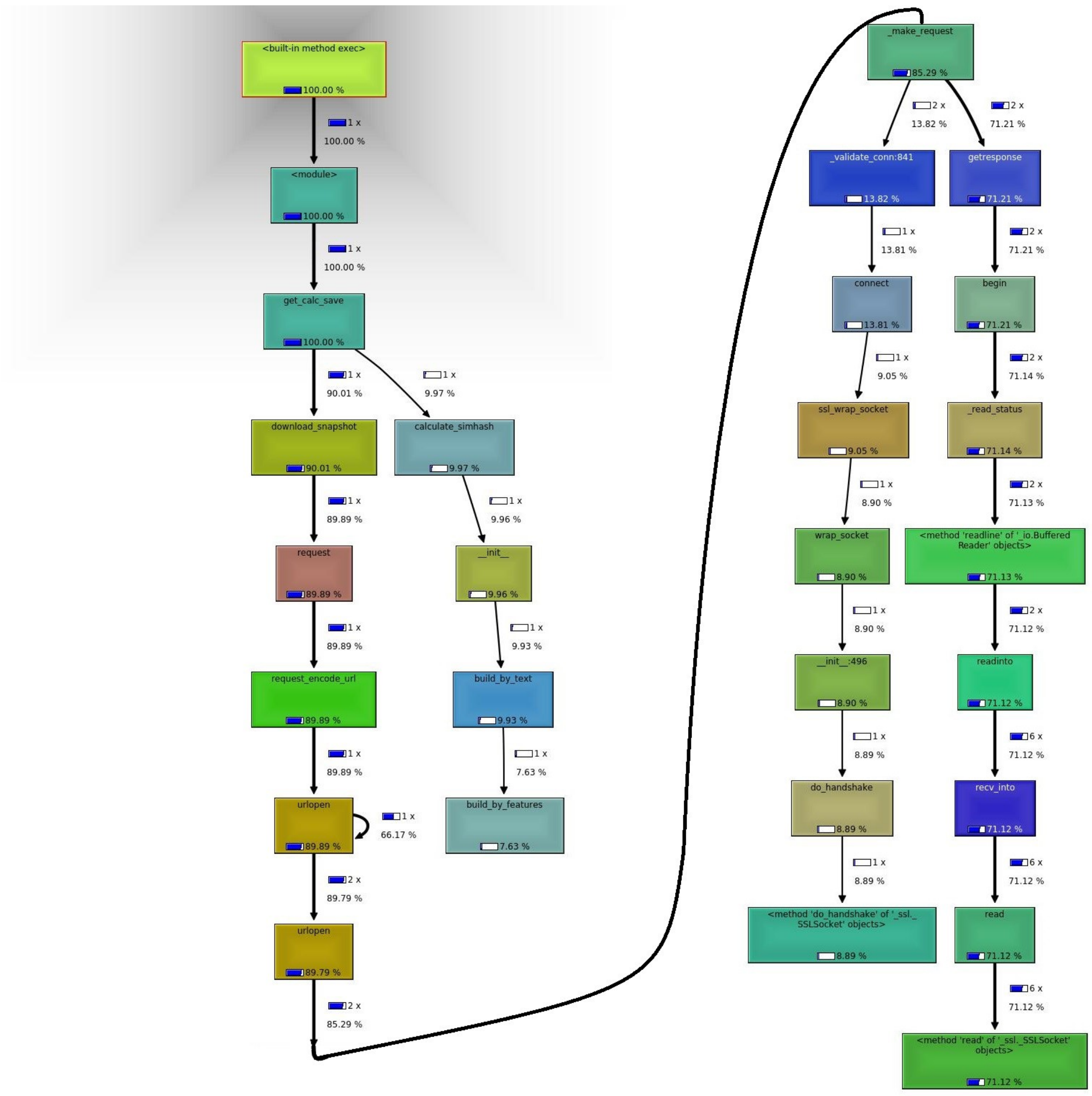


Figure 3.9: Callgraph of wayback-discover-diff for iskme.org



After running multiple experiments the run-time percentages offered insight for both cases. The download procedure needed optimisation. The call graphs made it clear that a large amount of the download time was spent on methods regarding the SSL connection. To be specific, in small webpages 79% of the download time was spent on establishing the SSL connection and only 21% on actually downloading the file. In bigger webpages (https://terrytao.wordpress.com) 74.9% of the download time was spent on SSL procedures.

Requesting the snapshots from the non-secure URL of the Wayback Machine resulted in an important reduction in the time spent downloading them. In more detail, for smaller webpages (iskme.org in this example) the average download time dropped from 324 seconds to 210 seconds, resulting in

Figure 3.10: Runtime spent for the SSL connection for terrytao.wordpress.com

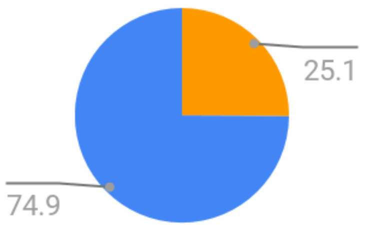
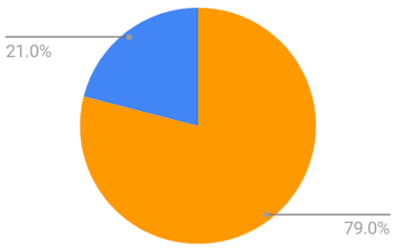
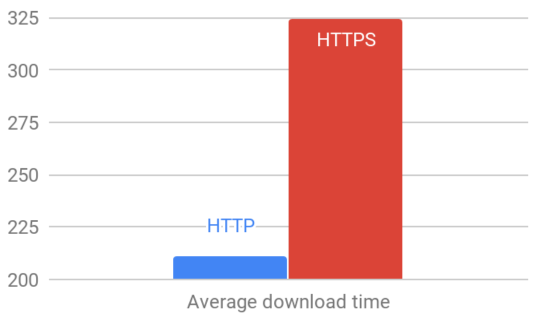


Figure 3.11: Runtime spent for the SSL connection for iskme.org



a 34.94% decline. For larger webpages (https://terrytao.wordpress.com in this example) this change reduced the download time by 43.28%, from 86 seconds to about 49 seconds.

Figure 3.12: Average download time for iskme.org



3.3.2 Hashing algorithm

The next step was to research hashing algorithms and find one that might be faster from the library's default MD5. Changing from the default MD5 to the xxhash (https://pypi.org/project/xxhash/) algorithm improved the run-time by about 23.32% in smaller webpages like iskme.org reducing the mean time that it took to run the build_by_features method from about 67 seconds to almost 51 seconds. In larger webpages (https://terrytao.wordpress.com) the change of the hashing algorithm improved the run-time by about 12.33% dropping the run-time of the build_by_features method from about 106 seconds to almost 93 seconds.

