Department of Computer Science and Engineering

The Chinese University of Hong Kong

CSCI 4999 Final Year Project II

TAO2203

Luminate - Web Annotation Tool

Written by

WILLIANTO, Angelica (1155136022)

Supervised by Prof. Yufei Tao

19 April 2023

Table of Contents

Abstract

Online research and information consumption has become a major part of our daily lives, whether for leisure or for work. With the excessive amount of information on the internet, capturing, managing, and sharing information via annotation becomes crucial. People continue to rely on the time-consuming process of copying and pasting links and text. This way of managing information remains largely inefficient and limits opportunities for collaboration and interactive learning. We need a tool that can streamline the annotating workflow and foster a collaborative learning environment.

Our solution to this is a web annotation and social collaboration tool named Luminate. Luminate is a browser extension that gives users the ability to highlight text, add their own personal annotations, and share insights on any webpage. Highlights and notes can also be viewed through the Luminate Web App. Users can set their annotations as public or private, and follow other users to view their shared annotations. Using Luminate, users can retain their highlighted text and notes as long as the extension is activated. This provides users with a tool to manage, organize, access, and collaborate on their online information in an easier and more efficient way. Our app will be beneficial to a variety of people, including students, researchers, teachers, and others who regularly consume a large amount of online information and seek to engage in collaborative learning experiences.

Acknowledgements

We would like to extend our sincere gratitude to our supervisor Prof. Tao Yufei for his guidance and kind support throughout the course of this project.

We would also like to thank everyone who helped us with the project or read this report.

1. Introduction

1.1 Motivation

Whether at home or at work, a large portion of our information consumption and research has shifted online. A lot of our time is now spent looking into information online. The internet has become an invaluable resource for us to access information quickly and easily. We can find information on almost any topic, from the most obscure to the most popular. We browse through websites for topics such as COVID-19, news reports, work projects, travel, shopping, school assignments, etc.

As technology continues to evolve, the amount of online information available to us grows exponentially. According to siteefy [1], there are currently around 1.14 billion websites, and 252,000 new websites are created every single day. Finding the right information in this massive amount of information is difficult. Once we find information, we must not lose track of it.

Moreover, the learning process is often enriched when we can share our insights with others and learn from their perspectives. Traditional methods of annotating web content do not facilitate easy sharing or collaboration, limiting the potential for interactive learning experiences.

A simple method to keep track of all the information we would like to keep is using the bookmark function that is available in all browsers. However, bookmarking is not enough. Without the content and note information, it is hard to remember why we marked down a link a month ago and which part of the website is important.

Hence, a more detailed way to take notes from a website is by copying the text and pasting it into a cloud-based note-taking app or a text editor such as OneNote or Microsoft Word. The step is usually as follows:

Figure 1.1.1 Common Annotating Workflow

There are three concerns regarding this method:

1. **Inefficient:** This process of gathering information remains largely time-consuming and involves a lot of tedious labour. There are at least three steps involved required to complete a single note.
2. **Loss of context details:** While reviewing this note, we can only read the sentence we pasted into the document, rather than the whole webpage. If we want to understand the sentence in the whole context of the webpage, we have to enter the link and scroll to the exact location of this sentence. This whole process can be made more efficient.
3. **Limited collaboration:** The traditional annotating workflow is lacking in features that allow users to share their insights and learn from others' perspectives, ultimately limiting the potential for interactive and collaborative learning experiences.

We need a tool that makes it easy to annotate information on webpages, removes all the friction of copy-pasting links and text on a note app, and facilitates sharing and collaboration among users. A tool that saves the exact information you need

on a webpage, keeps track of the location of the webpage, and fosters interactive learning experiences by allowing users to share their insights with others.

1.2 Background

Every time we read, we have a habit of highlighting and writing notes in the margin. Additionally, we often engage in discussions and share our insights with others to enhance our understanding of the subject matter. Although highlighting, writing notes, and sharing insights in the margins of a text is simple on paper and in print, replicating the experience online remains tricky. A web annotation tool with social annotation features allows users to highlight text, write notes, and share insights as they browse the web, as they do on paper.

When we highlighted some text from the e-book on iBooks and the Kindle, we greatly enjoyed the user experience, so we wanted to recreate the user experience, but for web pages instead of books. Additionally, we thought the Google Docs commenting and collaboration features could also be replicated and incorporated into our web annotation tool. This would create an interactive learning environment where users can not only annotate web content but also share their annotations with others, fostering collaboration and enriching their online experience.

1.3 Significance of the project

We are building a web annotation and social collaboration tool called Luminate. Luminate is a browser extension that allows users to highlight text, leave notes, and share insights on any website. The highlights and notes made by the users will remain on the site as long as the browser extension is turned on. By allowing users to create, store, and share notes and highlights, we provide a means for users to organize, manage, access, and collaborate on their online information more effectively.

This tool will be helpful for users who frequently conduct research by reading a large number of online papers or other online resources, as well as for those who wish to engage in meaningful discussions and share perspectives with others. Users can simply highlight relevant parts, add notes, set their annotations as public or private, and follow other users to view their shared insights. They can also see everything they marked all at once on our web app, allowing them to easily access content that is important to them, which can help them in their studies, work, or even just for entertainment.

By incorporating social annotation features, Luminate fosters interactive learning experiences and enriches users' online experience, transforming it into a collaborative learning platform. All in all, Luminate is an innovative tool that aims to provide a better way to take notes, search, review, and collaborate on notes on the internet.

1.4 Objective

The overall goal of the project is to design and build a web annotation tool, which is made up of 2 components: a chrome extension and a web application. Both of them serve different functionalities.

Chrome Extension Functionalities	**Web App Functionalities**
• Highlight and add notes on any webpage on the internet • Save these annotations to the cloud • View personal annotations and friend’s public annotations directly on the webpage • Set annotations as public or private.	• View all saved links and annotations made • Organize saved links and annotations • Follow other users and view their public annotations

2. Design

2.1 Sequence Diagram

2.1.1 Basic Features

The sequence diagram below illustrates the basic functionality of our app.

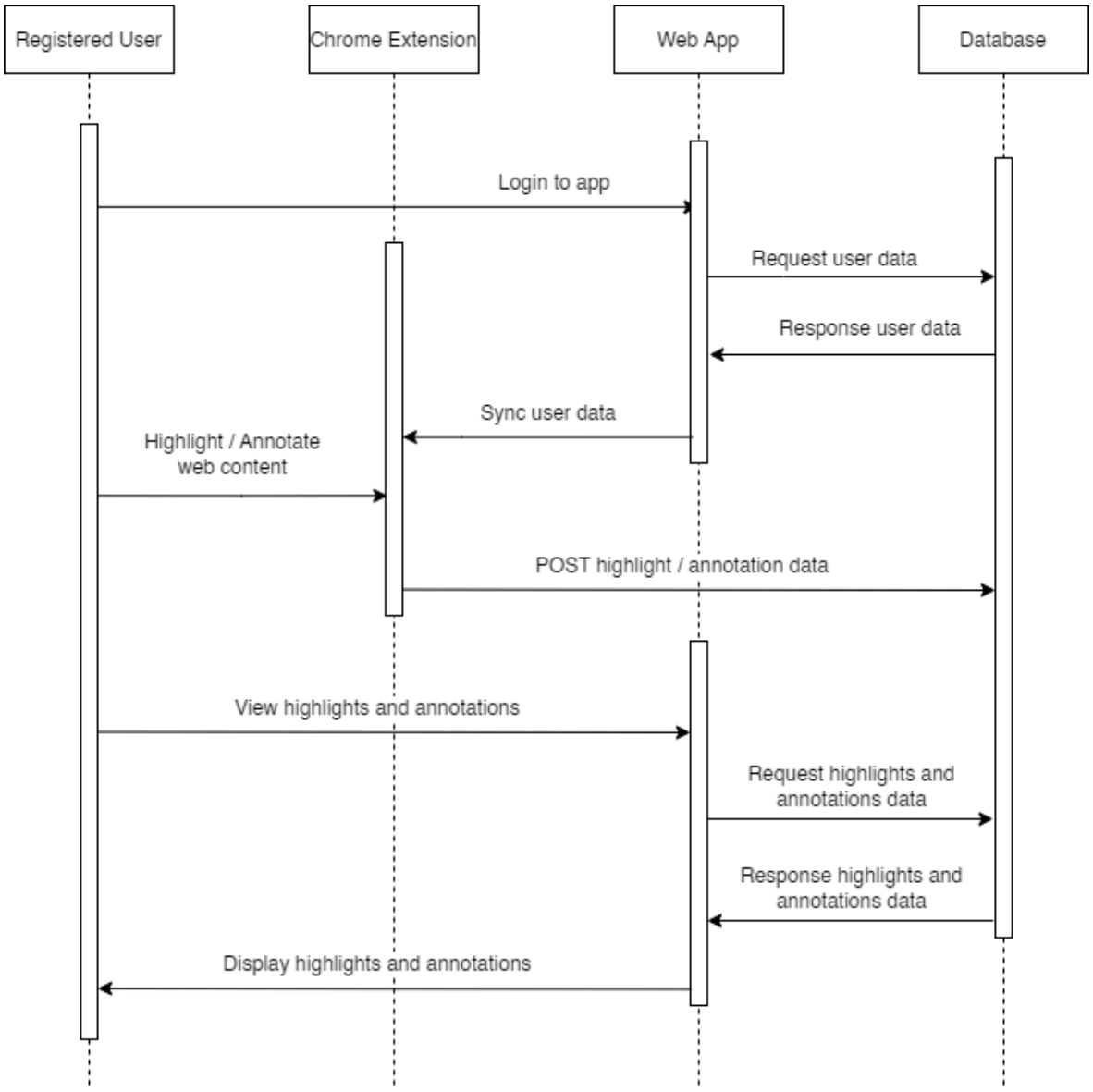


Figure 2.1.1.1 Luminate Basic Feature Sequence Diagram

2.1.2 Social Annotation Feature

The sequence diagram below illustrates the social annotation feature of our app.

