

Design Studio 3



Subject	Design Studio 3	
Class	Informatics 121: Software Design 1	
Professor	André van der Hoek	
Date	December 1st, 2019	
Team	Session #2, Team 7	
Members	Name	Name
	Sazeda Sultana (ID#18592717)	Chris Baro (ID#33371425)
	Mohra Arsala (ID#29658422)	Malcolm Treacy (ID#74514735)
	Glen Wu (ID#32928250)	

Table of Contents

Customer Prompt	3
Audience	3
Other Stakeholders	7
Goals	7
Constraints	8
Assumptions	9
Ideas	9
Application Design	9
Interaction Design	11
Architecture Design	19

Customer Prompt

History!

You are tasked with designing a new piece of software that captures the online history of web sites. The idea is to build an archive that allows user to go back to previous versions of any of the web sites of which the history is being captured, and to see what might be different at different points in time.

The company believes it has a brilliant idea that lots of companies will want to use. Universities may want it to understand how the catalog has changed; law firms may want to use it to capture what laws were application when; businesses may want to capture their institutional history and how they portrayed themselves externally; and even individuals and families may want to capture their online historical presence for future grandkids and more. The company knows that creating this software may be very difficult; so it is looking at you to figure out what may or may not be possible.

The company has long built a reputation of its existing software, thought completely unrelated to this idea, being super usable. They would want that to be the case for the new software as well -- they view it as their key competitive advantage.

The company, however, is not versed in software design. It is hiring you to design their new and hopefully revolutionary system.

Audience

Project Sponsor (Customer requesting History!)

One of the team's primary audience members is the company that has hired us to design their software. They are a company known for having "super usable" software and would like the software the team designs to also be super usable. Inside the company, the following individuals and organizations will be involved in the software's success and daily operation.

Company Executives: The company executives will have broad control over the short term, long term, and alternative goals as it relates to the software being developed. The corporate team must be satisfied with the teams designs, and will be in charge of final project approval. While not directly involved in the technical implementation they will have broad oversight over all aspects of the project. This can include but it not limited to influences on: UX/UI, technical details, financial details, and policy implementation. All these factors could potentially affect the project.

Project Managers: Company project managers involved in the day-to-day oversight of the *History!* Software design team. The company's project managers will be directly interacting with our team to ensure the project is on track and following their specifications. The project manager is also the main avenue for corporate executives to communicate their comments or concerns regarding the project. Project managers will be the team's main point of contact with the team.

Developer Team: After final project handoff and approval, the customer's developer team will be in charge of taking over day-to-day technical development of the project. This can include but is not limited to: software updates, bug fixes, and promotional changes to the code. They will need to be knowledgeable of the architecture and implementation of the *History!* Project.

DevOps Team: The customer's DevOps Team will be in charge of the day-to-day operation of the servers and hardware ecosystem powering the project. This includes maintaining storage, request servers, and other kinds of support servers used to power the *History!* software.

Accounting Team: The customer's accounting team will oversee the project's financial success. Their role as it relates to the *History!* Project is to set and implement the projects pricing scheme and any kind of promotional details as it relates to the project.

Customer Service and Support Team: The company's Customer Service and Support Team will be in charge of interacting with customers of *History!* as well as tasked with resolving any issues related to the software. They will need to be intimately familiar with how the software works and potential fixes to customer concerns and complaints.

Marketing Team: The company that has hired our team will also have a person or persons involved in the marketing of *History!* They will need to be familiar with how the software works, and all of its features. They will be working closely with the accounting team in situating the project in a manner that will give it the best chance of customer adoption and success.

Public Institutional Entities

Institutional entities are broadly defined as any kind of government organization that might have an interest in tracking the historical variation of their government-affiliated web presences.

History! has the potential to transform how government archives currently track website changes. The size, scope, and type of institutional entity that might be interested in the software is large and varied. The following are some of the core potential institutional users:

City Government Entities: Many city governments have an interest in tracking the historical events, changes, and challenges facing the community. Lots of communication occurs through city-run websites, and they may be interested in tracking and cataloging these kinds of events. Events can be broadly understood as emergency events that may have affected the community, community festivals and parades, and changes to city policies. Many cities and other kinds of government organizations must also follow government transparency guidelines and comply FOIA (Freedom of Information Act) requests.

County Government Entities: County government organizations have many overlapping authorities and functions with city government organization, but also have many important functions of their own. County government organizations are involved in the governance of agricultural operations, transportation development, land surveying and mapping, and other high-level government operations. *History!* can help in tracking historical changes and events as it relates to these departments.

State Government Entities: State governments and their interrelated governmental authorities have a vested interest in tracking important information that affects the

citizens in many different ways. Large-scale health policies and programs are implemented on the state level, as well as environmental regulation and oversight, transportation guidelines, and changes to existing law. All these kinds of high-level government programs need to be documented and categorized to improve transparency and promote community participation.

Public Utility Companies: Utility companies often change policies, implement community programs, and make changes to their services. Our software can track these changes and help utilities track what changes and policy directions are more effective than others. This gives utilities an important historical perspective on how to structure their services and better support their community.

School District Entities: Policy changes can affect how school districts operate from year to year. This can include changes to school boundaries, changes in school leadership and personnel, as well as historical events (sports/academic championships). Capturing these kinds of events is an important exercise, and our software can help how districts would be able to track these kinds of important events.

College District Entities: College districts are tasked with helping their community members reach their career and academic goals. Historically speaking there are constant changes to technical programs such as HVAC, Nursing, and other kinds of certifications. They must also support students in achieving transfer requirements for students wishing to pursue advanced degrees. By tracking these changes students, faculty, and staff will be able to better understand how to achieve their goals and minimize confusion.

Public University Entities: Large public universities have hundreds of degree options, thousands of employees, and several dozen departments. These are large organizations that often have long and storied histories. Our team's software would allow these kinds of institutions to track and archive these kinds of historical changes.

Corporate Entities

Another important potential audience member can be broadly categorized as corporate entities. These are non-governmental organizations that have significant online presences. They are involved in a wide variety of industries and could use the historical perspective that *History!* can offer. These kinds of corporate users are concerned with leveraging a historical perspective with performance metrics to improve outcomes.

Movie Studios: Movie studios spend billions on marketing new films. A historical perspective of their online marketing presence could potentially help them make better marketing decisions. There is also a rich tradition of archiving movie memorabilia, *History!* would be able to archive movie websites for generations to come.

Restaurants: The food and hospitality industry could use the archival features to save past menu items and promotions. Traditionally there are lots of incremental changes that take place in the restaurant industry. Archiving these changes and overlaying them with performance metrics would offer this industry a new perspective.

Hospital Systems: Hospital systems have an interest in archiving past events, offerings, and promotional campaigns. This industry is very aware of keeping and

archiving precise records. *History!* could potentially help this industry track changes to their sites and medical personal.

Law Firms: Law firms have an interest in tracking the history of their legal team, important legal cases, as well as monitoring nearly constant changes in legal precedent. *History!* would be able to assist a law firm in all these regards.

Technology Companies: Technology companies tend to adapt depending on customer needs. Tracking and archiving these kinds of changes can assist technology company's maintain a historical perspective when it comes to important decisions. Incremental changes in the layout of a website can be tracked and overlaid with performance metrics to get a better idea of how their design changes are being received by their customers.

Retailers: Retailers and market trends in this industry often follow highly cyclical consumer trends. *History!* can offer not only an archival record of their own online presence, but also of their competitor's online presence. By tracking what their competitors are doing in their one online presence, companies can respond more effectively to market trends. This would give retailers a competitive advantage over companies not using the proposed application.

Marketing Agencies: Similar to retailers, all marketing agencies have campaigns that are highly successful and other that are not as successful. Today's marketing focuses heavily on a company's online presence. By tracking past online campaigns, marketing agencies can build up a large archive of past successes and learn from their failures. An application like *History!* Could also aid these agencies in reusing the more successful parts and aspects of their ad campaign.

Individual Entities

Family Archival Enthusiasts: Much of people's lives and memories are documented through an online presence. Whether it be Facebook, Instagram, or another kind of social media there are many memories and important family events that might be lost when a technology company ceases to exist (eg. MySpace). People interested in tracking their family's presence and their own presence could leverage *History!* to archive their memories for generations to come.

Historians: Similar to one's social media presence, many historical events are documented and chronicled over the internet. This loss of physical historical artifacts puts the historical record at risk of being lost. Historians need to be able to accurately document and archive these kinds of historical events for future generations.

