

# Progress Report 1

## Web Summarizer and Shortener

COSC 4P02  
February 15th, 2024  
Naser Ezzati-Jivan

Tanvir Hasan - <i>Product Owner</i>	(6599328)	<a href="mailto:th18ai@brocku.ca">th18ai@brocku.ca</a>
Muhammed Bilal - <i>Scrum Master</i>	(6695738)	<a href="mailto:bb18hb@brocku.ca">bb18hb@brocku.ca</a>
Hamza Sidat - <i>Developer</i>	(6599591)	<a href="mailto:hs18so@brocku.ca">hs18so@brocku.ca</a>
Marium Nur - <i>Developer</i>	(7327182)	<a href="mailto:mn21xu@brocku.ca">mn21xu@brocku.ca</a>
Anjali Sabu - <i>Developer</i>	(7337033)	<a href="mailto:as21qj@brocku.ca">as21qj@brocku.ca</a>
Abdul-Maalik Siddiqui - <i>Developer</i>	(6785828)	<a href="mailto:as19fa@brocku.ca">as19fa@brocku.ca</a>
Amani Anderson - <i>Developer</i>	(6617344)	<a href="mailto:aa18ex@brocku.ca">aa18ex@brocku.ca</a>

# Table of Contents

<b>1. Introduction</b>	<b>2</b>
<b>2. Design</b>	<b>2</b>
2.1. Header/Footer	2
2.1.1. Header	2
2.1.2. Footer	2
2.2. Authentication Page	3
2.2.1 Login Page	3
2.2.2 Sign Up Page	3
2.2.3 Forgot Password / Verify User Page	4
2.2.4 Reset Password Page	4
2.3. Landing Page	5
2.3.1. Hero Section	5
2.3.2. Services Section	6
2.3.3. Reviews Section	6
2.4. Summarizer Page	6
2.5. URL Shortener Page	7
2.6. Feedback Page	8
<b>3. Implementation</b>	<b>8</b>
3.1. Text Extraction	8
3.1.1. Text extraction tests	8
3.2. Summarizer	10
3.2.1. Summarizer Tests	11
3.3. URL Shortener	11
3.3.1. URL Shortener Tests	12
3.4. Postgres Database	12
<b>4. Release Planning</b>	<b>13</b>
4.1. Sprint 1 (26 Jan - 9 Feb)	13
4.2. Sprint 2 (9 Feb - 23 Feb)	13
<b>5. Problems</b>	<b>13</b>
5.1. URL Shortener	14
5.2. Site Responsiveness	14
5.3. Github	14
5.4. API Tokens	15
5.5. Text Extraction	15
<b>6. Future Steps</b>	<b>15</b>
6.1. Frontend	15
6.2. Backend	16
<b>7. Meeting Notes</b>	<b>16</b>

# 1. Introduction

Over our first two sprints, we were able to flesh out an overview of our application, regarding which frameworks, databases, and APIs were going to be utilized throughout. In addition, we noted all the required tasks in Jira and assigned each of them a priority level. This report will summarize the development processes that resulted in our current design, implementation, and release planning, and some of the problems we faced. Firstly, the Design section will demonstrate the overall look of the website and each component by itself. Here, we will explain the logic behind each feature on the front end. Secondly, in the Implementation section, we will go over the backend of the website, where we discuss the main features and all of the frameworks and APIs that were utilized to perform the necessary functionalities of the website. Furthermore, the Release Planning section will state all of the steps and processes we took to develop our current work. This includes the two sprints and associated scrum meetings. Lastly, we will review any problems or challenges we faced during our development and planning processes and how we can fix them moving forward.

## 2. Design

### 2.1. Header/Footer

#### 2.1.1. Header



The header is sticky so that the user is always one click away from using the different components of the website. React icons have been used for the profile and dark mode icons. We are still looking into other website logos before finalizing a logo for the website.