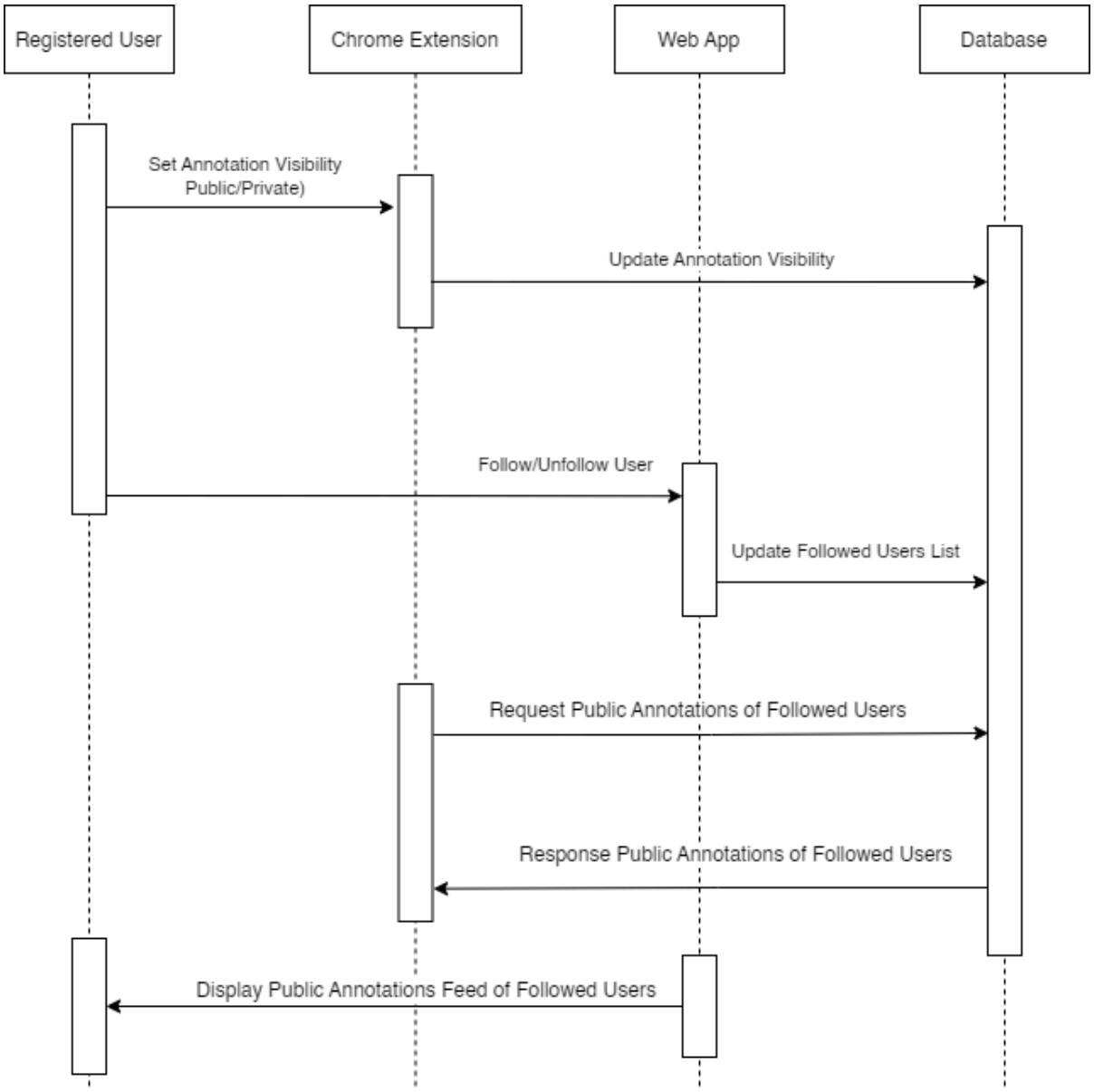


Figure 2.1.2.1 Luminate Social Annotation Feature Sequence Diagram

2.2 Basic Features

2.2.1 User Registration

Because our app requires storing user activity and data in the cloud, only registered users are allowed to use our app.

When the user clicks the register button on the main page, they will be taken to the user registration page. Users will be required to provide their email address and name and also set a password for their account. Optionally, they can also upload a profile picture. A confirmation email will then be sent to the user afterwards, which includes a verification step that they have to complete. After the verification process, their account is successfully created.

An alternative is to use Google account authentication. Using this, users don't have to provide any information, as the data mentioned about will be extracted from their existing Google account they used to connect.

2.2.2 User Login

Registered users can directly log in to our app by providing their email address and password, or log in by using their Google account if they previously registered with their Google account. The information will be passed to the Firebase Authentication system to determine if the login is successful. If successful, the user will be directed to the app afterwards.

2.2.3 Save and Unsave Page

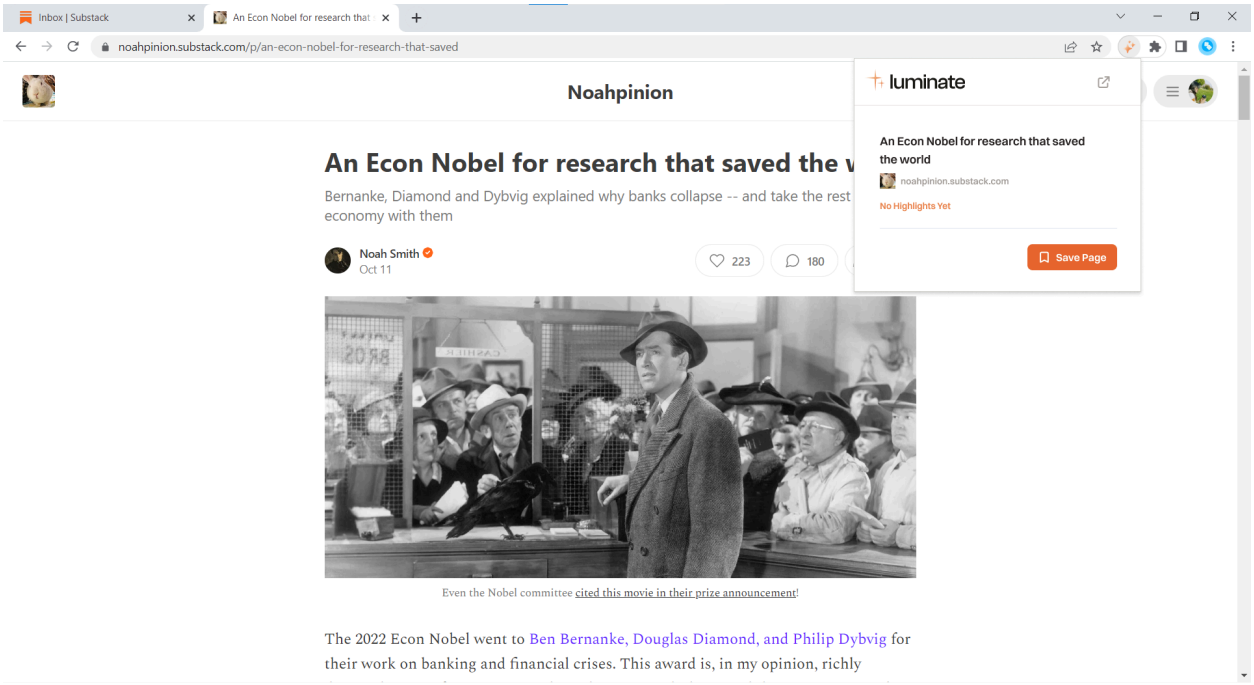


Figure 2.2.3.1 Save Page Extension Pop-Up UI

The **Save Page** functionality of our app works the same way as the **Bookmark** functionality available in all browsers.

To save a page, users can simply click the Luminate icon in the Google Chrome Extensions Toolbar. A popup will appear with a ‘Save Page’ button.

After clicking the ‘Save Page’ button, the page will be saved into the user account’s database. The button will immediately change to ‘Unsave Page’, in which users can click to unsave the page if they had mistakenly saved a page they did not mean to.