Figure 3.13: Average build_by_features time for iskme.org

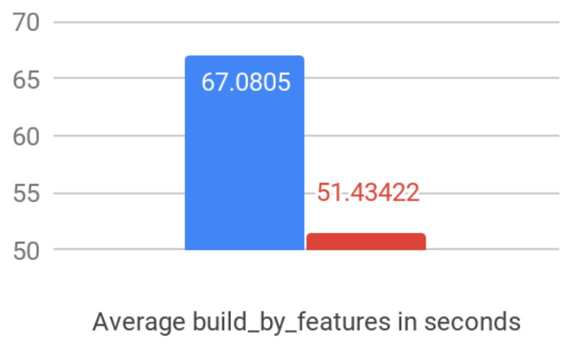
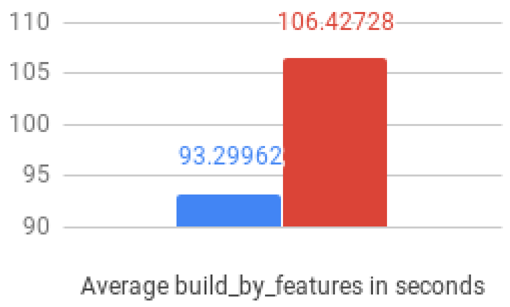


Figure 3.14: Average build_by_features time for terrytao.wordpress.com



3.3.3 Different approach to calculating the features list

Last but not least, taking a closer look into how the Simhash library works it is apparent that when not given a list of features and their respective weights but a String, as had been happening so far, it calculates the features list on its own. More specifically:

1. It lowercases all the characters in the string
2. Breaks the document - string into smaller ones, each string consisting of 4 characters
3. It creates for all the possible combinations by shifting a character at a time.
4. It shorts them
5. It counts how many times each generated string appears
6. It assigns them this frequency as their weight.

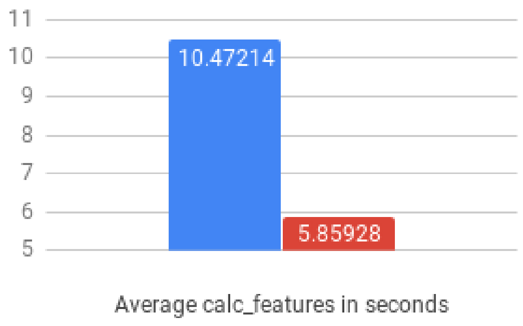
Since documents that contain words are compared, not only there is no need to calculate all the 4-character long combinations of the snapshot's HTML but it is more useful to calculate the features using words as a criteria. The author's implementation of an algorithm that calculates the features list of a given snapshot is the following:

1. Strip down all the HTML markup
2. Turn all the characters to lowercase
3. Create an array of the words in the snapshots
4. Short them
5. Assign the frequency they appear as each words weight.

Then pass this dictionary to the library for it to continue the procedure as it normally would.

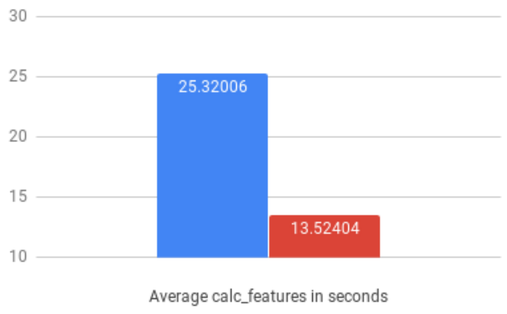
The differentiated algorithm introduced a 44.05% reduction in the time needed to calculate the features list for smaller webpages (iskme.org) dropping from 10.47 seconds to almost 5.85.

Figure 3.15: Average calc_features time for iskme.org



On larger webpages the gain is even bigger, counting to 46.59% faster run-time for the features calculation, dropping from about 25 seconds to about 13.5 seconds.

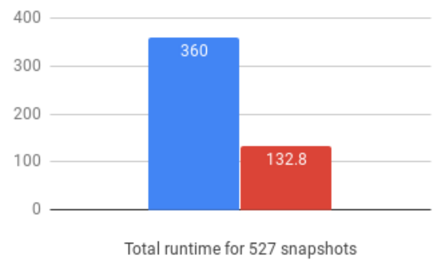
Figure 3.16: Average calc_features time for terrytao.wordpress.com



3.3.4 Overall improvement

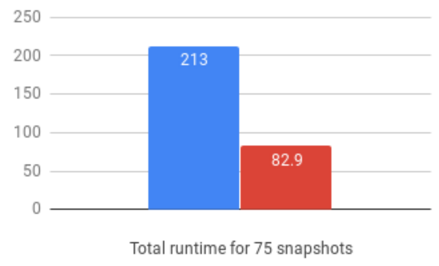
Applying all the above mentioned improvements to the firstly written application has an extraordinary effect on the time needed to fetch each snapshot, calculate its Simhash and store it in the database. For all the 527 snapshots of iskme.org in 2017 the total run-time dropped from 360 seconds to 133 seconds, resulting in 63.11% improvement.

Figure 3.17: Total runtime for 527 snapshots of iskme.org



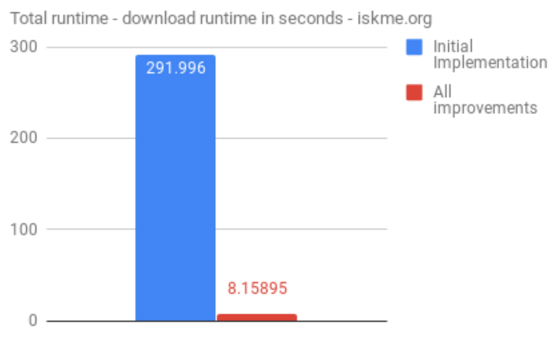
For the 75 snapshots of terrytao.wordpress.com in 2017 the total run-time dropped from 213 seconds to 83 seconds, resulting in 61.08% improvement.

Figure 3.18: Total runtime for 75 snapshots of terrytao.wordpress.com



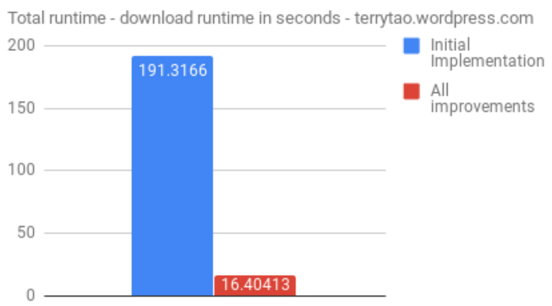
When we take into account that the speed that the snapshots are downloaded from the Wayback Machine is the only think out of this application's control, it is apparent that the improvement in the run-time is bigger than just 63.11% and 61.08% respectively. For iskme.org the total run-time excluding the time needed to download the actual files dropped from 292 seconds to about 8 seconds. This is a 97.21% improvement in the total run-time.