Academics: It is hard to accurately document, and study large-scale industry trends in a rigorous scientific fashion. *History!* could be used as a tool for researchers to better understand industry trends, and solve complex problems by allowing them to analyze trends over long periods of time.

Accessibility Analysts: Accessibility remains an important yet sometimes overlooked aspect in the design of online websites. Analysts studying the general accessibility of websites across the internet could use *History!* to better understand how designers are improving their designs to facilitate the use of their websites by people with various kinds of disabilities.

Other Stakeholders

Cloud Service Providers: The team's current architectural design implements a cloud-based ecosystem of tracking changes to websites. Analysis, storage, and CMS services will all be provided through a cloud service. While not directly involved in the project, the cloud service provider chosen to implement the project will be an integral part and stakeholder in the project's success.

DCMA Advocates: There is a large and growing industry of DCMA legal advocates working on behalf of corporations like Disney to ensure their copyrighted content is not being hosted illegally on the internet. *History!* could potentially inadvertently archive content that violates DCMA standards.

Online Privacy Advocates: *History!* could potentially archive content that violates *Right to Be Forgotten* laws like the European Union has adopted. The application would have to comply with all requests involving their citizens. There are also instances where individuals or corporations have won a legal court order forcing certain content to be removed from the internet.

Subjects of *History!* Tracking: Individuals are free to track any publically available website. This does not necessarily mean they have consented to being tracked. These entities will have a tangential association with the application. In certain instances we might have to notify individuals or entities that they are being tracked and archived.

Goals

The software allows a user to view the version history of any web that has been designated as "tracked".

- It creates an archive of any websites history for future use.

The software allows the user to view any two designated versions, selected by date or range.

- The software determines the differences of the designated pages and allows the user to view these differences side by side within the browser by highlighting them.

The stakeholder has a history of software with an exemplary UI and UX experience.

- This application will achieve these same goals.

The software shall clearly communicate to the user whether the website they are on is or is not being tracked.

- **[Nielsen - Visibility of System Status]** Users should be able to tell fairly quickly whether the website is being tracked or not.

The software shall clearly communicate clearly where and when specific archives have been retrieved from relative to the historical record.

- **[Nielsen - Visibility of System Status]** Users should be able to quickly tell where and when a particular version of a website came from.

The software shall allow the user to easily start, stop, and pause the tracking of websites.

- **[Nielsen - User Control and Freedom]** Users need to be able to easily control the websites they are tracking, including starting/stopping/pausing sites.

The software shall allow users to easily browse and select specific archives related to the website they are tracking.

- **[Nielsen - User Control and Freedom]** Users should be able to easily and quickly pick out the archive they would like to compare with the present version of the website.

The software shall confirm with the user that he/she would like to discontinue tracking particular websites.

- **[Nielsen - Error Prevention]** The software should prevent users from inadvertently stopping the application from tracking of websites. It will instead prompt the user to confirm they would like to do so.

The software should be able to track the vast majority [>98%] of websites users would like to track, excluding password-protected websites.

- **[Nielsen - Flexibility and Efficiency of Use]** The software should be flexible and robust enough to track a wide variety of websites built on most common frameworks.

The software shall not intrude or distract from the user's usual browsing experience.

- **[Nielsen - Aesthetic and Minimalist Design]** The software should be able to indicate to the user the system status without distracting the user from browsing the tracked website.

The system shall save archived websites after all users have elected to stop tracking the specific site for a minimum of 30 days.

- **[Nielsen - Help Users Recover from Errors]** If users inadvertently stop tracking a website, the application will afford the user the opportunity to undo the change.

The user shall be able to access a FAQ to get answers to common technical / interface issues.

- **[Nielsen - Help and Documentation]** The application shall provide the users with enough information and documentation regarding the application to solve/answer the majority of issues/questions.

Constraints

- The application assumes technology requirements of an internet capable device with a qualified browser and internet access.

- Browser selection will be a constraint since only a qualified browser will work with the application/browser extension.
- Storage capacity is limited, even with a cloud-based system. There may be an upper limit of storage capacity any one user can consume at any given time.

Assumptions

The software assumes public-facing web pages.

- Websites that require a login will not function with the software/browser extension.

Ideas

Several ideas were generated during the brainstorming process.

- Users receive notifications (if opted in) when websites are updated.
- Users can follow specified sites.
- Users can designate the method and frequency of notifications
- Users can track/record HTML version changes.
- Users can attack UI/frontend changes.
- The Google Cloud API can generate descriptions of differences of selected websites.

Application Design

Overall Design: The user will be able to track the history of any website on the internet using the Chrome extension *History!*. The application allows the user to view and compare any previously archived websites. The user has full control over stopping and starting tracking and also can view a list of currently tracked websites. The application has the following features.

Hardware Requirements: Since all that's required for the application is a qualified browser and extension, the application will work on any device with a browser and internet access. This includes desktops (Mac/Windows), mobile devices (iOS/Android) as well as tablets.

Language Support: The application will support any language the browser supports.

Accessing archived sites: Similar to the UI in Google docs, the user can click to see the website history through a Chrome extension at the top of the page. The user can interact with the archived version of the webpage as if it were the current version. The version history is also available for offline viewing.

Tracking Function: Once the extension is installed, through a popup on the right side of the browser, the user can start or stop tracking on any website. The application can track from the present to when a user decides to stop tracking a website, or from when any user has started tracking the website.

Calculating differences: Similar to GitHub, the application saves all incremental versions allowing it to compare and highlight differences. The user can view the current site and the archived one side by side highlighting what the differences are and two

sites can also be viewed between different points in time. The comparison of websites is limited to two side by side. In addition to highlighting the differences in the two versions, a textual description of the images and syntax/sentiment of the language will be provided for the two versions.

Notifications: The user will be notified when there is a change in a tracked website through a Google Chrome notification. The user can designate the frequency and method for notification such as email or SMS.

History Range: The dates specified for tracking can be a range or specific dates
Versions are divided by inverse chronological order (newest version comes first)
By minutes → By hours → By days → By weeks → By months → By years

Mind Map

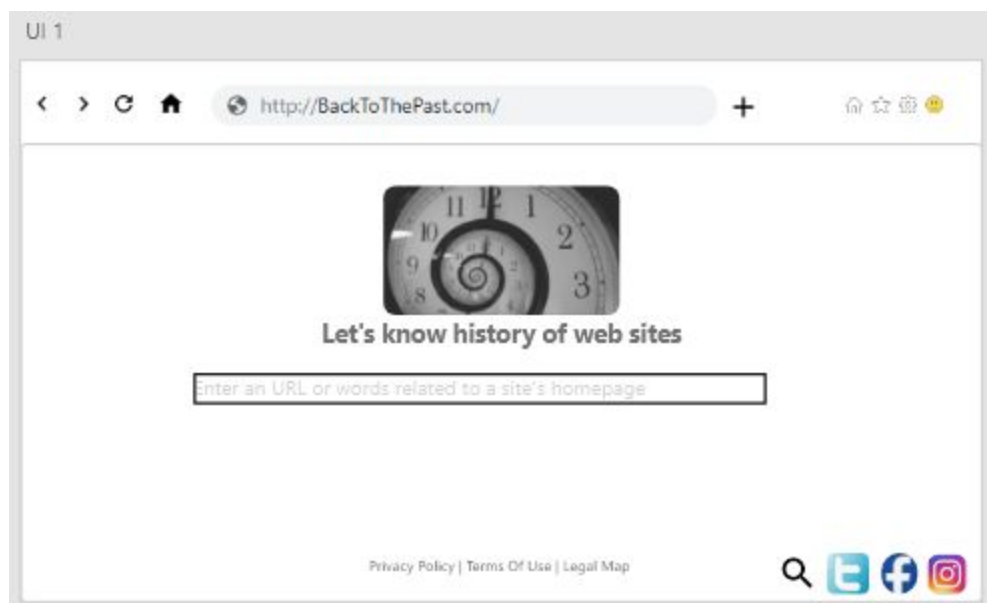
<https://www.lucidchart.com/invitations/accept/83227cb4-da2f-46c6-aec9-8007b37cc7cb>