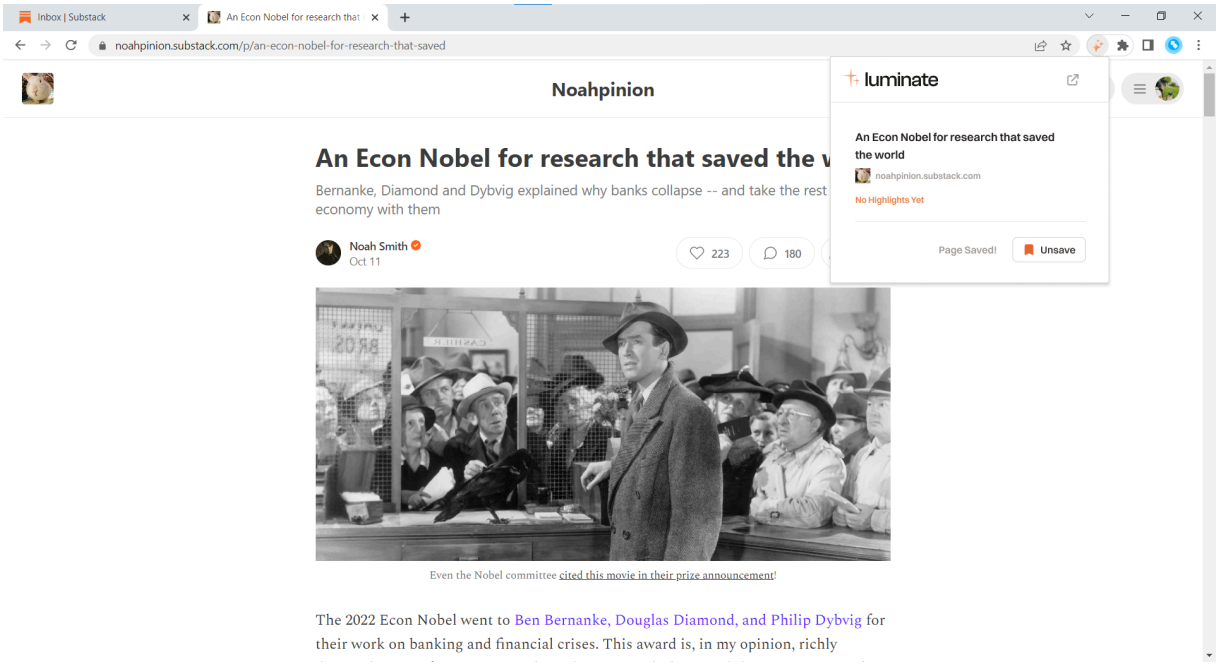


Figure 2.2.3.1 Unsave Page Extension Pop-Up UI

Saved pages can be viewed and deleted through the web app. Deleting a saved page will also delete the highlights on that webpage from users account.

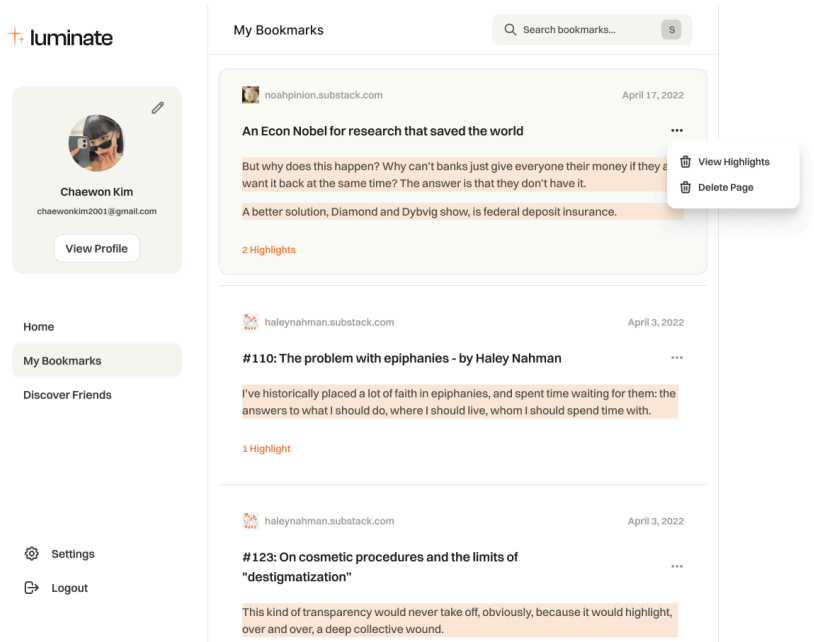


Figure 2.2.3.2 Delete Page Web App UI

2.2.4 Highlight Items and Save

Highlighting items is one of the main features of our app. Users can highlight items on a webpage by making sure that the extension is active and turning on the highlighter on that specific webpage.

To turn on the highlighter, there are 2 ways:

- Right-click and click the 'Turn on highlighter' context menu

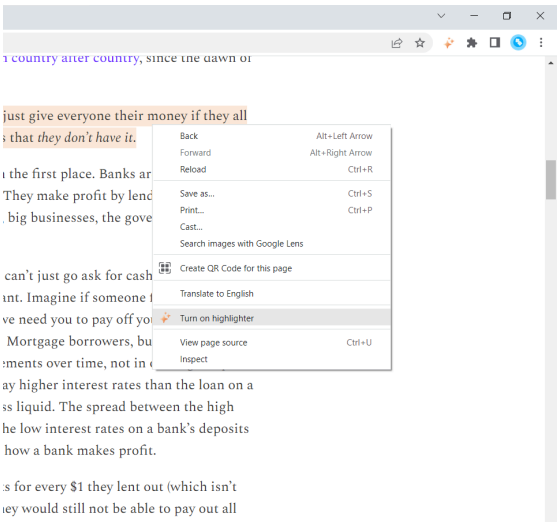


Figure 2.2.4.1 Turn on Highlighter Context Menu

- Press "Alt + H" on the keyboard

When the highlighter is turned on, the cursor will turn into a highlighter icon as shown below. Users can then select items on the webpage with the highlighter cursor, and the items will be highlighted. We try to mimic how highlighters behave in real life.

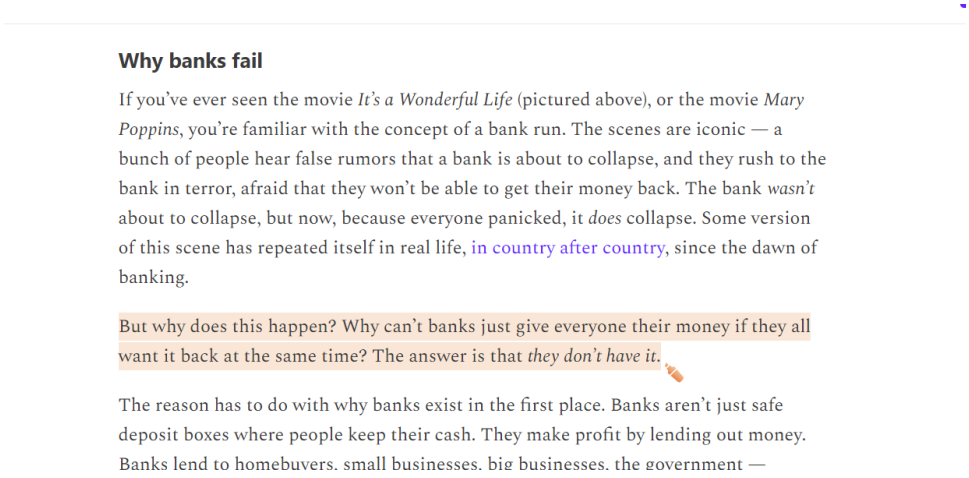


Figure 2.2.4.2 Highlighter Cursor UI

After turning off the highlighter, the cursor will immediately return to its original state. To turn off the highlighter, users can:

- Right-click and click the 'Turn off highlighter' context menu

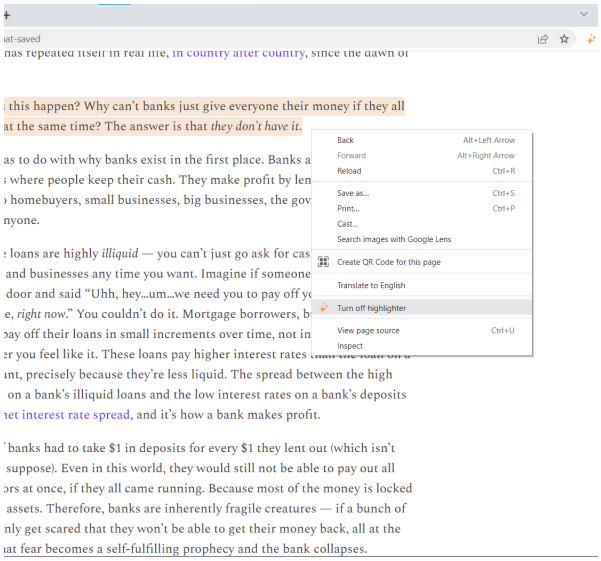


Figure 2.2.4.3 Turn off Highlighter Context Menu

- Press "Alt + H" on their keyboard

Highlighting an item on a webpage will immediately save the page to the app.

2.2.5 Delete Highlights

There are 2 ways for users to delete their highlighted items:

- Delete highlights on a particular webpage: Users can hover over their highlight in the webpage, and a popup will appear. The popup has a delete button which the user can click if they want to delete the selected highlight.

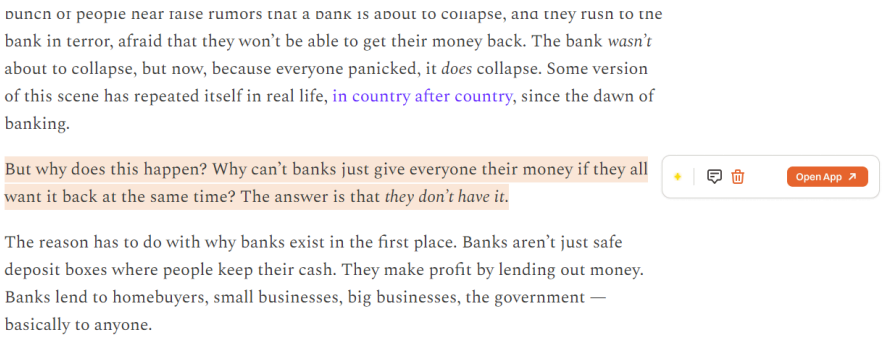


Figure 2.2.5.1 Delete Highlight on a Webpage UI

- Delete highlights on our web app: Users can go to the web app and view their highlights, they can also delete all the highlights that they want to delete.