Figure 3.19: Total processing runtime for 527 snapshots of iskme.org



For terrytao.wordpress.com the total run-time excluding the time needed to download the actual files dropped from 191 seconds to about 16 seconds. This is a 91.43% improvement in the total run-time.

Figure 3.20: Total processing runtime for 75 snapshots of terrytao.wordpress.com



Chapter 4

Integrating the deliverables into the Wayback Machine

This chapter is purposed to provide an insight as to how wayback-diff and wayback-discover-diff are integrated into the Wayback Machine platform. Screenshots along with an explanatory text will be the keys to achieving said target.

4.1 wayback-discover-diff

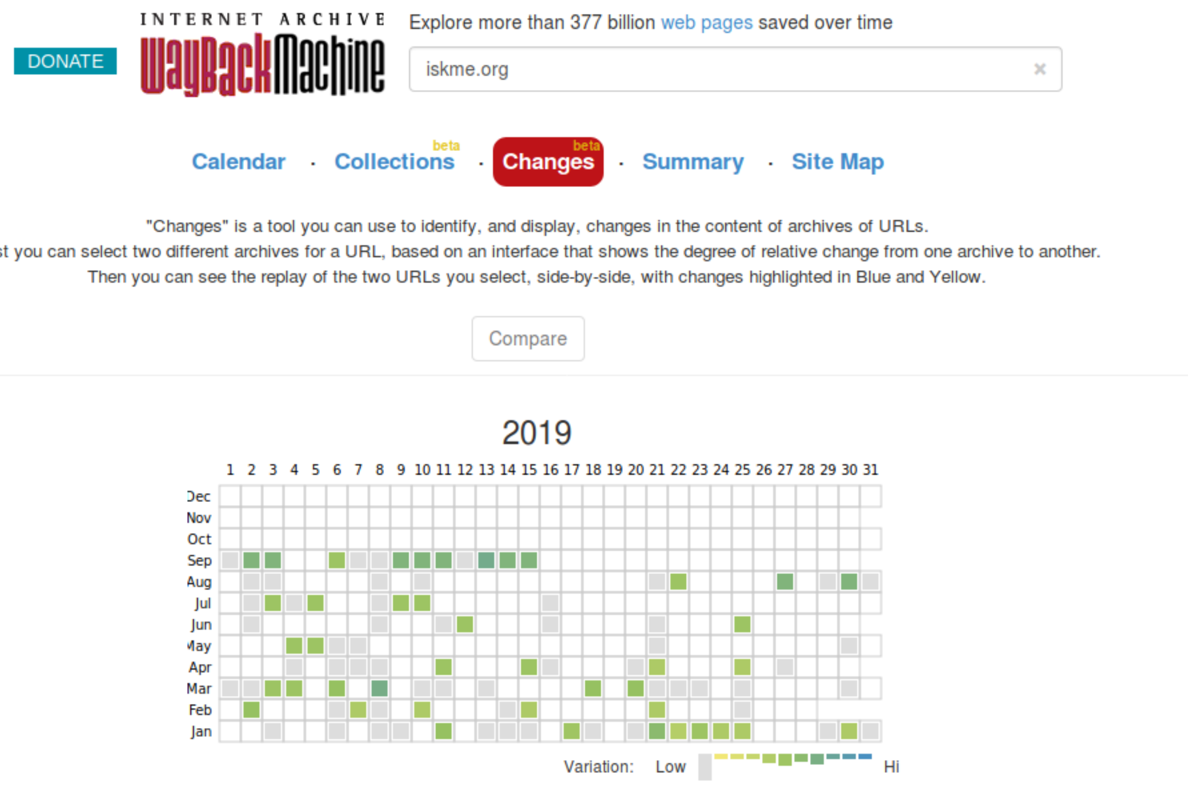
As wayback-discover-diff is a backend application, it does not offer a direct route of interaction with the users even though it provides them with information crucial towards acquiring a profound understanding of how a webpage has changed over time.

The information that wayback-discover-diff provides about a snapshot is, after decoding the base64 value, is a 26-digit number. This is the output of the hashing algorithm on the snapshot after the content has been processed from the algorithm mentioned in chapter 3. This is where the simhash values of the snapshots demonstrate their value. A pair of two simhash values can be used to calculate the hamming distance between them. A service that takes advantage of this opportunity is the Wayback Machine ”Changes” tool.

The Changes tool, using the simhash values provided by the wayback-discover-diff backend presents a heat-map calendar. The heat-map calendar is colored depending on the hamming distance of the two selected snapshots.

The aim of this tool is to provide users with a visual representation of the amount of changes that were made between one specific snapshot and all the others available. For example, for iskme.org the initial screen for 2019 looks as follows.

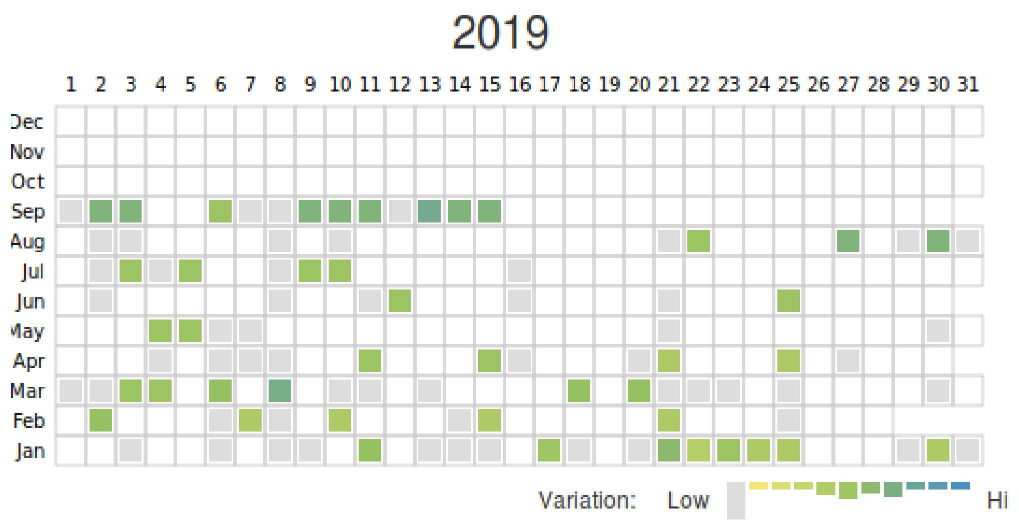
Figure 4.1: The main page of the Changes tool in the Wayback Machine.


DONATE
INTERNET ARCHIVE
WaybackMachine
Explore more than 377 billion web pages saved over time
iskme.org
Calendar · Collections beta · Changes beta · Summary · Site Map

"Changes" is a tool you can use to identify, and display, changes in the content of archives of URLs.
First you can select two different archives for a URL, based on an interface that shows the degree of relative change from one archive to another.
Then you can see the replay of the two URLs you select, side-by-side, with changes highlighted in Blue and Yellow.