Interaction Design

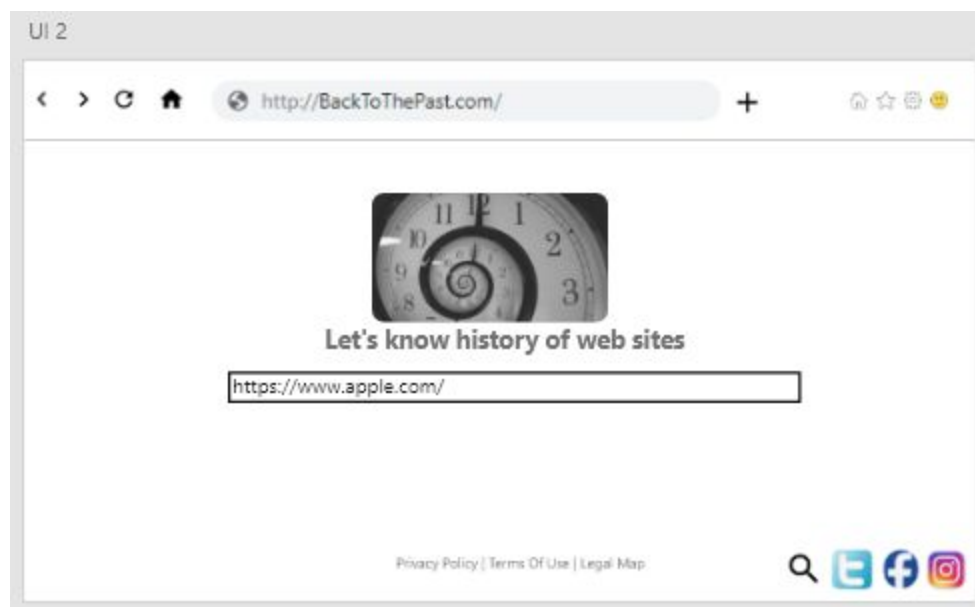
Introduction


The purpose of the interaction design was to depict how the user will interact with the tracking software. This design was created to demonstrate how the user will achieve the best possible user experience from the software. To do such, each of the components with which user will interact to visualize two different versions of the website, is described below in detail.

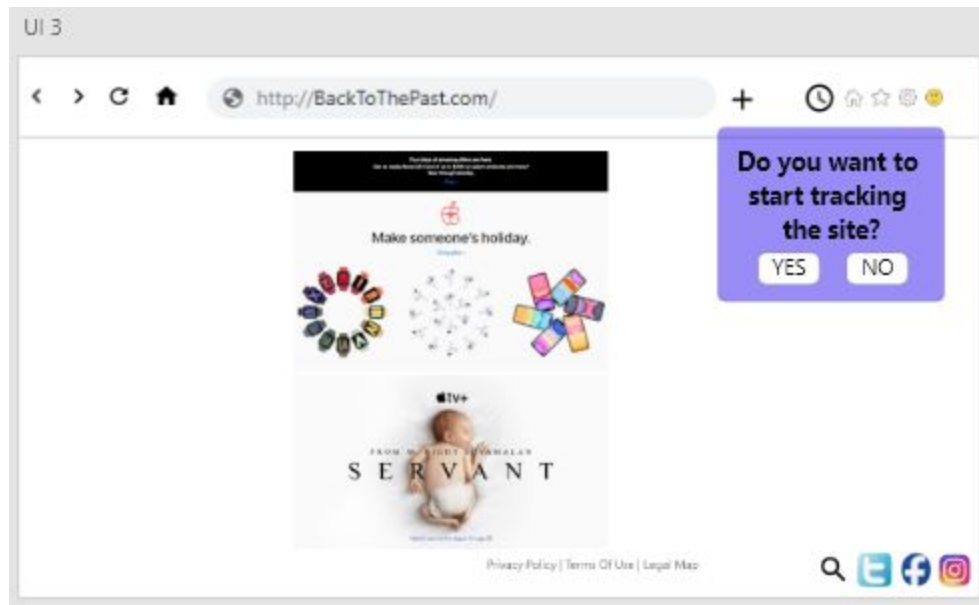


Icon Image	Icon Description
	This is the URL of the website where user will search for websites to track different versions of websites. The name of the website is Back to the Past.
	This icon represents the logo of the website. The counter clockwise image of the logo is intended to help users have an idea that the website is intended to take users to previous


	versions of websites from the past.
	This is a search box which prompts users to enter either a URL or words related to a website's homepage for the destined website.
	This is a generalized search box where if the users are looking for any specific things to find, they can search for it in this box.
	Through these social media icons, users can create and share contents, and participate in social networking with Back to the Past users.

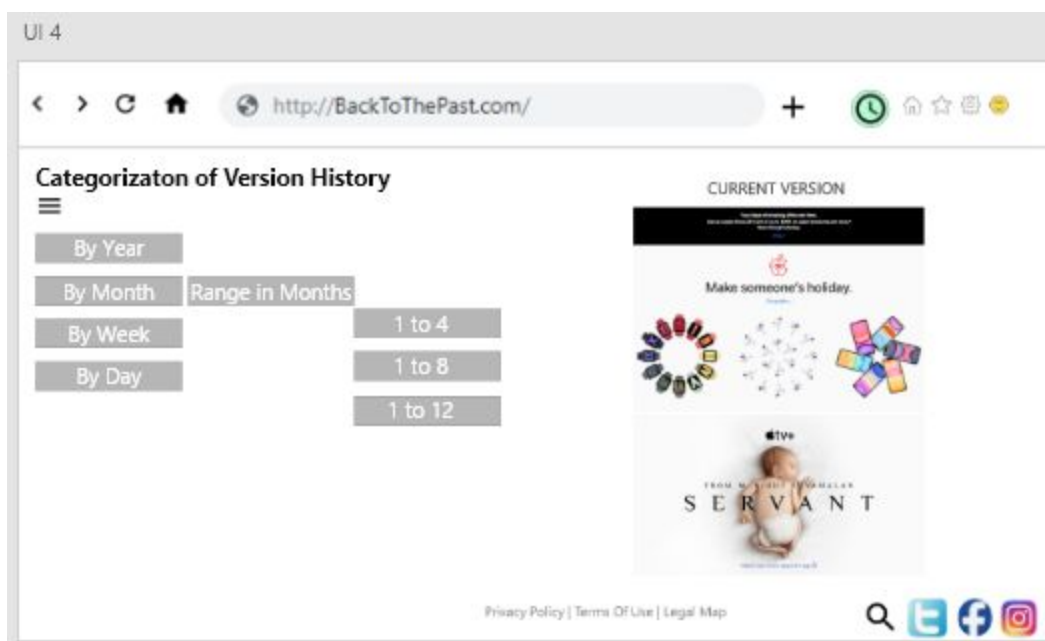






Icon Image	Icon Description
	As soon as users enter their destined website address in this search box, it becomes visible in the box.



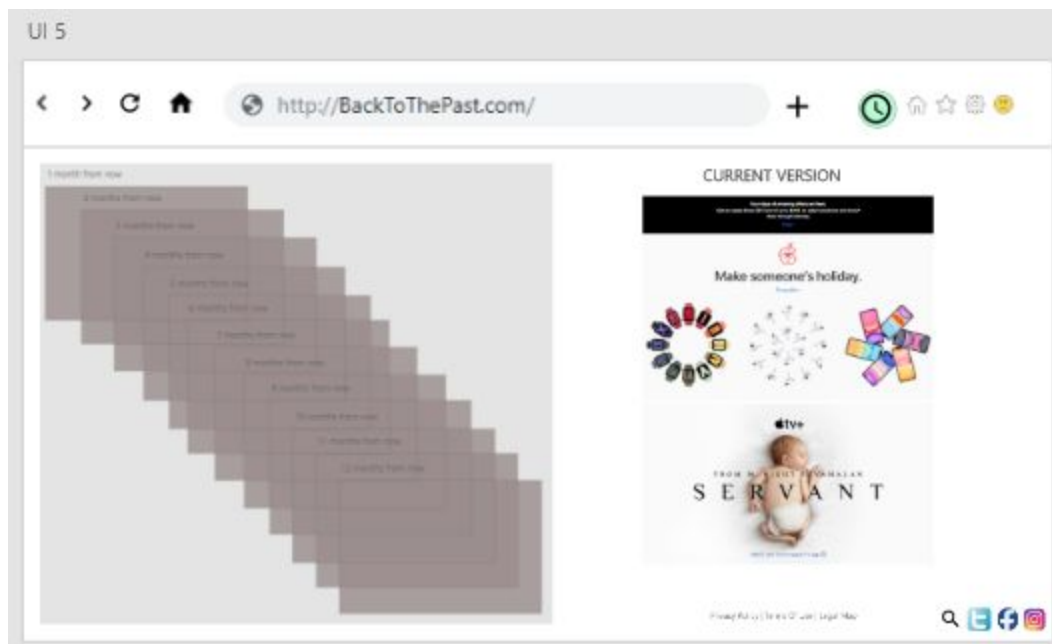
Icon Image	Icon Description
	After users enter the URL or address of their destined website, it displays the current version of the website.
	As soon as the current website becomes visible in the interface, Chrome or Safari Extension Icon becomes noticeable to users.
	Following the visibility of the extension icon, a pop-up prompt shows up in the user interface indicating if the user wants to start tracking the site or not.
	This is one of the two icons which is a button for asking users permission to start tracking the website. "YES" indicates if the agrees to

	start tracking.
	This is one of the two icons which is a button for asking users' permission to start tracking the website. "NO" indicates if the user does not agree to start tracking.