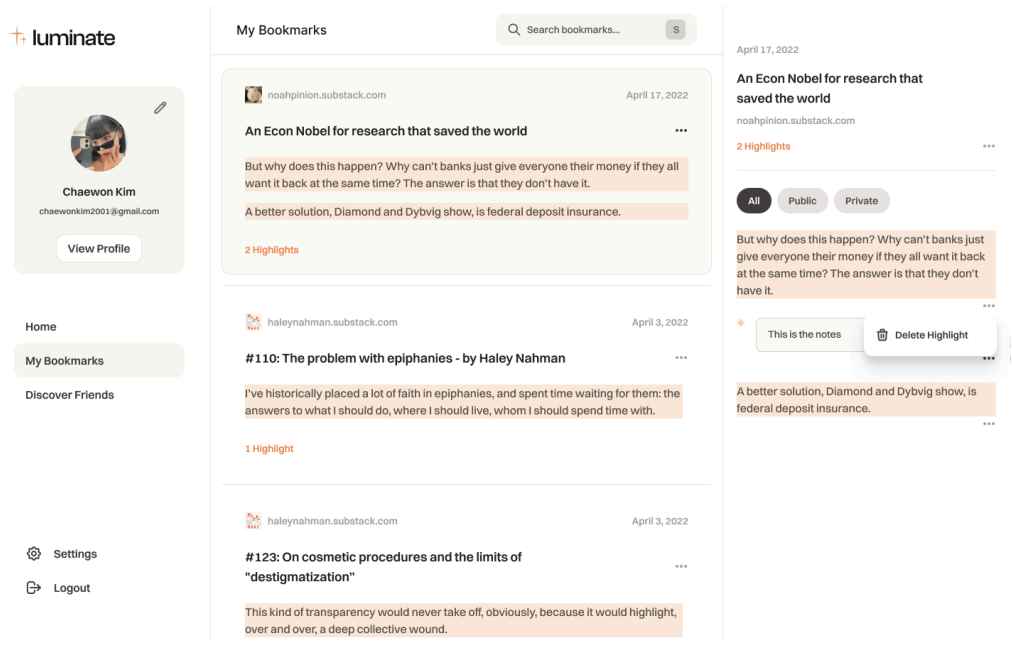


Figure 2.2.5.2 Delete Highlight in our Web App

2.2.6 Add Annotations

Adding annotations is one of the main features of our app. To annotate an item on a webpage, users must first highlight the text. A popup will appear when the user hovers over the highlight.

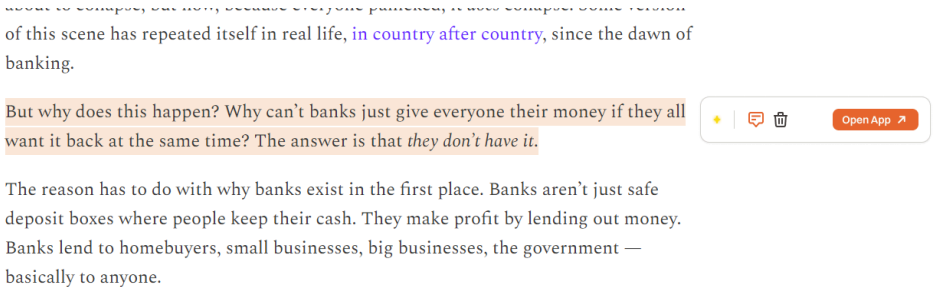


Figure 2.2.6.1 Annotate on a Webpage UI

Users can click the annotate button, and a text box will appear. Users can then write notes and save them.

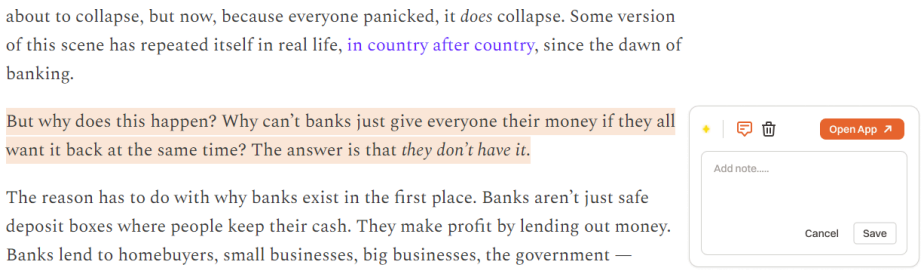


Figure 2.2.6.2 Annotate Text Box UI

The annotation will be associated with the highlighted text.

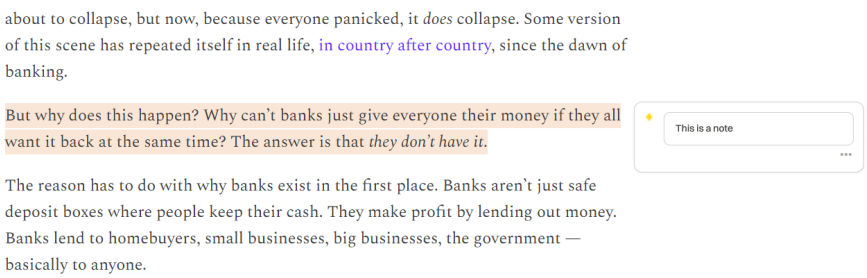


Figure 2.2.6.3 Annotations UI

2.2.7 Delete and Edit Annotations

There are 2 ways for users to delete or edit their annotations:

- Delete or edit annotations on a particular webpage: There will be a three dots button under the annotations on a webpage. Users can click that button and then click ‘Delete Note’ to delete the annotation.

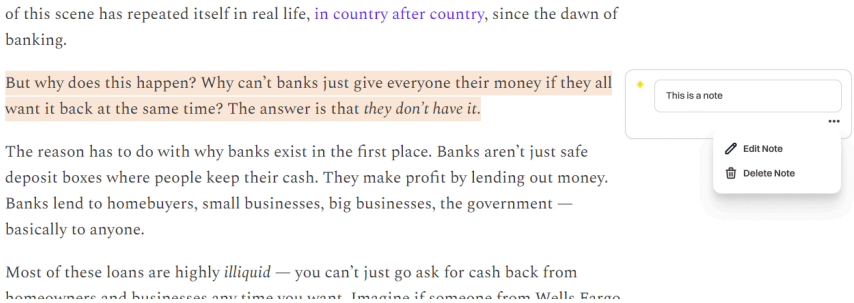


Figure 2.2.7.1 Delete or Edit Note on a Webpage UI

- Delete or edit annotations on our web app: Users can go to the web app and view their annotations, they can click the three-dot button under the annotations and click ‘Delete Note’ to delete the annotation.

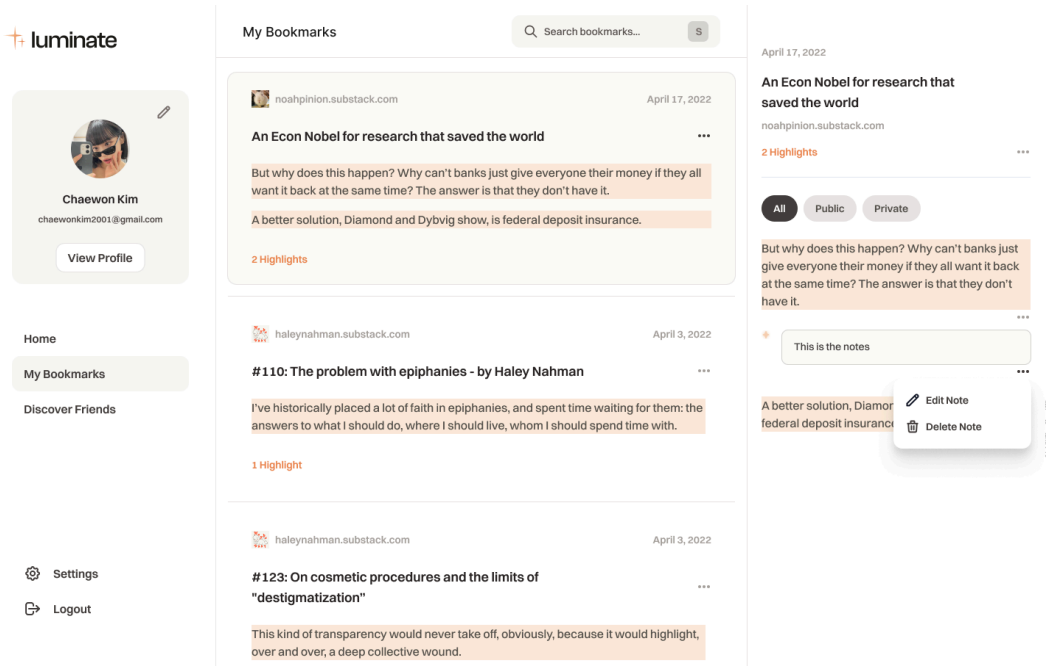


Figure 2.2.7.2 Delete or Edit Note in our Web App UI

2.2.8 View saved pages and highlights

Users' highlights and annotations can be viewed all at once on our web app.