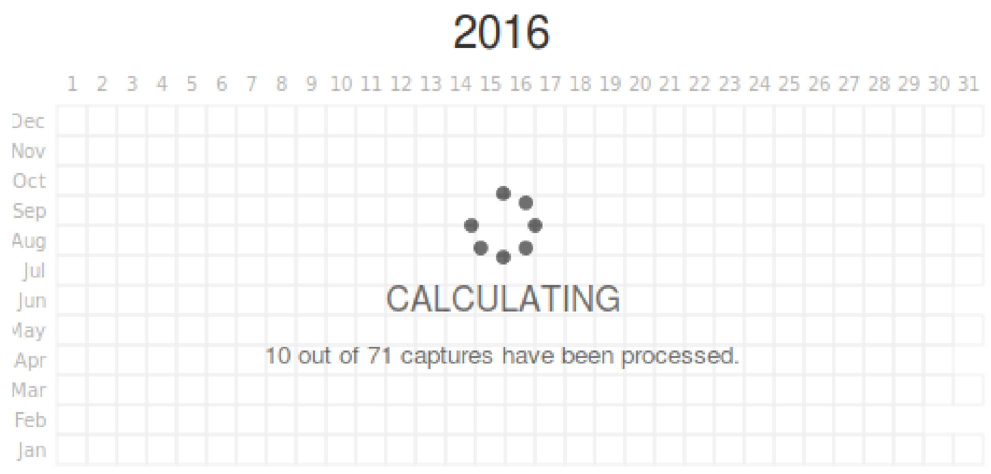
Compare
2019
Variation: Low Hi

Figure 4.2: The Changes tool for iskme.org for 2019 in the Wayback Machine.



Of course if the simhash information is not available, a request is sent to wayback-discover-diff to generate it. This tool uses the *job status* endpoint of wayback-discover-diff to provide updates on the simahash calculation progress.

Figure 4.3: The Changes tool showing the simhash calculation progress as reported from wayback-discover-diff.

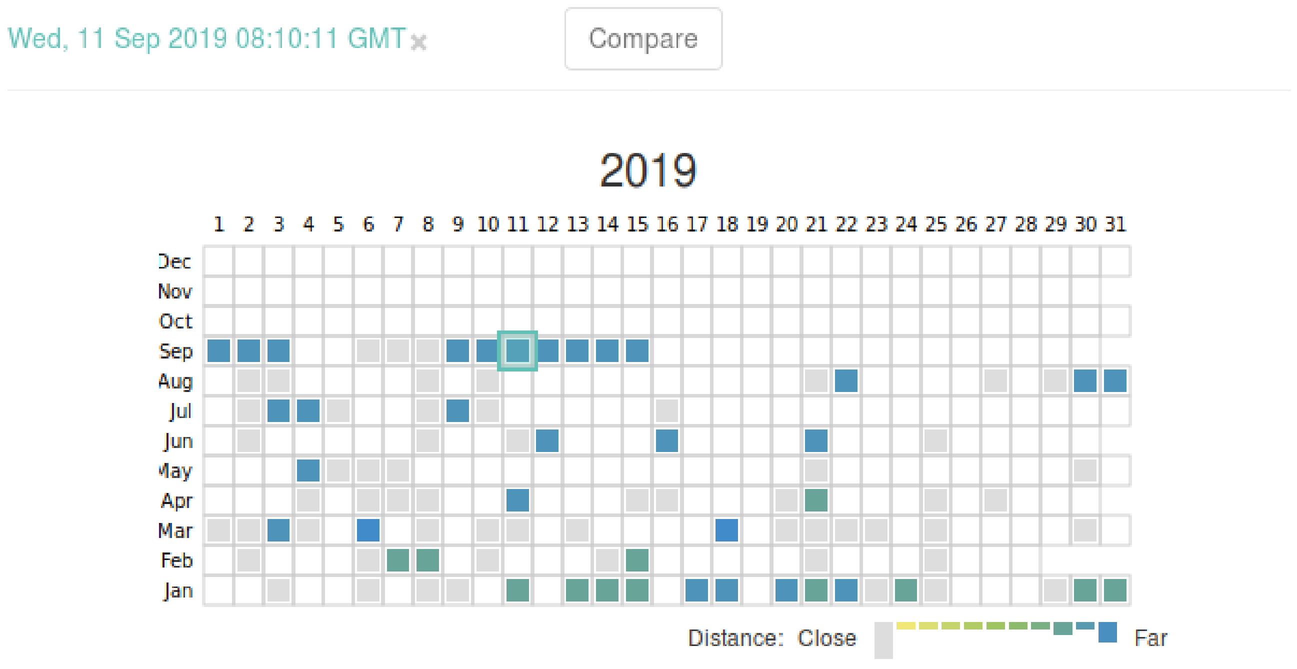


CALCULATING

10 out of 71 captures have been processed.

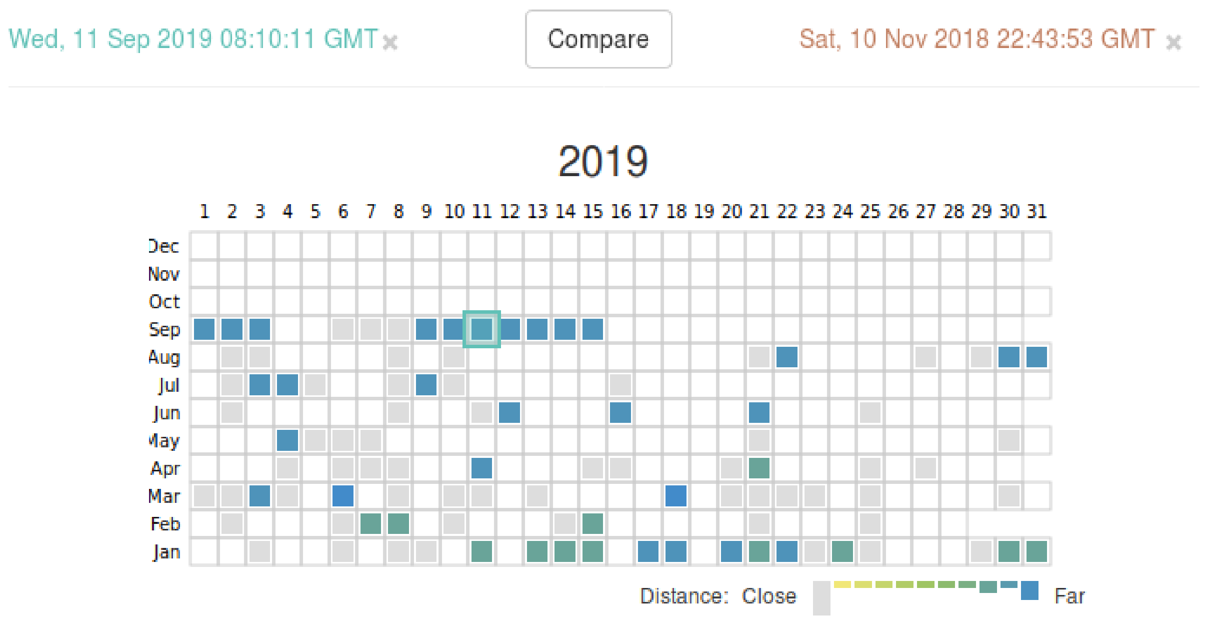
As soon as the user selects a snapshot, the color-coding of the heatmap changes. All the distances are calculated from the simhash value of the selected snapshot.