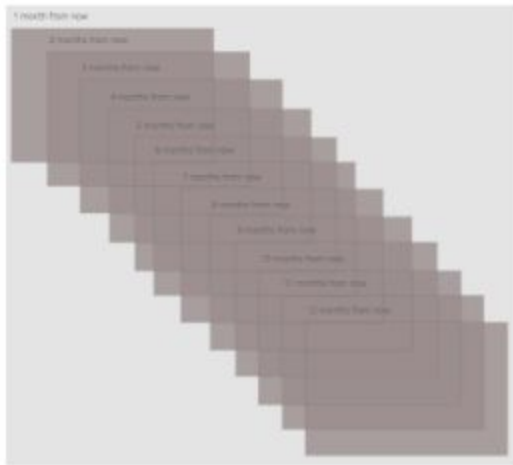
Icon Image	Icon Description
	If the user presses "YES" in the pop-up prompt, extension icon for tracking website turns green and stays green as long as users intend to track the website.
	This is a menu icon to categorize version histories.
	"By Year" involves categorization by years.
	"By Month" involves categorization by months.

1 to 4	If user selects “By Month” categorization, then the option of choosing month 1 from now to month 4 becomes visible.
1 to 8	This is another option for categorization by month. Users can select month 1 from now to month 8.
1 to 12	This is another option for categorization by month. Users can select month 1 from now to month 12.
By Week	“By Week” involves categorization by months.
By Day	“By Day” involves categorization by days.

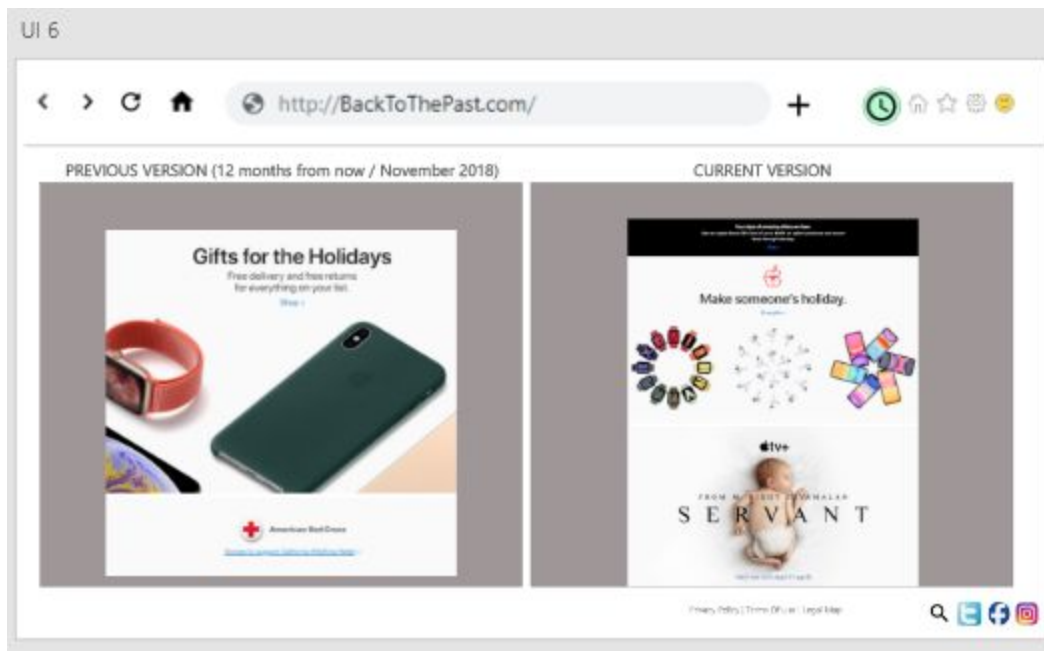


Icon Image

Icon Description





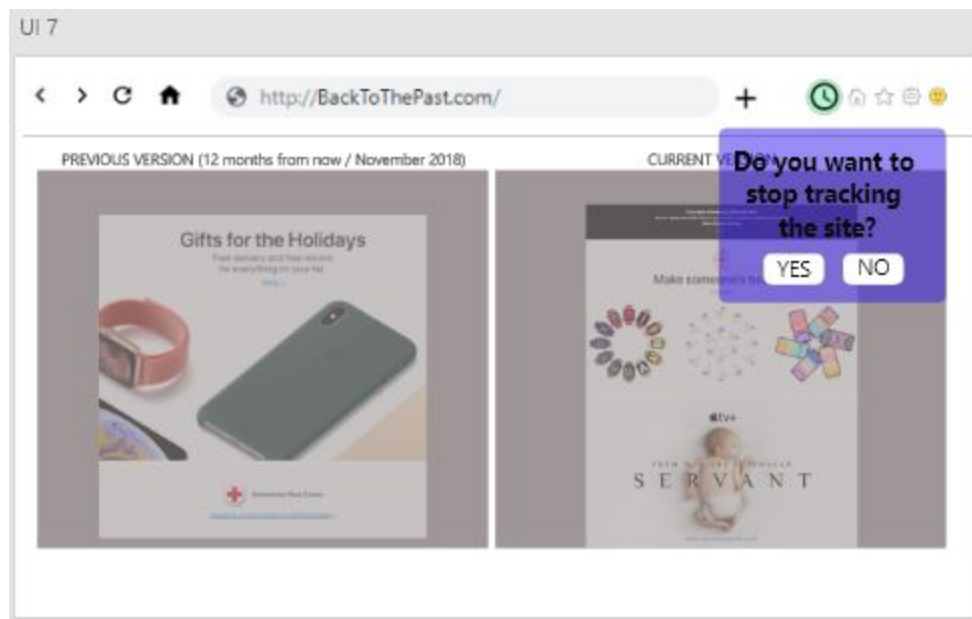
If users chose categorization “By Month” and range in months “month 1 from now to month 12”, then 12 months version from the current version will be distinguishable in the interface. Users can chose any version from these versions.






Icon Image

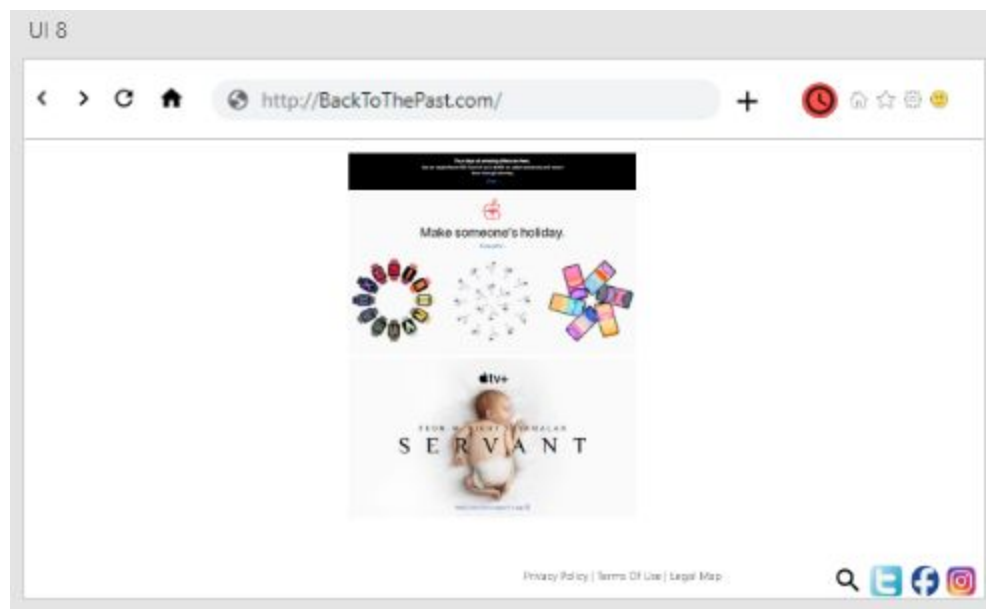
Icon Description


<div data-bbox="212 212 488 226">PREVIOUS VERSION (12 months from now / November 2018)</div>  <div data-bbox="602 212 688 226">CURRENT VERSION</div> 	<p>From 12 months versions from present time, if the users chose “12 months from now”, the current version will be shown in the right and the past chosen version will be shown to the left. Viewing two objects vertically (side-by-side) is easier for human eyes. Thus, versions were depicted vertically for the users for comparison with ease.</p>
---	--