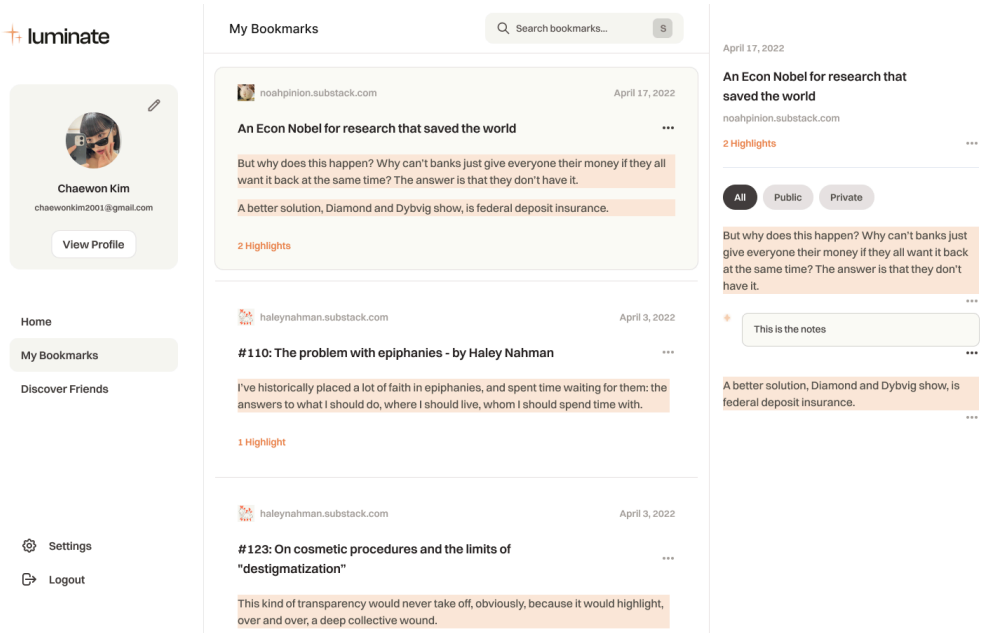


Figure 2.2.8.1 View Pages and Highlights UI

2.2.9 Search saved pages

Users can search for a query on all of their saved pages using the search bar at the top of the page.

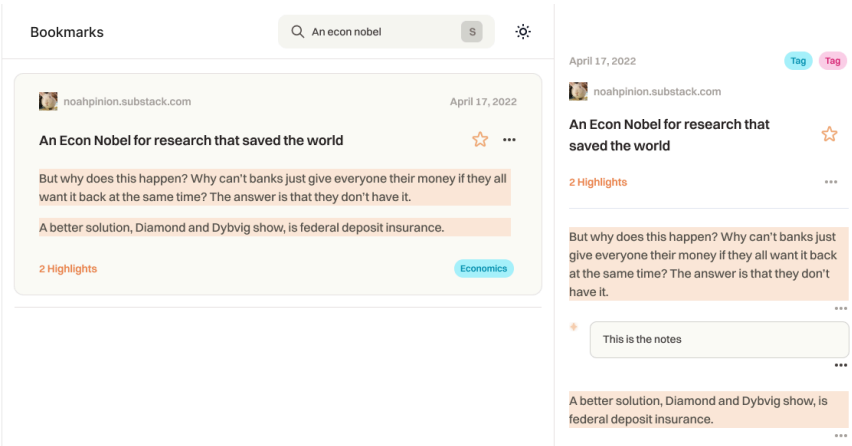


Figure 2.2.9.1 Search Saved Pages UI

2.3 Social Annotation Features

2.3.1 Set Highlights and Annotations as Public or Private

With the new social annotation feature, we provide users with the ability to control the visibility of highlights and annotations. Users can choose to set their annotations as public or private based on their preferences. By default, all annotations are set to private, ensuring that the user's notes are protected and visible only to them.

To set an annotation as public or private, users can click on the visibility toggle within the annotation editor on the webpages. Public annotations can be viewed on the user’s profile page on the webpage or the URL webpage of the annotation by other users who follow the creator or come across the shared annotation, thereby fostering collaboration and interactive learning.

2.3.2 Follow and Unfollow Users

Luminate allows users to follow and unfollow other users to create a personalized network of shared knowledge. By following other users, one can view public annotations made by those users, promoting collaborative learning and the exchange of ideas.

To follow or unfollow a user, one can visit the user's profile page and click the "Follow" or "Unfollow" button. This action will update the user's network, and any new public annotations made by the followed user will appear in the follower's feed and on webpages where the followed user has made their annotations.

2.3.3 View Public Annotations of Followed Users

Once a user follows another user, they can view the public annotations made by the users they follow. This feature encourages users to learn from others' perspectives and engage in meaningful discussions on various topics.

The public annotations of followed users can be viewed through:

- The web app
- The webpages where the followed user has made their annotations

In the web app, they can be viewed through the follower's feed, where a stream of public annotations made by the followed user is displayed. They can also be viewed through the followed user's profile page.

Followed users' highlights and annotations will also appear on the specific webpages where they have made annotations.

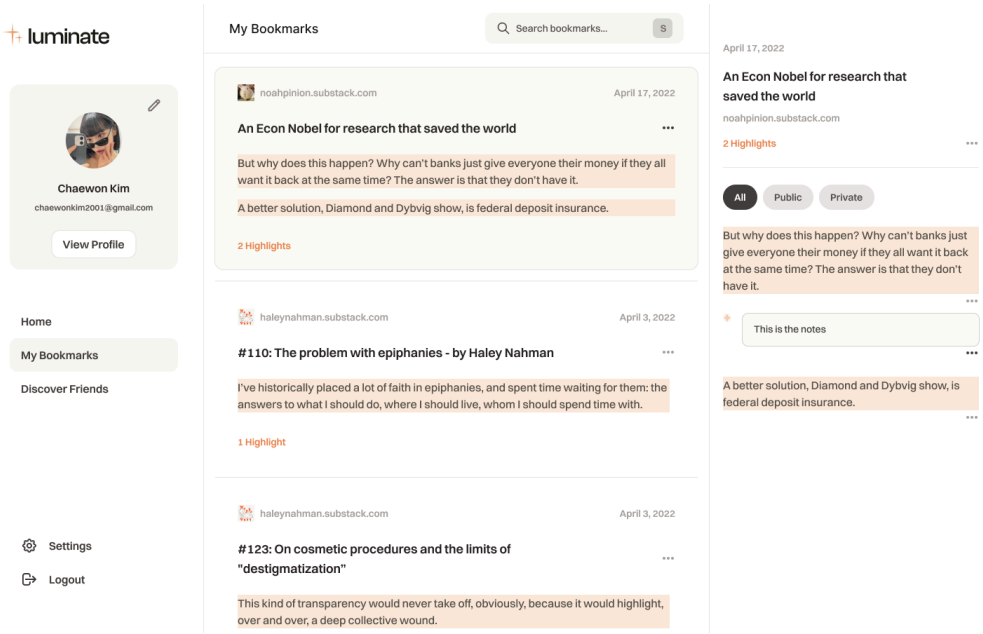


Figure 2.3.3.1 User Page UI

By accessing the perspectives and insights of those they follow, learners expand their knowledge, expose themselves to new ideas, and foster an intellectually vibrant community. The annotation feed and profile pages thus serve as touchpoints for collaborating, debating issues, and building shared understanding.

3. Technical Specifications

3.1 Tech Stack

Luminate consists of three main components, which include a Chrome extension, a web application and a back-end to facilitate data storage and user authentication. Through careful considerations on enhancing the scalability and robustness of the product, Luminate adopts the following stack:

Chrome Extension	
Extension Platform	Manifest V3
Front-End	ReactJs
	TailwindCSS
Utils	Webpack

Web Application	
Front-End	NextJs with Typescript
	TailwindCSS

Back-End	
Database	Cloud Firestore
User Authentication	Firebase Authentication

3.1.1 Manifest V3

Google Chrome extensions are created using the Manifest V3 application programming interface (API) [2]. These extension Manifest control how Chrome extensions communicate with your browser. It provides the browser with details about the extension, including the most crucial files and potential functionalities the extension might use [3].

In comparison to the most recent version, Manifest V2, the current version (V3) of the Chrome Extension platform includes a number of additional features. The security, privacy, and functionality of the Chrome Extension experience have been improved with the addition of these new features [3]. The following are some of the significant updates:

Service workers

Background pages, which are used in Manifest V2, will be swapped out for Service workers in the next version. The service worker is often in charge of maintaining the cache, preloading resources, and allowing offline web pages in standard online applications. All of these functionalities are included in the new service worker in Manifest V3 and will be accessible even when offline.

Network Request Modification

The new declarativeNetRequest API will take care of network request adjustments. With the help of this new API, extensions may efficiently and privately alter and stop network requests. This API's main purpose is:

- Instead of intercepting and procedurally altering a request, the plugin asks Chrome to assess and alter requests on its behalf.
- The extension defines a set of guidelines: patterns to match requests against and actions to do when matched. The browser then adjusts network requests in accordance with these criteria.

Using this declarative approach dramatically reduces the need for persistent host permissions.

Remotely Hosted Code

Extensions can import external JavaScript or Wasm files using Manifest V2. This creates a serious security issue since it makes it simple to inject malicious scripts into the user's browser. A significant security enhancement provided by Manifest V3 is the requirement that all script logic be contained within the extension's package.

Promises

Async/await and promises both now have complete support. A more complete user experience is made possible by preventing the promise from being returned when using an API method.

As of January 2023, Manifest V3 has officially been released. By June 2023, extensions that run Manifest V2 will no longer be supported [5]. Hence, with all the new features Manifest V3 has to offer and the risk of imminent deprecation, we believe that it is imperative for Luminate to adopt Manifest V3.

3.1.2 ReactJS

Web apps are increasingly being created with more contemporary web development frameworks rather than traditional JavaScript in recent years. ReactJS is one prominent contemporary framework. React JS is a JavaScript library that uses declarative programming and was created and is maintained by Facebook. It is a compact JavaScript framework that may be used to construct reusable UI components as well as user interfaces [6]. Important characteristics include:

Reusable Custom Components
Components are independent and reusable pieces of code. Modularizing scripts and HTML elements into components encourages code reusability.

Fast Rendering
Initial rendering and re-rendering are the two methods of rendering. When a component first shows on the screen, initial rendering takes place, and re-rendering occurs when React has to update the app with new data. ReactJs enables quick rendering by only updating components that need to be updated because of changes to the state, parent, context, or hook.