Figure 4.4: The Changes tool for iskme.org for a selected date in 2019 in the Wayback Machine.



When the user selects a second snapshot from the color coded calendar, the option to compare those snapshot appears on the top of the page as indicated in the screenshot bellow. Clicking on the compare button opens a new tab in the user's browser that compares the selected snapshots using wayback-diff.

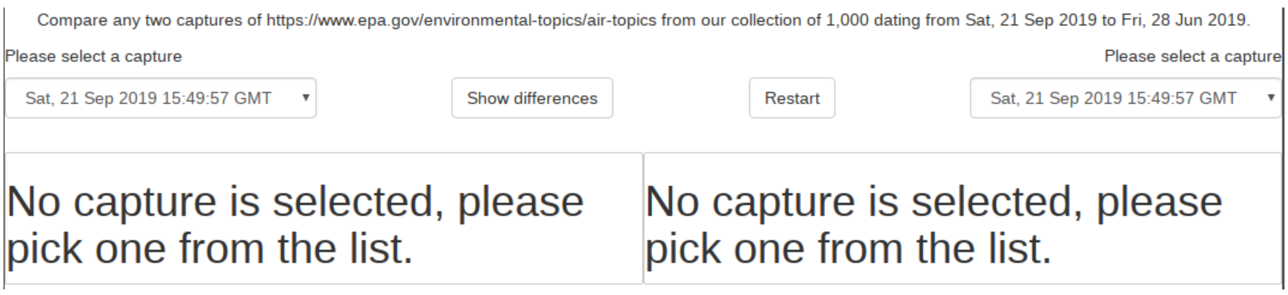
Figure 4.5: The Changes tool for iskme.org for two selected dates in the Wayback Machine.



4.2 wayback-diff

The initial screen of wayback-diff aims to provide as much information as possible about the available snapshots to the user without overwhelming them. Being called from the *http://localhost:3000/diff/WEBSITE* URL schema, it fetches and shows two dropdown boxes that allow users to select the snapshots they want to compare.

Figure 4.6: The initial wayback-diff user interface.


Compare any two captures of https://www.epa.gov/environmental-topics/air-topics from our collection of 1,000 dating from Sat, 21 Sep 2019 to Fri, 28 Jun 2019.

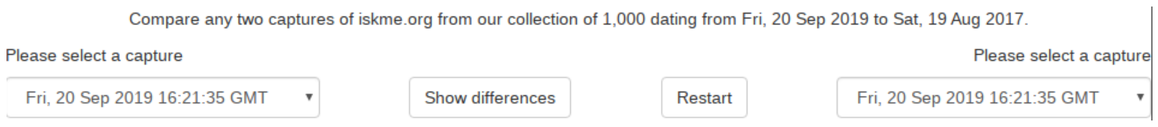
Please select a capture — Please select a capture

No capture is selected, please pick one from the list.

No capture is selected, please pick one from the list.

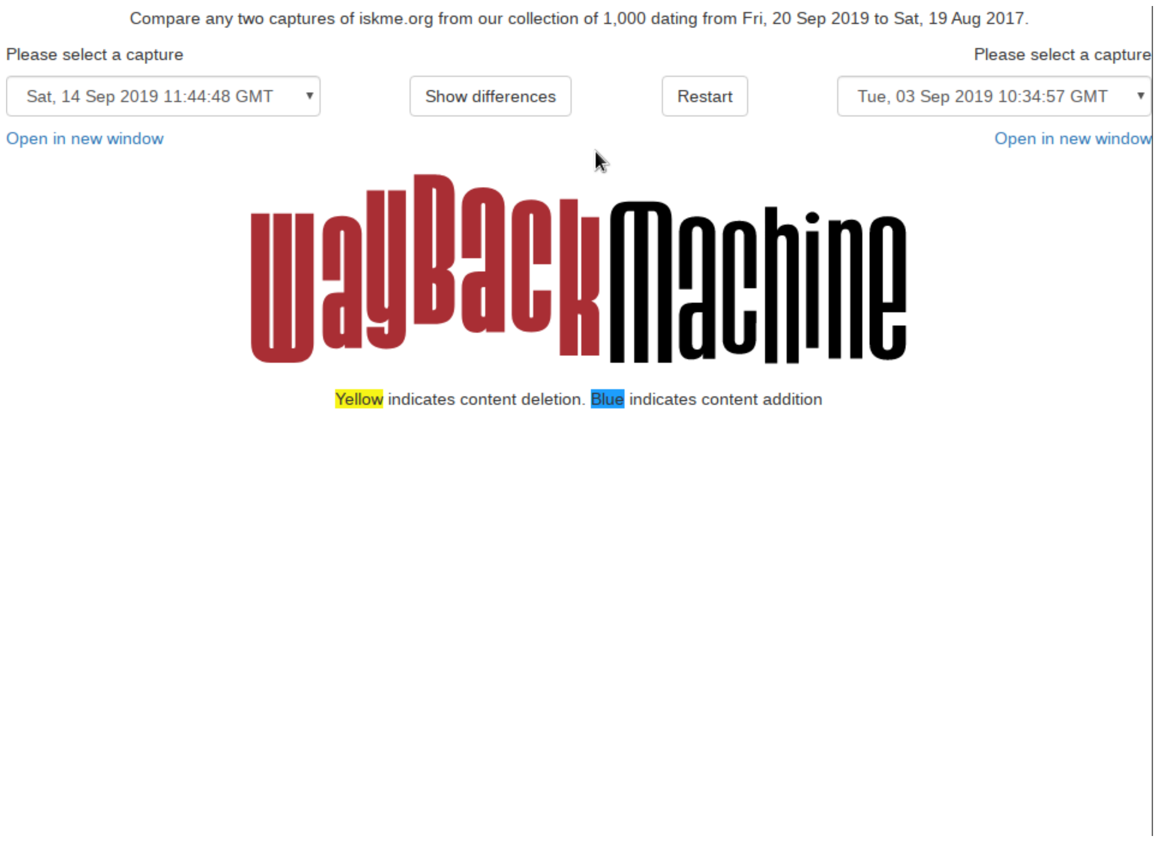
The timestamps of the available snapshots are shorted by year and are filtered using the snapshots' digest value, helping users make more meaningful comparisons.