Icon Image	Icon Description
	<p>If the user intend to stop tracking the site and click on the extension icon, a pop-up window with prompt will appear. The prompt will ask the user if he or she wants to stop tracking the site or not.</p>

<p>PREVIOUS VERSION (12 months from now / November 2018)</p>  <p>CURRENT</p> 	<p>When the pop-up window appears in the screen, the background of the site will have a transparent blur effect so that user's attention remains in the pop-up prompt.</p>
---	--



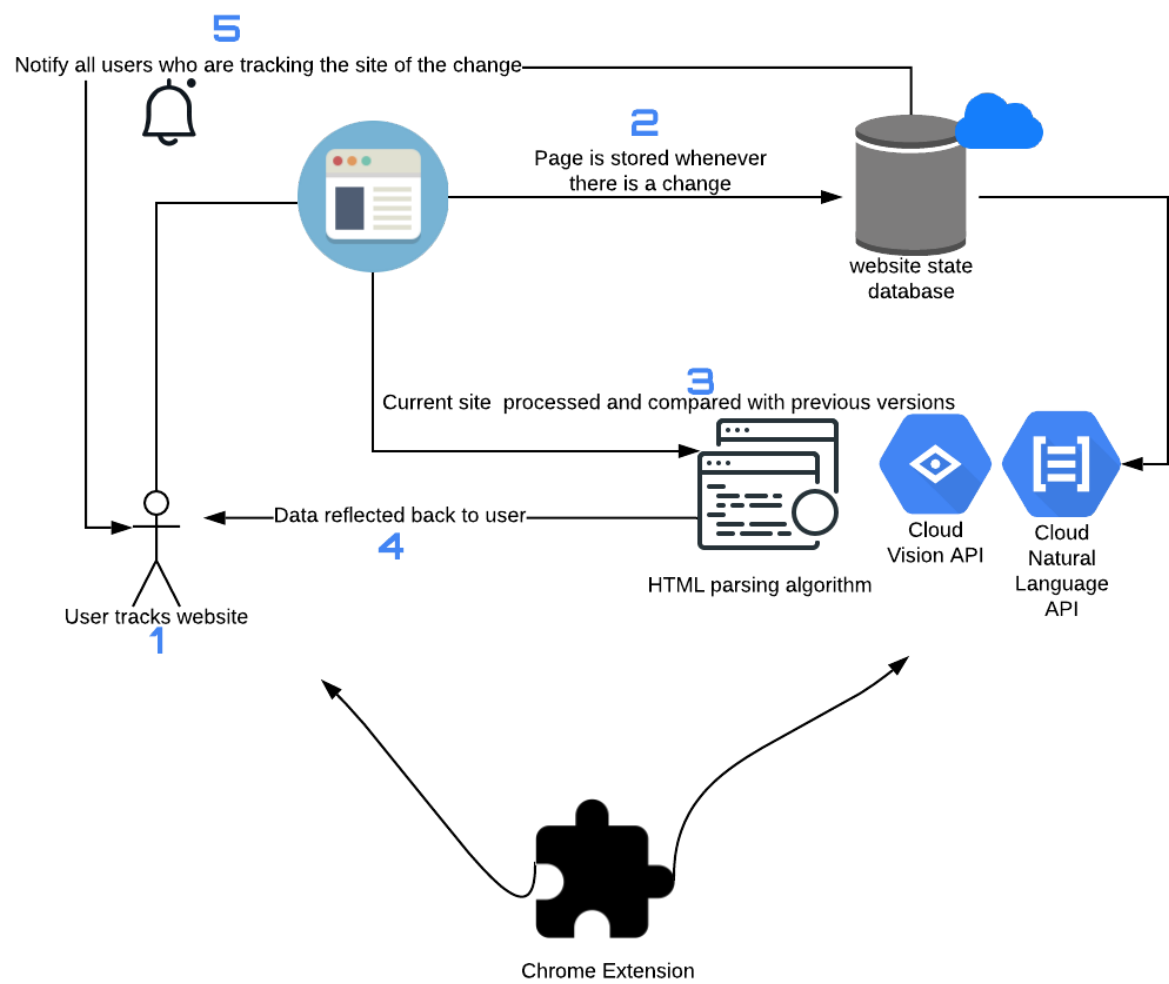
Icon Image	Icon Description
	<p>If user selects “YES” from the pop-up prompt, the extension icon will turn red meaning tracking is tuned off.</p>

Architecture Design

Introduction:

The conceptual core of this (and any other) software project is the architecture design. This project will be implemented as a Google Chrome extension because doing so ensures that the needs of all audiences and stakeholders are met. In a broad view, the architecture of this project involves storing the tracked webpage states in a cloud based database. When a user decides to start tracking a website that has not yet been tracked by any other users, The current state of the web page will be stored in the database. Whenever there is a change detected in the tracked page, the new page will be stored in the database. If there is a change detected, a chrome notification will be sent to the user via the chrome.notification API. When a user decides to compare previous version of the website they are tracking, the current state of the website along with the previous states stored in the database will be compared through an HTML parsing algorithm, the Cloud Vision API, and the Cloud Natural Language API. The changes detected and described by these processes will then be reflected back to the user.

Overall Architecture Flow Chart



Action	Description
1: User tracks website	When the user begins tracking the website, the website is added to the array of tracked websites associated with the user.
2: Page is stored whenever there is a change	The webpage is monitored for a change in the HTML. Anytime there is a change in the HTML, the new version of the webpage is saved in the database. This is only done once for any website, even if the site is being tracked by multiple users. As a result of using a cloud based database, when choosing the website versions from which to compare pages the user has the ability to select versions from either as far back as they have began tracking the page (if they are the first user to track

	the page) or as far back as the first user that began tracking the page started tracking. This is because the tracked website data of all users is stored together in a shared database to reduce the memory requirement of local storage. Each website is stored in the database as a dictionary in which the key is the timestamp of the website version, and the value is the website version itself.
3: Current site is processed and compared with previous versions	Using the HTML parsing algorithm, the Google Cloud Vision API, and the Google Cloud Natural Language API, the current state of the site will be compared with the previous state of the site chosen by the user. These processes will work together to find and describe the differences between the two versions of the site. <i>A more detailed description of how the comparison process will work is provided after this table.</i>
4: Data reflected back to the user	After the comparisons between the two sites have been made, the data will be reflected back to the user.
5: Notify all users who are tracking the site of a change	Using the chrome.notification API, a notification will be sent to users who are tracking a website when there is a change. This will be done simultaneously with step 2, so whenever it is determined that there has been a change to one of the sites in the database a notification will be sent.

Comparison Process:

The comparison process is composed of three parts: an HTML parsing algorithm, the Google Cloud Vision API, and the Google Cloud Natural Language API. These components work together to analyze the difference between two different versions of the site, and relay the relevant information back to the user about the differences. As seen in the above architecture diagram and subsequent description table, the current version of the web page is compared with the selected previous version of the webpage using all three of these comparison methods.

- **HTML Parsing Algorithm:** the HTML parsing algorithm compares the two versions of the webpage through the differences in their HTML code. The HTML code is easily obtainable from the webpage, and the differences in HTML between the two website versions will be returned from the algorithm. The basic flow for this algorithm would look something like the following
 - Look for identical chunks of HTML in both of the versions*
 - If the chunks are located in different parts of the page, add this as a change*
 - For all the text/images on each page*
 - If the text/image only exists in on version of the page, add this as a change*
- **Google Cloud Vision API:** by using the Cloud Vision API, changes in the different images used on the webpage can be conveyed. The API will provide an overall summary

of the images on each version of the page, and the result of using the API will be produced as part of the comparison process.