3.1.3 TypeScript

By enabling developers to include type safety in their projects, Typescript enhances the JavaScript experience [7]. For instance, function parameters and variables in JavaScript do not specify the kind of desired input. This can take time and be error-prone because developers must consult the documentation. When the kinds of data do not match, TypeScript offers the ability to report errors and allows the user to declare the types of data that are being passed around within the code.

Hence, to adopt only the best development practices with the goal of creating a robust product, the Luminate web app is built using the TypeScript framework instead of ordinary JavaScript.

3.1.4 NextJS

We are utilizing NextJs in addition to ReactJs to create our web application. By managing the tooling and setup required for React, as well as adding extra structure, functionality, and optimizations for the application, NextJs is a React framework that enables developers to create web apps more quickly. With the help of Next.js features, common application requirements, including routing and data fetching, may be implemented significantly more easily [8].

The ability for developers to create hybrid applications with both statically generated pages and server-side rendered pages is one of NextJs' key advantages. Developers can give users basic indexable HTML by using server-side rendering (SSR), which allows them to render JavaScript code on the development server. Compared to using client-rendered JavaScript, this not only makes the page load faster but also increases visibility in the search engine.

3.1.5 TailwindCSS

Luminate will use TailwindCSS [9] to facilitate its front-end styles for both the Chrome extension and the web app in place of vanilla CSS. Vanilla CSS frequently creates a lot of bother because it can be difficult to correlate the effects of a certain .css style to a particular class in the HTML. This problem is resolved by TailwindCSS, which develops utility classes with a predefined set

of features that make it simple to incorporate pre-existing classes into the HTML code. More specifically, some of its qualities are as follows:

Customizability

Despite having a default setup, it is easy to alter it and apply our own unique styles. Colour schemes, styling, spacing, themes, etc. may all be easily customized thanks to the configuration file. The configuration file enables easy customization of colour palettes, styling, spacing, themes, etc. TailwindCSS allows the flexibility to replicate any UI design that any traditional CSS can do.

Utility Patterns

Tailwind CSS eliminates the hassle of naming classes. The availability of common utility patterns provides solutions to many issues, including class specification, class organization, class cascading, and a host of other issues. Making custom components is made easier by utility classes. With Tailwind CSS, there is no requirement for hard coding.

Responsive Layouting

The Tailwind CSS framework has a mobile-first strategy by default. The accessibility of utility classes facilitates the construction of sophisticated responsive layouts. Every TailwindCSS style can be applied conditionally at different breakpoints, hence making it easy to build responsive interfaces.

3.1.6 Webpack

Webpack [10] is a module bundler that parses through our code and builds what it refers to as a dependency graph, which is made up of numerous modules that our web app would need to work as intended. Then, based on this graph, it generates a new 'build' package with the absolute lowest number

of files needed, frequently just a single bundle of.js and.html files as specified in the webpack configuration.

Extensions built on top of ManifestV3 follow a strict file structure and only interpret code written in vanilla HTML, CSS and JavaScript. Since we have decided to use modern web development frameworks and tools such as React, TypeScript and TailwindCSS, we utilize webpack to load and compile all of the code to only a few .js and .html file extensions that are readable by the Manifest API.

3.1.7 Firebase

Firebase is a Backend-as-a-Service (Baas) that is provided by Google [11]. It offers a range of tools and services to developers so they can create high-quality apps and scale as their user base expands. Depending on the requirements of the project, the Firebase development toolkit can be used separately. For Luminate, we use Firebase Authentication for user authentication and account management along with CloudFirestore as our real-time database for synchronizing user data.

Cloud Firestore

Cloud Firestore is a scalable database from Firebase and Google Cloud for servers, browsers, and mobile applications [12]. It supports offline support for mobile and web so developers can create responsive apps that function regardless of network delay or Internet connectivity. It synchronizes data among client apps via real-time listeners.

Although Cloud Firestore and MongoDB are both well-known NoSQL databases, Luminate chooses Cloud Firestore since our application will not

require complicated queries or aggregations. Instead, since query scaling would be essential, Cloud Firestore is a strong fit.

Firebase Authentication

In addition to enabling phone authentication, Google, Twitter, Facebook, and GitHub logins, Firebase Authentication is a secure authentication system that offers an end-to-end identity solution [13]. It also comes with an open-source UI library that simplifies the creation of the numerous auth processes necessary to provide users with a positive experience. Common flows include things like login hints, account linking, and password resets.

We choose to use Firebase Authentication not only because of the wide range of login support and built-in flows, but we find the APIs to be much more interoperable given that we are making use of Firebase's Cloud FireStore for the database.

3.2 Project Architecture

The following architecture diagram demonstrates how all components and the corresponding tech stack interact with one another:

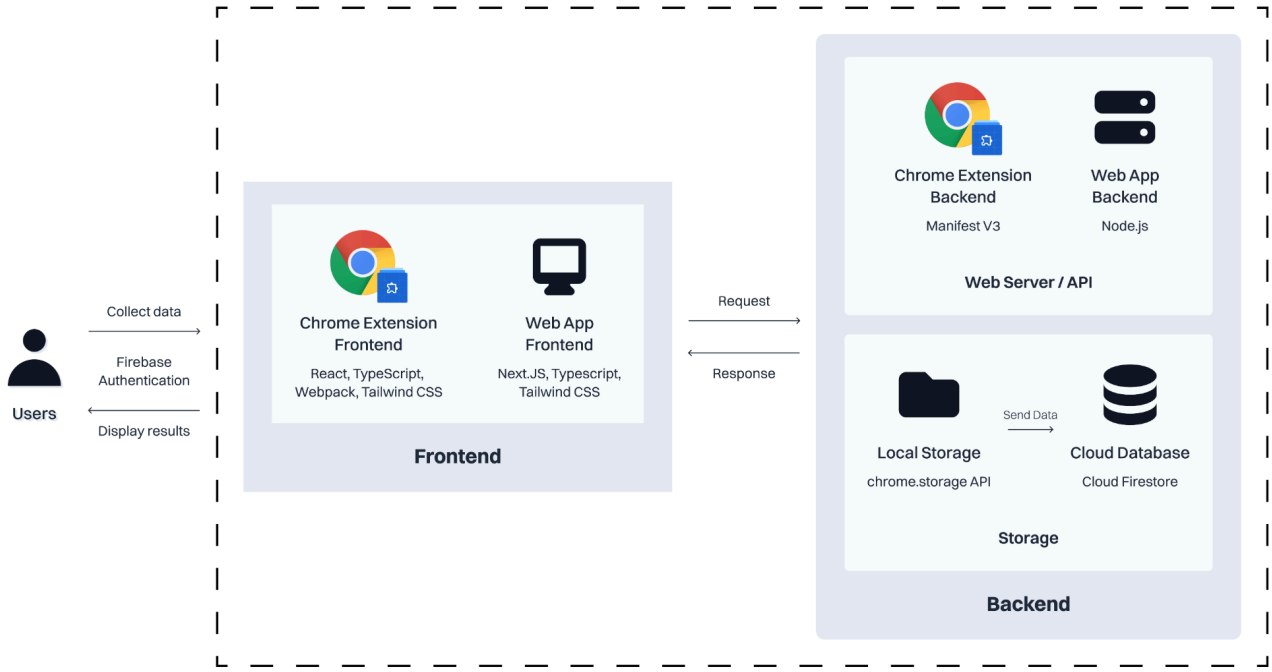


Figure 4.2.1 Application Architecture

We believe that our chosen tech stack and architecture is the best way to go about developing our product. Factors taken into consideration include the scalability of the application, ease of implementing new functionalities, quality of software and the costs of maintenance and service.

4. Implementation

4.1 Chrome Extension Architecture

The execution environment of Chrome extensions is separate from the JavaScript inside the regular web page. Within this separated environment, any Chrome extension follows a specific architecture as defined by the Manifest V3 API [14]. Four major components relevant to the development of Luminate include:

1. **Manifest.json File**

   The manifest file, with the filename manifest.json, provides the browser with details about the extension, including the most crucial files and any potential permissions.

   An example of Luminate's manifest.json

```
{
  "name": "Luminate Extension",
  "description": "An extension to save highlights!",
  "version": "1.0.0",
  "manifest_version": 3,
  "icons": {
    "16": "icon.png",
    "48": "icon.png",
    "128": "icon.png"
  },
  "action": {
    "default_popup": "popup.html",
    "default_title": "Luminate Extension",
    "default_icon": "icon.png"
  },
  "permissions": ["storage", "tabs", "contextMenus", "unlimitedStorage"],
  "options_page": "options.html",
  "background": {
    "service_worker": "background.js"
  },
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": ["contentScript.js"]
    }
  ],
  "web_accessible_resources": [
    {
      "matches": ["<all_urls>"],
      "resources": ["images/*.png", "images/*.svg"]
    }
  ],
```

```
  "commands": {
    "toggle-highlight": {
      "suggested_key": "Alt+H",
      "description": "Turns highlighter on or off"
    }
  }
}
```

2. **Background Scripts**
   Without any accompanying web page or even access to any DOM, background scripts execute in the browser process's background, typically in response to a content script.

   In our case, the background script is responsible for listening to commands from either keyboard shortcuts or context menus. It will then send the appropriate instruction to the content script. For example, the key shortcut “Alt + H” will send a message to the content script to toggle the highlighter on or off.

3. **Content Scripts**
   Content scripts give the extension the ability to read and alter a page's contents by injecting logic into it. JavaScript code that is contained in a content script is executed while a page is loaded into the browser.

   In our case, the content script is responsible for preparing the highlighter, highlighting and saving the new highlight or annotation to storage. This script is also responsible for retrieving all highlights for the page and highlighting the page upon refresh.