Figure 4.7: The dropdown menu in wayback-diff user interface.


Compare any two captures of iskme.org from our collection of 1,000 dating from Fri, 20 Sep 2019 to Sat, 19 Aug 2017.

As soon as two snapshots are selected and the user clicks on the ”Show Differences” button, the loader appears letting the user know that the snapshots are being compared in real-time.

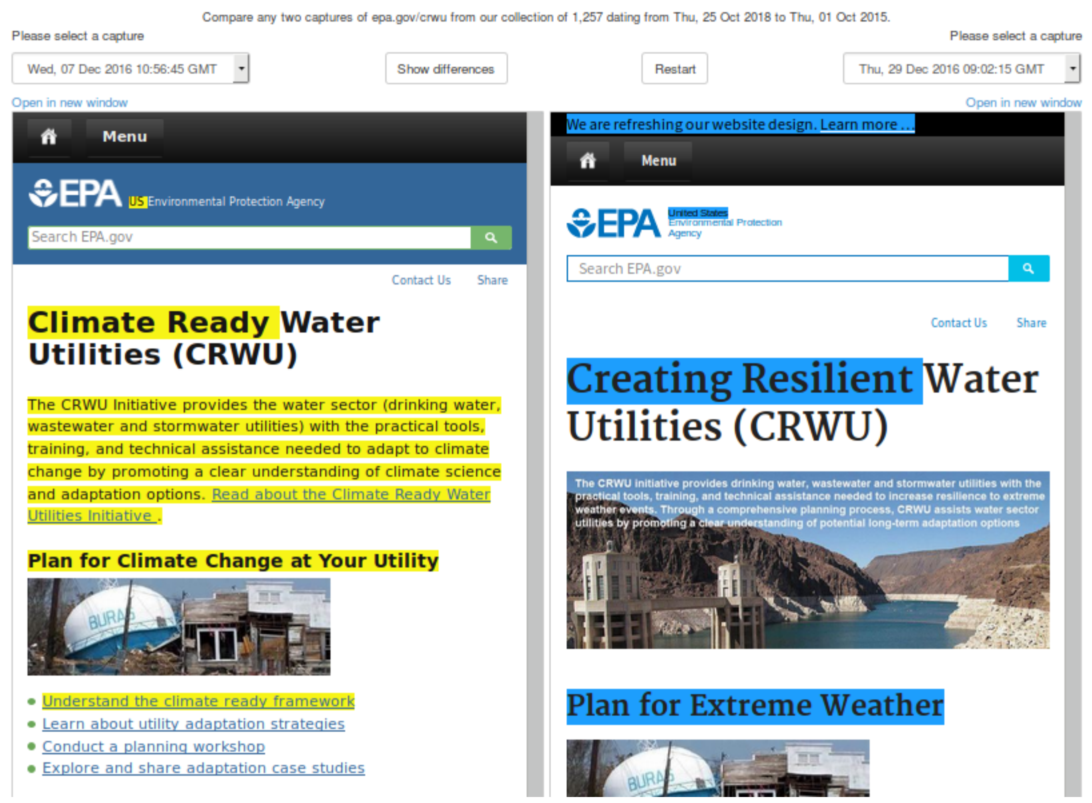
Figure 4.8: The loading wayback-diff user interface.


Compare any two captures of iskme.org from our collection of 1,000 dating from Fri, 20 Sep 2019 to Sat, 19 Aug 2017.

Yellow indicates content deletion. Blue indicates content addition

When the loading is complete, meaning that the two snapshots were downloaded, compared and returned to wayback-diff by the *web-monitoring-processing* application, they appear side-by-side inside two iFrames.

Figure 4.9: The comparison of two snapshots using wayback-diff.


Compare any two captures of epa.gov/crwu from our collection of 1,257 dating from Thu, 25 Oct 2018 to Thu, 01 Oct 2015.

Please select a capture: Wed, 07 Dec 2016 10:56:45 GMT

Please select a capture: Thu, 29 Dec 2016 09:02:15 GMT

We are refreshing our website design. Learn more ...

Climate Ready Water Utilities (CRWU)

The CRWU Initiative provides the water sector (drinking water, wastewater and stormwater utilities) with the practical tools, training, and technical assistance needed to adapt to climate change by promoting a clear understanding of climate science and adaptation options. Read about the Climate Ready Water Utilities Initiative .

Plan for Climate Change at Your Utility

- Understand the climate ready framework
- Learn about utility adaptation strategies
- Conduct a planning workshop
- Explore and share adaptation case studies

Creating Resilient Water Utilities (CRWU)

The CRWU initiative provides drinking water, wastewater and stormwater utilities with the practical tools, training, and technical assistance needed to increase resilience to extreme weather events. Through a comprehensive planning process, CRWU assists water sector utilities by promoting a clear understanding of potential long-term adaptation options

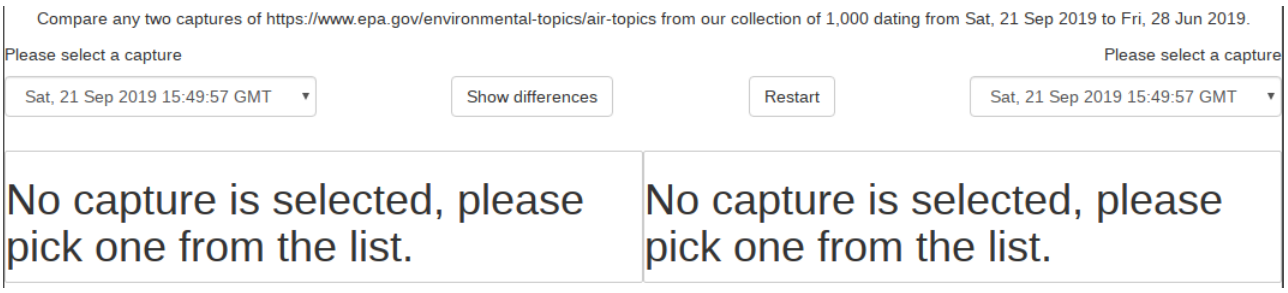
Plan for Extreme Weather

A case exists where a timestamp might be missing from either, or both of the positions in the URL schema, meaning that the URL has one of the three following formats:

1. http://localhost:3000/diff/TIMESTAMP_A//WEBSITE
2. http://localhost:3000/diff//TIMESTAMP_B/WEBSITE
3. http://localhost:3000/diff///WEBSITE

This case is handled by showing a placeholder text in the correct position on the screen, informing the user of the mistake.

Figure 4.10: The wayback-diff user interface with the both timestamps missing from the URL.


Compare any two captures of https://www.epa.gov/environmental-topics/air-topics from our collection of 1,000 dating from Sat, 21 Sep 2019 to Fri, 28 Jun 2019.