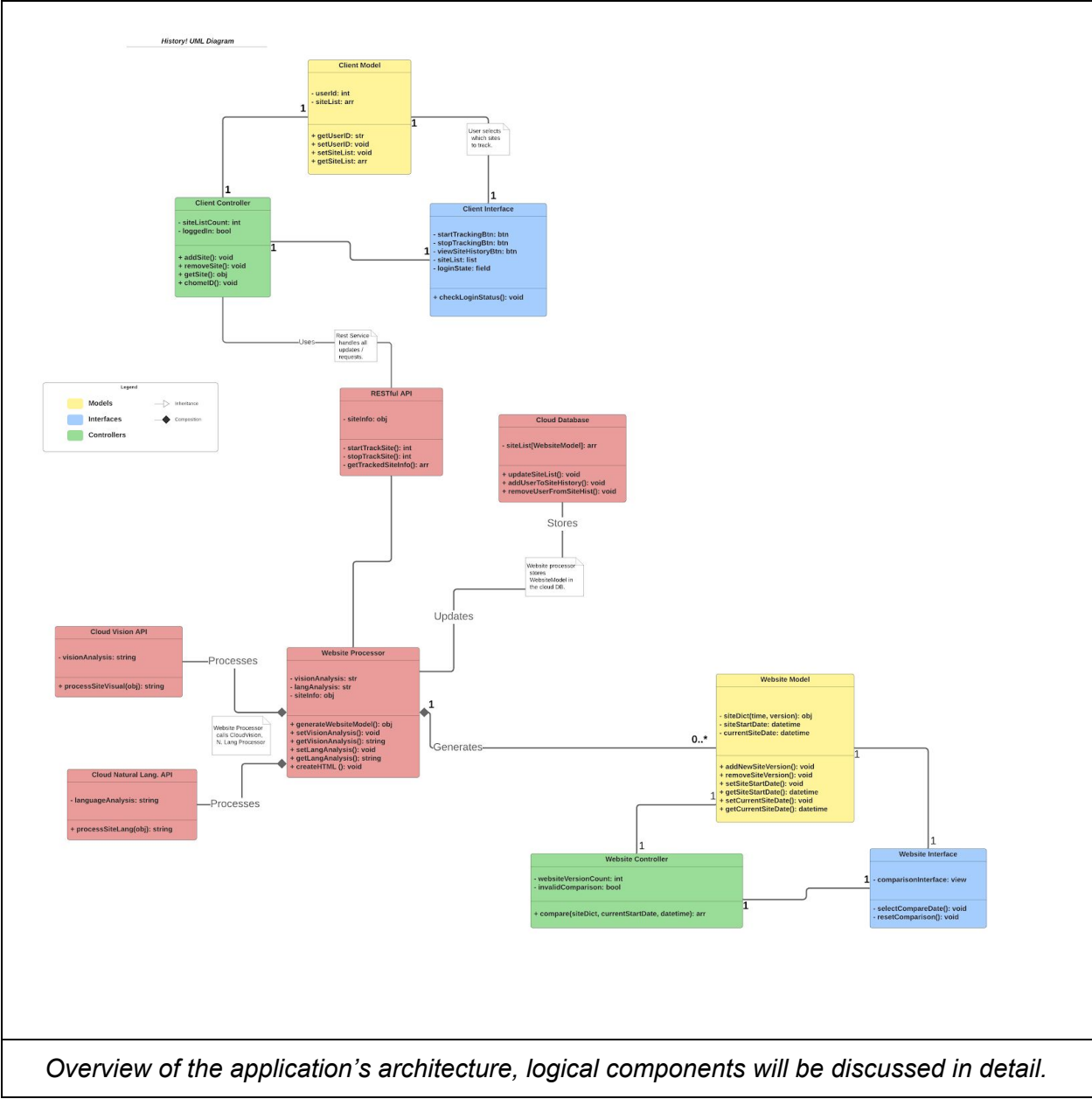
- **Google Cloud Natural Language API:** the Natural Language API will be used on the text of the two different versions of the webpage. This API has many capabilities that can be utilized, but for this software project the syntax and sentiment analysis will be used. Using these features of the API will produce a description of both the language structure and tone of each webpage.

UML Class Diagram

Description: The architecture of the team's chosen design was intended to be simple in order to maximize performance, and user ease-of-use. There are 3 primary components: the User Models, the Website Processor Models, and the Website Models. Together these components work together to track websites overtime, analyze changes, and update the database of stored websites. The architecture allows for uses to manage the sites they want to track and compare the site-version they are most interested in. To help define the architecture of the system the team broke down the application into logical parts.

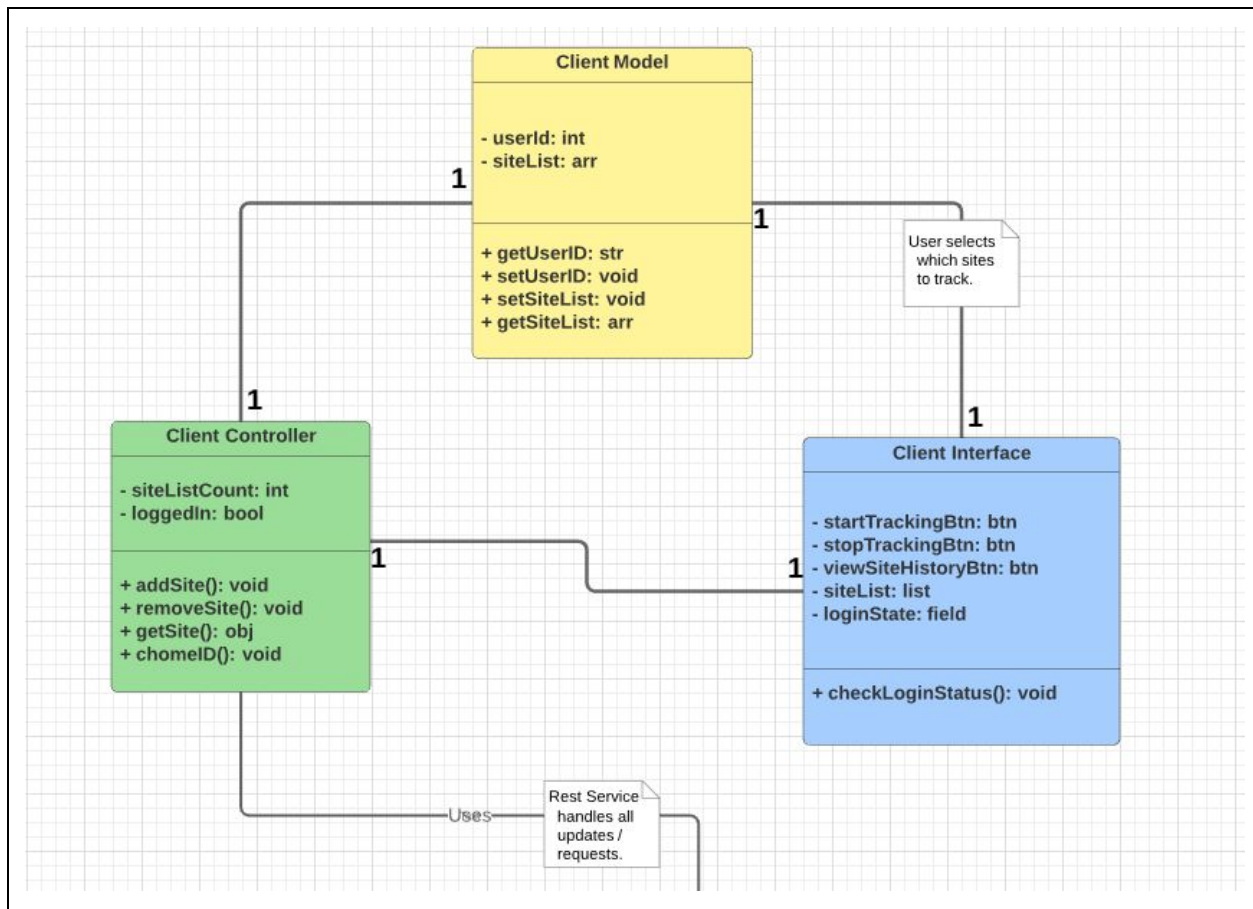
1. **The User View:** The user view is where the application's user will utilize the application to start tracking new websites. From this area, they are able to view the websites they are tracking, as well as having the option to stop tracking websites.
2. **The Website Processing View:** The processing view is where site changes are processed through the Cloud Vision API, as well as other site-analysis tools. Once a site has been processed and analyzed the website processing view will also store the new version into the cloud-based database.
3. **The Website Model View:** To allow users to compare multiple-versions of the same website the website model view is utilized. The WebsiteModel encapsulates the entire historical record of any given website. When a new version of the website is encountered, the WebsiteModel is updated to reflect those changes.
4. **Additional Components:** To help facilitate communication between the various components, and to store site information over longer periods of time that a user can locally save -- a RESTful service, and a cloud-based database will be leveraged. The service will process user interactions, as well as update the database with new site information. The cloud service stores the entirety of a websites record so that all users have the ability to look as far back as possible, irrespective of when the site was personally started to be tracked.