4. **Popup Page**
   When a user hits the action icon, an HTML file known as a popup is shown

in a small separate window. In Luminate, the Popup page is responsible for showing details of the current page, as well as highlights and annotations of the current page, retrieved from storage.

Another major component for the extension would be the storage capability. When used offline, our extension will store all necessary data using the chrome.storage API which utilizes the local browser memory. Meanwhile, for cloud storage, we make use of cloud firestore by Firebase. The following diagram further demonstrates how all of these components communicate.

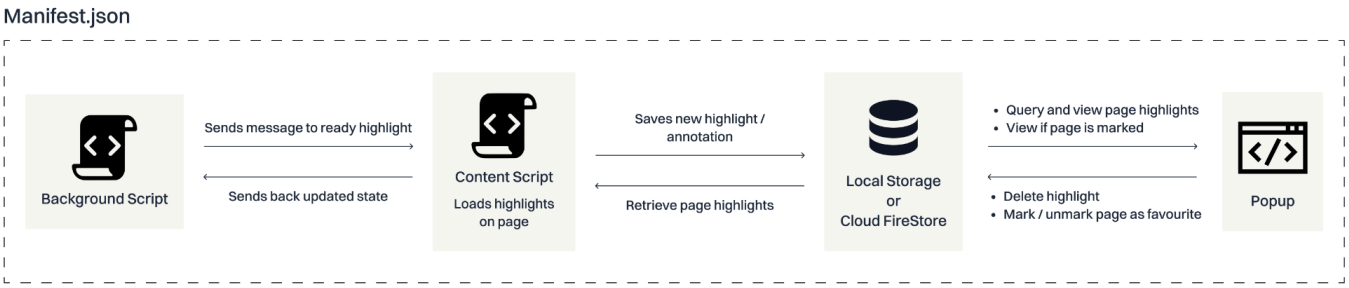


Figure 5.1.1 Chrome Extension Architecture

4.2 Next.js Implementation

We have built our web application using Next.js, a popular React framework that provides a range of benefits for server-side rendering. This approach allows for faster load times and improved SEO optimization, enhancing the user experience. Additionally, Next.js provides a range of built-in features that simplify API routing, making it easy to handle requests from the client in a secure and effective way.

Authentication with getServerSideProps and cookies

To ensure secure user authentication, we store session tokens as cookies in the user's browser. These tokens are retrieved using Next.js' getServerSideProps method, which fetches data from the server before

rendering the page. This ensures that users are validated on the server-side before sensitive data is rendered to the user. By using cookies for authentication, our web application is able to maintain user sessions across multiple pages, avoiding the need for repeated logins.

API Routes

Our web application also leverages Next.js API routes to handle requests from the client. These server-side functions can be accessed via HTTP requests and are designed to provide enhanced security and better separation of concerns between client and server code. By creating an API route within the /pages/api directory, we can easily create a range of server-side functions to handle HTTP requests, simplifying the development process and improving overall performance.

The API routes used by our webapp is as follows:

```
-api
  |-highlights
  |  |-delete-url.ts
  |  |-delete.ts
  |  |-edit-annotation.ts
  |  |-get-by-user.ts
  |  |-toggle-visibility.ts
  |
  |-users
  |  |-get-user.ts
  |  |-get-users.ts
  |  |-register.ts
  |  |-follow.ts
  |  |-unfollow.ts
  |
  |-feed.ts
```

```
|-get-batch-metadata.ts
|-url-metadata.ts
```

4.3 Database Model

We implement two collections for our Firestore data model. The two collections and its content are as follows:

Collection: Users

Field	Type	Description
userId (pk)	String	Unique identifier for the user document.
email	String	Email address of the user.
displayName	String	Display name of the user.
displayPicture	String	URL of the user's display picture.
bio	String	Short biography of the user.
website	String	URL of the user's website.
followers	Array<String>	Array of user IDs that are following this user.
following	Array<String>	Array of user IDs that this user is following.
savedPages	Array<String>	Array of URLs of pages that this user has saved.

Collection: Highlights

Field	Type	Description
uuid (pk)	String	Unique identifier for the highlight document.
selectionString	String	The selected text of the highlight.

selectionLength	Number	The length of the selected text.
container	String	The HTML element that contains the selected text.
color	String	The color of the highlight.
anchor	String	The anchor position of the highlight.
anchorOffset	Number	The anchor offset of the highlight.
focus	String	The focus position of the highlight.
focusOffset	Number	The focus offset of the highlight.
url	String	The URL of the webpage where the highlight was made.
userId	String	The user ID of the user who made the highlight.
date	Timestamp	The timestamp of when the highlight was made or last updated.
note	String	The note associated with the highlight.
isPublic	Boolean	Boolean value indicating whether the highlight is public or not.
snapshotHash	String	The hash value of the snapshot of the webpage where the highlight was made.

4.4 Recursive Highlighting

Highlighting text on a web page is tricky as the highlighted text users select can easily span across multiple different node elements or even at different levels in the DOM tree. To solve this problem, we figured out that the best way to highlight and load saved highlights involves a recursive algorithm that iterates that makes

extensive use of the built-in web APIs. Our algorithm takes inspiration and is a more compact version of another open source highlighting tool [15].

The pseudocode of our algorithm is as follows:

```
** DRIVER FUNCTION **

SELECTION selection = WINDOW.getSelection()
RANGE selectionRange = selection.getRange()
NODE parentContainer = selection.getCommonAncestorContainer()
STRING selectionContent = selection.getString()
INTEGER selectionLength = Length(selectionContent)

// SEE REFERENCE BELOW
NODE startNode = selectionRange.startContainer()
INTEGER startOffset = selectionRange.startOffset()
NODE endNode = selectionRange.endContainer()
INTEGER endOffset = selectionRange.endOffset()

STRING id = RandomID()

CALL Highlight (
      container: parentContainer,
      selectionContent: selectionContent,
      selectionLength: selectionLength,
      startNode: startNode,
      startOffset: startOffset,
      endNode: endNode,
      endOffset: endOffset,
      id: id,
      charsFound: 0,
      startFound: FALSE
)
** RECURSIVE HIGHLIGHTER **

FUNCTION Highlight (
      container: NODE,
      selectionContent: STRING,
      selectionLength: INTEGER,
      startNode: NODE,
      startOffset: INTEGER,
```

```
      endNode: NODE,
      endOffset: INTEGER,
      id: STRING,
      charsFound: INTEGER,
      startFound: BOOLEAN
) RETURNS (charsFound: INTEGER, startFound: BOOLEAN) {

      // Iterate sideways through all elements under same parent node
      FOR childNode in container AS child {

            // RECURSION END CASE: All characters highlighted
            IF (charsFound >= selectionLength):
                  RETURN

            // Check if current child is a text node
            IF (child.nodeType IS NOT TEXT_NODE):
                  // If not a text node, we dive deeper in the DOM tree
                  // by calling the recursive highlight function on the
                  // current child
                  charsFound, startFound = Highlight(child, ...)
                  RETURN

            // If text node, we scan if there is any element to highlight
            let startIndex;
            // Check if current child is startNode or start has been found
            IF child == startNode OR startFound {
                  // Enforce startFound to always be True in this case
                  startFound = TRUE

                  // If still no characters have been highlighted,
                  // we are in the startNode
                  // and should start from startOffset
                  IF charsFound > 0:
                        startIndex = 0
                  ELSE:
                        startIndex = startOffset

                  // Start iterating through characters starting
                  // from offset to the rest of characters
                  // of text in current child node
                  FOR (i = startIndex; i < Length(child.value); i++) {
                        // Check if done highlighting
```

```
                        IF (charsFound >= selectionLength):
                              RETURN

                        // Compare character to make sure we are
                        // highlighting the correct element
                        IF (selectionContent[charsFound] == child.value[i]):
                              charsFound++

                        // Increment index and move to next character
                        i++
                  }
            }

            // For example, if node is <>abchildcd<> and 'child' is selected
            // then startIndex = 3 hence selectedElement is <>children<>
            // then previous loop would stop at the end of 'child' and i = 8
            // hence selectedElement will be <>child<>
            // and rightElement will be <>ren<>
            NODE selectedElement = child.splitText(startIndex)
            NODE rightElement = selectedElement.splitText(i - startIndex)

            // Prepare new but modified HTML Highlight element
            NODE highlighted = new NODE('span')
            highlighted.addCSS(backgroundColor: COLOR)
            highlighted.text = selectedElement.value
            // Remove original element and replace it with modified
            // Highlight element
            selectedElement.remove()
            container.insertBefore(highlighted, rightElement)

      }

      RETURN (charsFound, startFound)
}
```

* Reference:

Let us take an example of a simple text that spans multiple elements. When highlighting ‘is is just an exa’, the defined properties of the selection will be as follows [16]:

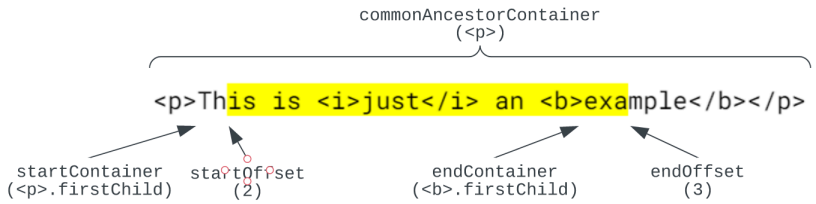


Figure 5.2.1 Web API Illustration

4.5 Dynamic Webpage Detection

When creating a web application that allows users to highlight and annotate webpages, one major challenge is the dynamic nature of webpages. If a page changes between the time a user makes a highlight and revisits the webpage, some highlights and annotations may no longer appear. To avoid this issue, we need a way to detect when a webpage has changed since the last time the user accessed the site.