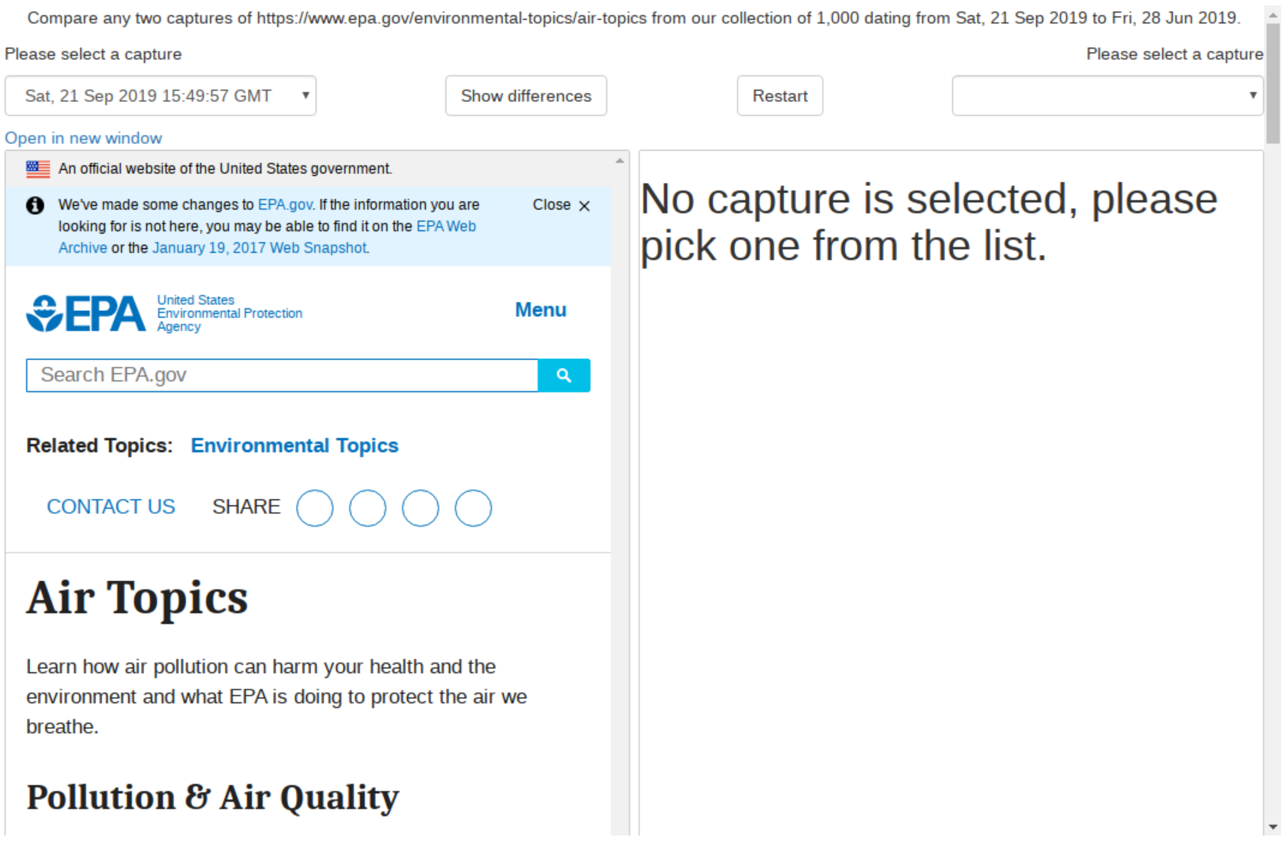
Please select a capture: Sat, 21 Sep 2019 15:49:57 GMT

Please select a capture: Sat, 21 Sep 2019 15:49:57 GMT

No capture is selected, please pick one from the list.

No capture is selected, please pick one from the list.

Figure 4.11: The wayback-diff user interface with the right timestamp missing from the URL.



Compare any two captures of https://www.epa.gov/environmental-topics/air-topics from our collection of 1,000 dating from Sat, 21 Sep 2019 to Fri, 28 Jun 2019.

Please select a capture

Sat, 21 Sep 2019 15:49:57 GMT

Please select a capture

Open in new window

An official website of the United States government.

We've made some changes to EPA.gov. If the information you are looking for is not here, you may be able to find it on the EPA Web Archive or the January 19, 2017 Web Snapshot.

Air Topics

Learn how air pollution can harm your health and the environment and what EPA is doing to protect the air we breathe.

Pollution & Air Quality

No capture is selected, please pick one from the list.

Show differences
Restart
Close ×
EPA United States Environmental Protection Agency
Menu
Search EPA.gov
Related Topics: Environmental Topics
CONTACT US
SHARE

In the last picture the filtering of the timestamps using the snapshots' digest value becomes apparent. Since the timestamp of the 21st September 2019 15:49:57 GMT is selected on the left dropdown menu, the snapshots that have the same digest value are filtered out from the right dropdpwn menu. So the timestamp of that snapshot does not appear on the right dropdown. It is important to note that if a webpage has not changed over a period of time (e.g. a month), all of the snapshots in tat period of time will have the same digest value and thus will be hidden. To show them again and be able to select them, the user can click on the *Reset* button, which reverts any filtering made to the snapshots.

Chapter 5

Conclusion and future improvements.

This chapter discuses whether the project's goals where met, it emphasizes on the main points of the author's work and it offers recommendations on future work that can be done to improve the project.

The aim of this project was to integrate the web-monitoring-processing software into the Wayback Machine. This required the modification of the web-monitoring-processing software and the development of wayback-diff. As of October 2018, a beta version of these applications has been running along the Wayback Machine. Thus, it is safe to say that the project's goals have not only been met but they were also exceeded. To support this statement readers can be referred to the development of the wayback-discover-diff application. Wayback-discover-diff can also be assumed an added-value-service to the Wayback Machine. One can assume so because it offers the opportunity to add important characteristics to the already saved snapshots of the Wayback Machine.

Wayback-discover-diff, also implemented for this project is, the moment this Thesis is written, live on Internet Archive's servers providing data to tools like the Wayback Machine's *Changes*.

The scope of this Thesis is the project as it was completed during the Google Summer of Code 2018 and in accordance with that the code discussed in this Thesis was developed up until October 2018. Since then further work has been done on the project. Improvements that have been developed since then or that should be developed are around the three main components of the project.

To begin with, The timestamp component up until October 2018 was not very efficient. The addition of the limit parameter to the CDX queries was required because more often than not webpages with a lot of snapshots n a year would generate a CDX response of significant size. The size of the CDX reply was found to be great enough to cause delays in the users' browsers. A way to avoid this issue would be to split the available snapshots into groups. These groups can be "Year", "Month", "Week",

”Day”. These groups can be fetched with different requests to the CDX server and would result in smaller CDX replies.