History! UML Diagram



Overview of the application's architecture, logical components will be discussed in detail.

User View



Client Controller

When users view other versions of the websites, `addSite()` will be called and will add the version to `siteList` inside **Client Model**. Users can also delete a version of the website from their `siteList` inside **Client Model**, and `removeSite()` will be called. Users are able to retrieve one version of the website from the `siteList` in **Client Model** with function `getSite()`.

Attributes

siteListCount: A real-time count of the number of websites the current user is tracking.

loggedIn: Boolean value that tracks whether the user is currently signed into the chrome browser.

Methods

addSite: This method is used when the user elects to track a new site, it will call the restful service to begin archiving the chosen website.

removeSite: This method is used when the user elects to stop tracking a given website. The method call on a restful service which will disassociate that UserID with the website's

archive.

getSite: This method is used when the user elects to start the comparison process in a website that they are currently tracking. The restful service will retrieve the version-history from the cloud service for the user to compare.

chromeld: The application is Chrome Extension, this extension can uniquely identify users based on who is currently logged into the browser. Using this information, the application can accurately recall the user, and which websites they were tracking.

Client Interface

Description: The client interface is where the user will be able to control the websites they are tracking. From this interface users will be able to start, stop, and view the websites they have elected to track.

Attributes

startTrackingBtn: This button will trigger the addSite method to begin tracking a given website.

stopTrackingBtn: This button will trigger the removeSite method to stop tracking a given website.

viewSiteHistoryBtn: This button will trigger the comparison view, where the user can start to compare different versions of the website they are interested in.

siteList: Site list is an easy to access list of websites a user has elected to track, from here the user can navigate to the website they are interested in.

loginState: This is an indicator that will quickly let a user know whether they are logged in or not.

Methods

checkLoginStatus: This method will check whether the user has successfully authenticated through the Chrome browser.

Client Model

Description: The client model will encapsulate individual users in the system. Within this system the ClientModel will store the user's unique id and the sites they are currently tracking. As the users stops/starts tracking websites, the ClientModel will be updated to reflect those changes.

Attributes

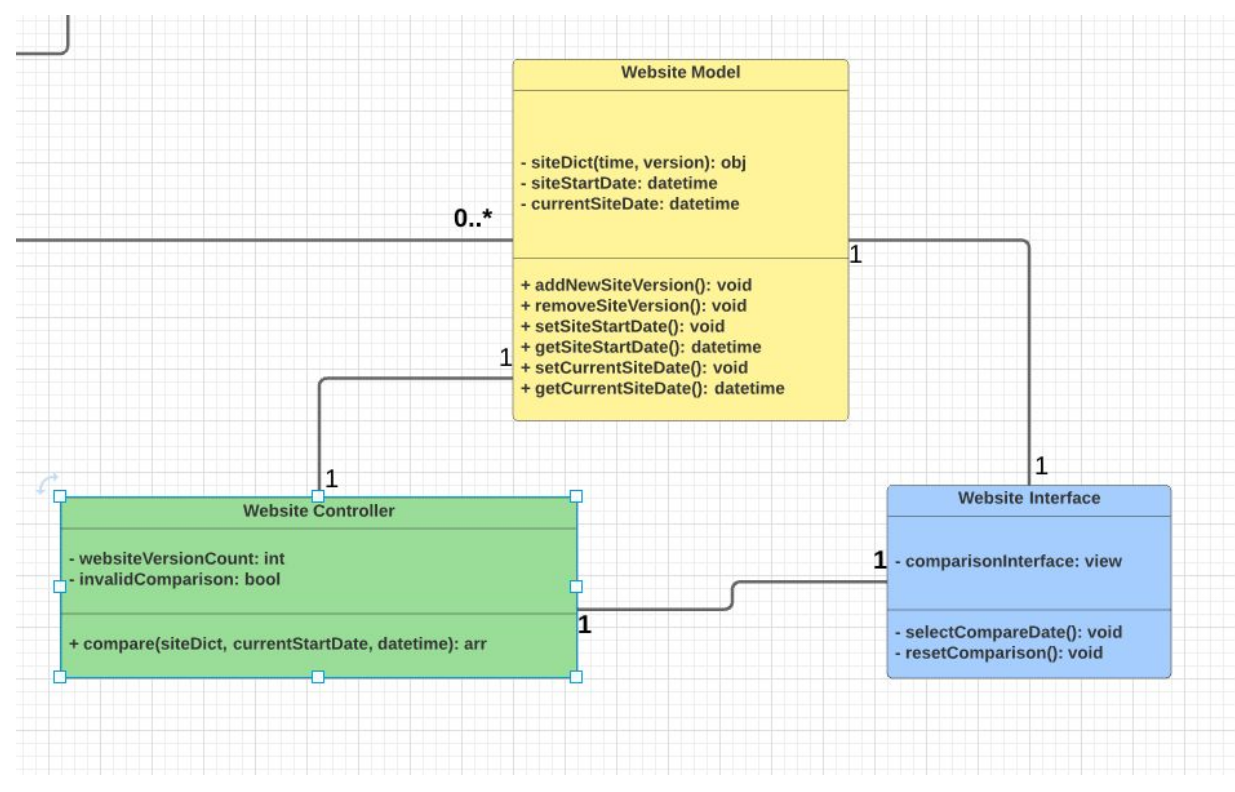
userID: This unique id will allow the system to uniquely identify and service users.

siteList: A user's site list is a unique list of all the websites the user has elected to archive.

Methods

getUserId: Method to retrieve a user's id.
setUserId: Method to set a user's id.
setSiteList: Method to set a user's site list.
getSiteList: Method to retrieve a user's site list.

Website Model



Website Model

Description: The website model is an encapsulation of any given website's historical record, beginning when the site was first selected to be monitored for changes. The website model contains a dictionary that is categorized by the date entries were captured. Using these dates, the system is able to accurately track website changes over time.

Attributes

siteDict: The site dictionary will store the entire historical record of any given website. When a website is updated a new entry will be added into the dictionary. The dictionary uses timestamps as keys.
siteStartDate: Date-time object identifying when a website was first started to be tracked.

currentSiteDate: Date-time object identifying the most current version of a website.

Methods

addNewSiteVersion: Method to add a new site version into the siteDict dictionary.

removeSiteVersion: Method to remove a site version from the siteDict dictionary (seldom used).

setSiteStartDate: Method to set a website's start date, the date the website was first started to be tracked.

getSiteStartDate: Retrieves the website's start date.

setCurrentSiteDate: Method to set a website's current site information, specifically the date of the latest version of the website.

getCurrentSiteDate: Retrieves the website's current-date.

Website Controller

Description: Website controller will serve as the primary engine to compare websites. Based on user input, the controller will recall the appropriate website version and trigger the comparison process. The comparison process, will populate the comparison view where the user will be able to analyze different versions of the same website.

Attributes

websiteVersionCount: A representation of the number of different versions of the website that exist in the dictionary in any given point in time.

invalidComparison: Boolean value whether a valid comparison was able to be generated.

Methods

compare: This method will allow the user to view two different versions of the same website. Using the site's timestamp information it will recall the correct version information and generate the comparison view for the user to utilize.

Website Interface

Description: Users will see two versions of the same website horizontally. The reason we designed the horizontal layout is that humans, as has been researched, naturally read from left to right. We covered more of it in our interaction design.