One solution we came up with is to hash all the elements of the webpage at the time the user makes a highlight and store it in the database. Then, when the user visits the webpage again, we can compare the new hash with the stored hash to check if any changes have been made.

To ensure the hash value remains consistent regardless of metadata changes, we exclude the id and class attributes from the hash calculation. We accomplish this by using a DOM tree traversal algorithm to visit all the nodes in the DOM tree and extract only the relevant data we want to include in the hash. This data includes the tag name, text content, and value of form elements.

Here is the function we developed for hashPage that excludes the id and class attributes:

```
function hashPage() {
  const hash = CryptoJS.algo.SHA256.create();
```

```
  // Traverse the DOM tree and extract the relevant data for hashing
  function traverse(node) {
    // Skip nodes that should be ignored
    if (node.nodeType === Node.COMMENT_NODE ||
        node.nodeType === Node.PROCESSING_INSTRUCTION_NODE ||
        node.nodeType === Node.DOCUMENT_TYPE_NODE) {
      return;
    }

    // Extract the relevant data for hashing
    const tagName = node.nodeName.toLowerCase();
    const textContent = node.nodeType === Node.TEXT_NODE ? node.textContent.trim()
: '';
    let value = '';
    if (tagName === 'input' || tagName === 'textarea' || tagName === 'select')  {
      value = node.value.trim();
    }

    // Hash the relevant data
    hash.update(`${tagName}:${textContent}:${value}`);

    // Traverse the child nodes
    for (let childNode of node.childNodes) {
      traverse(childNode);
    }
  }

  // Start the DOM tree traversal from the root element
  traverse(document.documentElement);

  // Finalize the hash and return the hash value as a string
  return hash.finalize().toString(CryptoJS.enc.Hex);
}
```

By warning users that changes have been made to the webpage, we can provide a better user experience and help them avoid confusion and frustration.

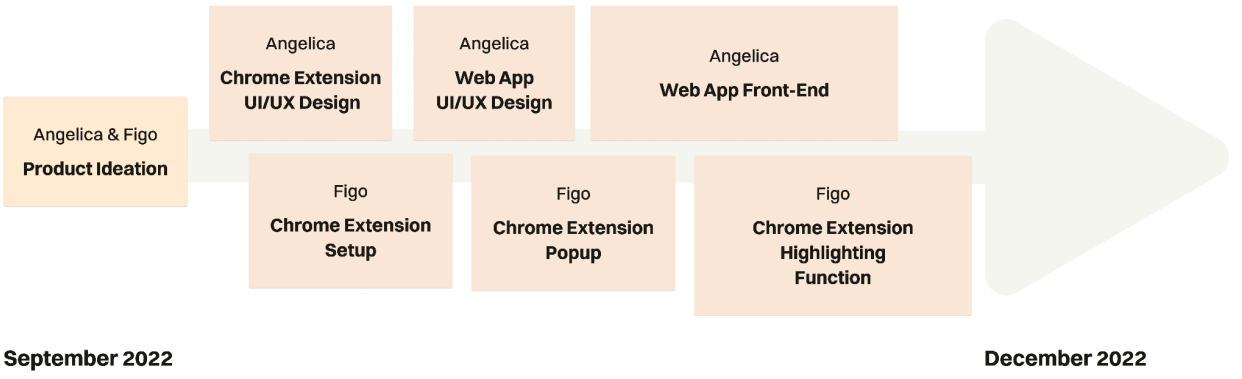
5. Division of Labor

5.1 Work Distribution

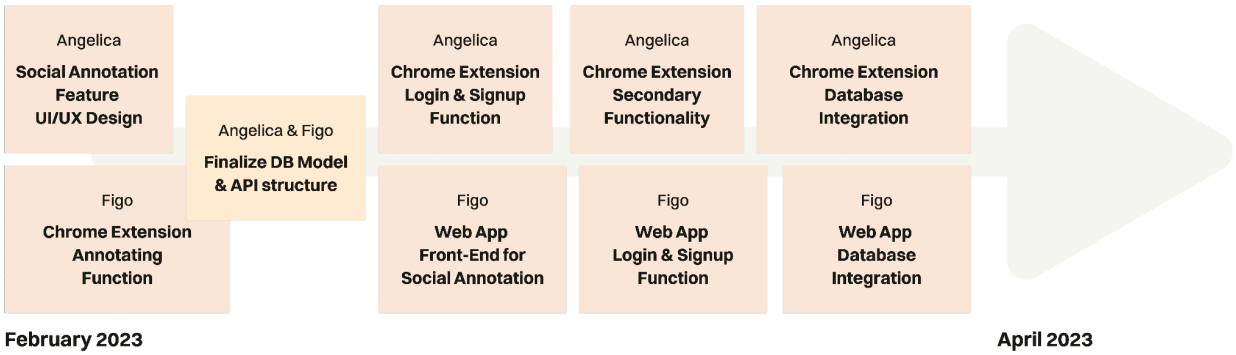
Throughout this project, we have maintained an equal division of labor and worked collaboratively to achieve our goals. We made a conscious effort to meet at least once a week to discuss our progress, address any challenges we faced, and set goals for the following week. Our meetings were mostly productive, and we made sure to discuss things that needed to be addressed together to ensure that we were both on the same page. As a result of this collaborative effort, we were both able to gain new technical skills and are proud of what we have achieved for this project.

Here is the timeline showing the breakdown of our work throughout the two terms:

Term 1:



Term 2:



5.2 Individual Contribution (Angelica)

During the first term, both Figo and I collaborated on the product ideation phase, where we brainstormed ideas and conceptualized the features and functionalities of Luminate. After that, I took charge of designing the User Interface (UI) and User Experience (UX) for the Chrome extension and web app. This involved creating mockups, wireframes, and prototypes to ensure a cohesive and user-friendly design for our application.

Meanwhile, Figo worked on the Chrome extension setup and the Chrome extension popup. We then divided the development work, with me focusing on the web app front-end, while Figo handled the Chrome extension highlighting function.

In the second term, I began by designing the UI/UX for the social annotation feature. After planning out what had to be done, Figo and I decided to switch roles. I took over the development of the Chrome extension, while Figo continued building the web app. This allowed me to gain hands-on experience in Chrome extension development, which further improved my understanding of the project as a whole.

I implemented the Chrome extension login and signup functions and the secondary functions related to the social annotation feature. I also worked on the Chrome extension database integration, which was another new experience for me, as I primarily worked with front-end development in the past. Figo, on the other hand, focused on the web app front for social annotation and the web app login and signup functions. He also took care of the web app database integration.

One of the main challenges I faced during the project was learning everything from scratch, as I had limited experience in Chrome extension development. This required me to quickly adapt to new technologies and tools while managing the design aspects of the project. Building the Chrome extension was a new experience, involving a lot of trial and error. However, overcoming these challenges has been a valuable learning experience, allowing me to expand my skill set and grow as a developer, and also improve my UI/UX skills.

In conclusion, working on Luminate has been a rewarding journey, filled with challenges and opportunities for growth. By collaborating closely with Figo and dividing tasks according to our strengths, we have managed to build something we're proud of.

6. Conclusion

For us, this project has been a steep learning experience. It all started with our desire to create something meaningful that people would use, and we eventually settled on the idea of a web annotation tool as a result of our frustration with the inefficiencies of taking notes from online resources. Throughout the project, we encountered a lot of things we have not learned before, such as knowledge management system, DOM manipulation, Chrome extension development, etc. We're committed to continue learning all the things we need to learn to build and eventually ship our app for people to use.

We are building Luminate with the mission to enhance all aspects of our information workflow. Our ultimate goal is to provide the ultimate information and knowledge management system that will transform how we research, consume, and organize information, starting with our web annotation tool.

7. References

[1] H. Nick, “How Many Websites Are There in the World?,” siteefy.com. https://siteefy.com/how-many-websites-are-there/

[2] “Welcome to Manifest V3,” developer.chrome.com. https://developer.chrome.com/docs/extensions/mv3/intro/

[3] “Overview of Manifest V3,” developer.chrome.com. https://developer.chrome.com/docs/extensions/mv3/intro/mv3-overview/

[4] “Is Google’s Manifest V3 the end of ad blockers?,” nordvpn.com. https://nordvpn.com/blog/manifest-v3-ad-blockers

[5] “Overview and timelines for migrating to Manifest V3,“ learn.microsoft.com. https://learn.microsoft.com/en-us/microsoft-edge/extensions-chromium/developer-guide/manifest-v3

[6] “React - A JavaScript library for building user interfaces,” reactjs.org. https://reactjs.org/

[7] “TypeScript: The starting point for learning TypeScript,” typescriptlang.org. https://www.typescriptlang.org/docs/

[8] “What is Next.js?,” nextjs.org. https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs

[9] “Tailwind CSS,” tailwindcss.com. https://tailwindcss.com/

[10] “webpack,” webpack.js.org. https://webpack.js.org/

[11] “Firebase Documentation,” firebase.google.com. https://firebase.google.com/docs

[12] “Firestore | Firebase,” firebase.google.com. https://firebase.google.com/docs/firestore

[13] “Firebase Authentication,” firebase.google.com.

https://firebase.google.com/docs/aut

[14] “Architecture overview - Chrome Developer,” developer.chrome.com. https://developer.chrome.com/docs/extensions/mv3/architecture-overview/

[15] P-L. Jerome, highlighter (Version 4.0.4) [Source code]. https://github.com/jeromepl/highlighter.

[16] “Selection and Range,” javascript.info. https://javascript.info/selection-range