Another important improvement would be to permanently save the Simhash values of the snapshots. Appending them in a data source like each snapshot’s CDX entry would fix two issues. First and foremost since the Simahashes are not going to be kept in a Redis database size which would continuously increase in size, thus rendering the need for its entries to expire meaningless. This would also have as a result the significant increase of the response time when a Simhash value is looked up. If it has been calculated once it will never expire and the user will not have to wait for it to be generated again.

Lastly, it should be investigated whether using a ProcessPoolExecutor instead of a ThreadPoolExecutor would help speed the Simhash calculation process up.

Bibliography

[1] *3 strategies to customise celery logging handlers.*
https://www.distributedpython.com/2018/08/28/celery-logging/.

[2] *About the apache http server project.*
http://httpd.apache.org/ABOUT_APACHE.html.

[3] *Advantages of using python over other languages.*
https://www.cleveroad.com/blog/python-vs-other-programming-languages.

[4] *aiohttp 3.6.1 documentation.*
https://aiohttp.readthedocs.io/en/stable/.

[5] *Anaconda | the world's most popular data science platform.*
https://www.anaconda.com/.

[6] *Anaconda vs. miniconda vs. virtualenv.*
http://deeplearning.lipingyang.org/2018/12/23/
anaconda-vs-miniconda-vs-virtualenv/.

[7] *A brief overview of react router and client-side routing.*
https://medium.com/@marcellamaki/a-brief-overview-of-react-router-and-cli

[8] *Conda — conda documentation.*
https://conda.io/en/latest/.

[9] *The definition of iframe defined and explained in simple language.*
https://techterms.com/definition/iframe.

[10] *Design — gunicorn 19.9.0 documentation.*
http://docs.gunicorn.org/en/latest/design.html.

[11] *Edgi: Web monitoring project.*
https://github.com/edgi-govdata-archiving/web-monitoring.

[12] *Fullstack react: What is react?*
https://www.fullstackreact.com/30-days-of-react/day-1/.

[13] *Google code-in.*
https://codein.withgoogle.com/about/.

[14] *Google summer of code.*
https://summerofcode.withgoogle.com/archive/.

[15] *Google summer of code 2018 final report - wayback-discover-diff.*
https://github.com/ftsalamp/gsoc2018-InternetArchive#wayback-discover-diff.

[16] *Google summer of code 2018 final report - wayback-discover-diff.*
https://github.com/ftsalamp/gsoc2018-InternetArchive#wayback-discover-diff.

[17] *Google summer of code blog: So what is this community bonding all about?*
https://googlesummerofcode.blogspot.com/2007/04/so-what-is-this-community-bonding-all.html.

[18] *Google summer of code timeline.*
https://developers.google.com/open-source/gsoc/timeline.

[19] *Higher order functions (composing software).*
https://medium.com/javascript-scene/higher-order-functions-composing-

[20] *An intro to threading in python.*
https://realpython.com/intro-to-python-threading/.

[21] *Introduction to celery.*
http://docs.celeryproject.org/en/latest/getting-started/introduction.html.

[22] *Introduction to lodash.*
https://medium.com/front-end-weekly/introduction-to-lodash-71dbee093b

[23] *Javascript: What the heck is a callback?*
https://codeburst.io/javascript-what-the-heck-is-a-callback-aba4da2de

[24] *Jinja.*
http://jinja.pocoo.org/.

[25] *Lodash documentation.*
https://lodash.com/docs/4.17.11.

[26] *Monitoring and management guide — celery 4.3.0 documentation.*
https://docs.celeryproject.org/en/latest/userguide/monitoring.html#management-command-line-utilities-inspect-control.

[27] *No license.*
https://choosealicense.com/no-permission/.

[28] *Pep 3333 – python web server gateway interface v1.0.1.*
https://www.python.org/dev/peps/pep-3333/#original-rationale-and-goals-from-pep-333.

[29] *pip - the python package installer.*
https://pip.pypa.io/en/stable/.

[30] *Props.*
https://facebook.github.io/react-native/docs/props.

[31] *Python virtual environments: A primer.*
https://realpython.com/python-virtual-environments-a-primer/#why-the-need-for-virtual-environments.

[32] *Rendering to iframes in react.*
https://medium.com/@ryanseddon/rendering-to-iframes-in-react-d1cb92274f86

[33] *Stateless component vs pure component.*
https://medium.com/groww-engineering/stateless-component-vs-pure-componen

[34] *Typechecking with proptypes.*
https://reactjs.org/docs/typechecking-with-proptypes.html.

[35] *Understand javascript callback functions and use them.*
https://javascriptissexy.com/understand-javascript-callback-functions-and

[36] *Understanding react components.*
https://medium.com/the-andela-way/understanding-react-components-37f841c1

[37] *Understanding "state" in react components.*
https://thinkster.io/tutorials/understanding-react-state.

[38] *urllib3 1.25.3 documentation.*
https://urllib3.readthedocs.io/en/latest/.

[39] *Using jsx via babel.*
https://www.reactenlightenment.com/react-basic-setup/3.2.html.

[40] *A visual guide to state in react.*
`https://daveceddia.com/visual-guide-to-state-in-react/`.

[41] *Web server gateway interface.*
`https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface`.

[42] *What is gunicorn, and what does it do?*
`https://vsupalov.com/what-is-gunicorn/`.

[43] *What is iframe (inline frame)? - definition from whatis.com.*
`https://whatis.techtarget.com/definition/IFrame-Inline-Frame`.

[44] *What is jsx?*
`https://www.reactenlightenment.com/react-jsx/5.1.html`.

[45] *What is npm?*
`https://www.npmjs.com/what-is-npm`.

[46] *What is python? executive summary.*
`https://www.python.org/doc/essays/blurb/`.

[47] *What is reactjs and why should we use reactjs?*
`https://www.c-sharpcorner.com/article/what-and-why-reactjs/`.

[48] *What is the python global interpreter lock (gil)?*
`https://realpython.com/python-gil/`.

[49] *Why you need python environments and how to manage them with conda.*
`https://medium.freecodecamp.org/why-you-need-python-environments-and-`

[50] *Wsgi servers.*
`https://www.fullstackpython.com/wsgi-servers.html`.

[51] *Yaml ain't markup language (yaml™)version 1.23rd edition, patched at 2009-10-01.*
`https://yaml.org/spec/cvs/spec.pdf`.

[52] *Yarn: A new package manager for javascript.*
`https://code.fb.com/web/yarn-a-new-package-manager-for-javascript/`.

[53] *Yet another markup language (yaml) 1.0.*
`https://yaml.org/spec/history/2001-12-10.html`.

[54] B. Balter, *Open source license usage on github.com.*
`https://github.com/blog/1964-open-source-license-usage-on-github-com`, Mar 2015.

[55] P. M. J. Levine, Sheen S., *Open collaboration for innovation: Principles and performance.* `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1096442`, 2013.

[56] S. S. Levine and M. J. Prietula, *Open collaboration for innovation: Principles and performance*, Organization Science, 25 (2014), p. 1414–1433.