Attributes

comparisonInterface: This is the main comparison interface that will allow the user to

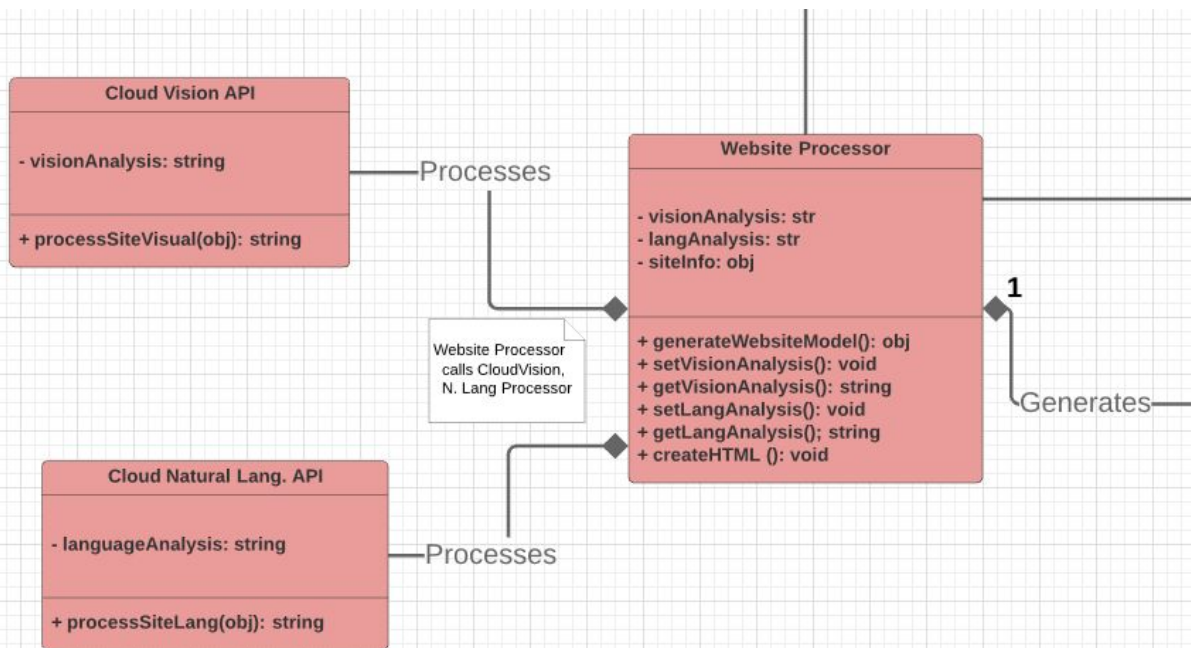
compare different versions of the same website.

Methods

selectComparisonDate: Method that will start the comparison process, based on a user's selection.

resetComparisonDate: When a user is finished comparing a given website this method will clear the comparison interface.

Website Processor UML



Cloud Vision API

Description: The cloud vision API works with the website processor to produce valuable insights for users. Specifically the cloud vision API will be able to analyze the visual representation of any given website, produces a string value.

Attributes

visionAnalysis: String-based representation of the website's visual characteristics.

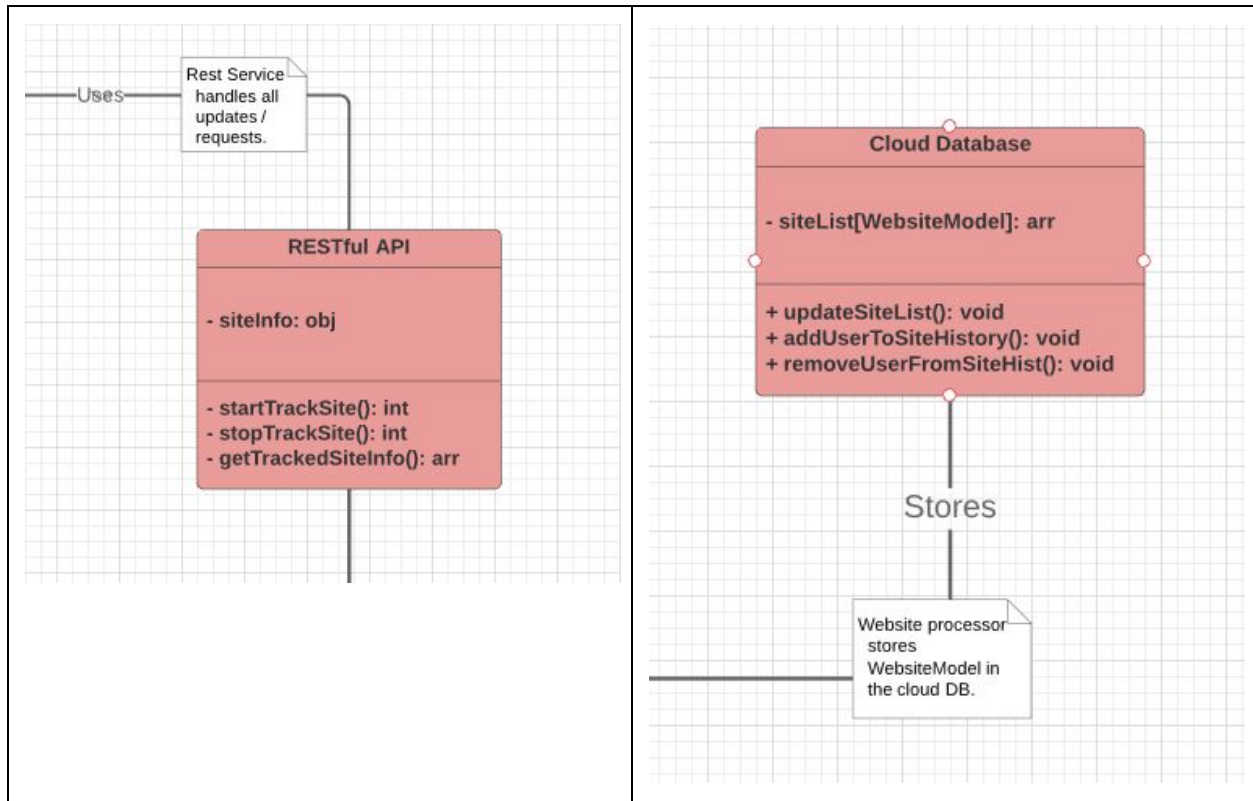
Methods

processSiteVisual: Method that will call the CloudVision API and feed it the necessary site information, returns a string-based interpretation of the site information.

Cloud Natural Language API
Description: The cloud-based natural language processor will analyze changes to the website's text-based characteristics and produce a representation of the natural-language characteristics of the given website. This element monitors for changes of websites based on the textual content of a given website.
Attributes
languageAnalysis: String based representation of the website's textual characteristics.
Methods
processSiteLanguage: Method to call the Natural Language API to produce an analysis for the languageAnalysis attribute.

Website Processor
Description: The website processor is where websites are processed, analyzed and stored in the cloud database. It also calls and utilizes the Cloud Vision, and Natural Language processor components. Together this information is compiled to form a site model in the WebsiteModel dictionary.
Attributes
visionAnalysis: Representation of the analysis of the website's visual characteristics. langAnalysis: Representation of the analysis of the website's textual characteristics. siteInfo: Encapsulation of any given website's archival history.
Methods
generateWebsiteModel: Method that is triggered when a website is updated, created, or altered in anyway. This method will update the website's corresponding website model and updates the database with a new version of the website model. createHTML: Method to produce a representation of the website's HTML.

Additional Components	
RESTful Services	Cloud Storage



RESTful API Model

Description: This component encapsulates the restful service used to communicate user interactions to the website processor. It can start, stop, and retrieved tracked site information. When triggered it sends a `siteInfo` object to be processed in the user interface.

Attributes

siteInfo: This is an object of website model that the user has selected to examine further.

Methods

startTrackSite: Method to begin tracking site information.

stopTrackSite: Method to stop tracking site information,.

getTrackedSiteInfo: Method to retrieve tracked-site information.

Cloud Storage Model

Description: Cloud-based storage for archiving site information. The website processor updates, retrieves, and creates new website models to be stored in this database.

Attributes

siteList: The database schema that stores the websites the application is currently tracking.

Methods

updateSiteList: Method to update the site information being stored.

addUserToSiteHistory: Method to add a user's request to a site being archived.

removeUserFromSiteHist: Method to remove a user's tracking request to a site being archived.