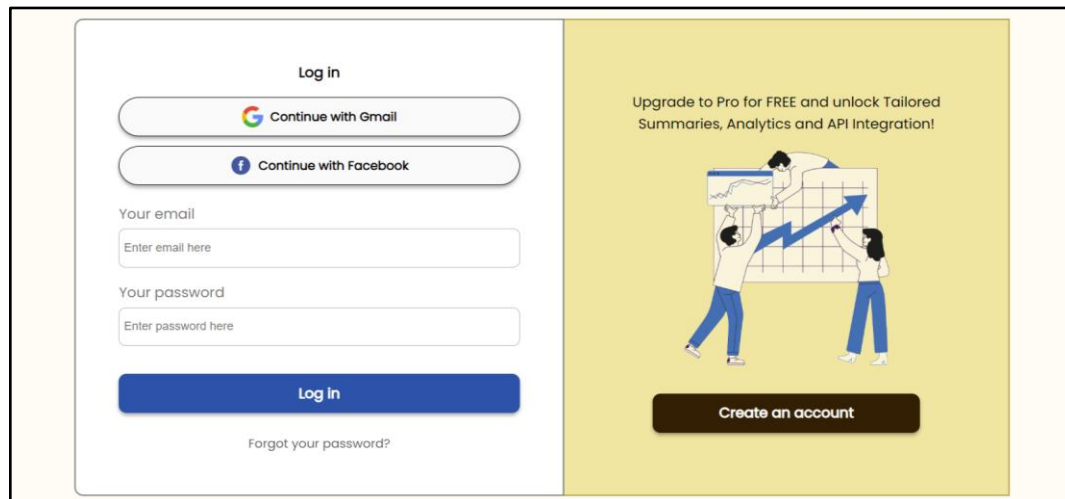
#### 2.1.2. Footer



### 2.2. Authentication Page

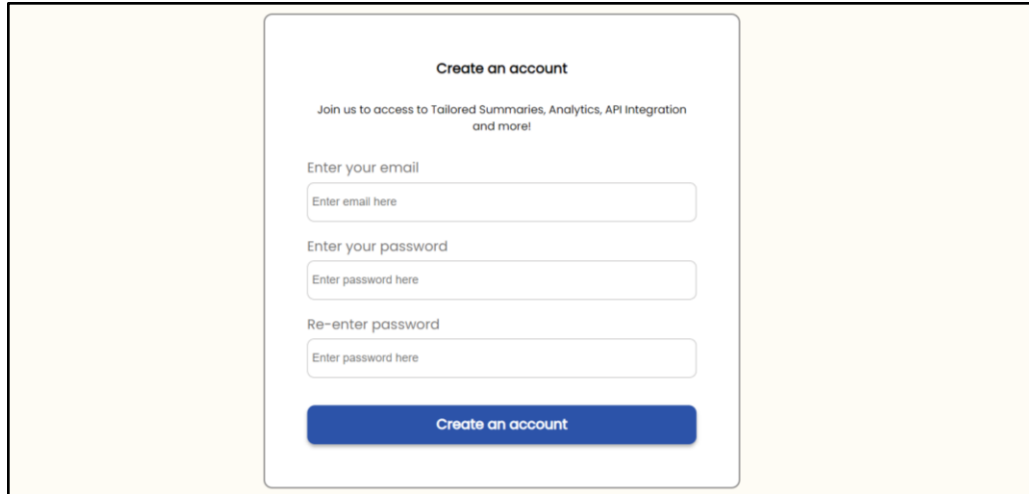
The Authentication page is where users can manage their accounts. It includes functionalities such as creating a new account, signing in to an existing account and resetting passwords if required. Overall, this page is inspired by many modern-looking authentication pages, incorporating stylish designs, shadows, and an overall neat look. Note that these user interfaces will be slightly expanded upon the integration of the actual authentication functionalities.

### 2.2.1 Login Page



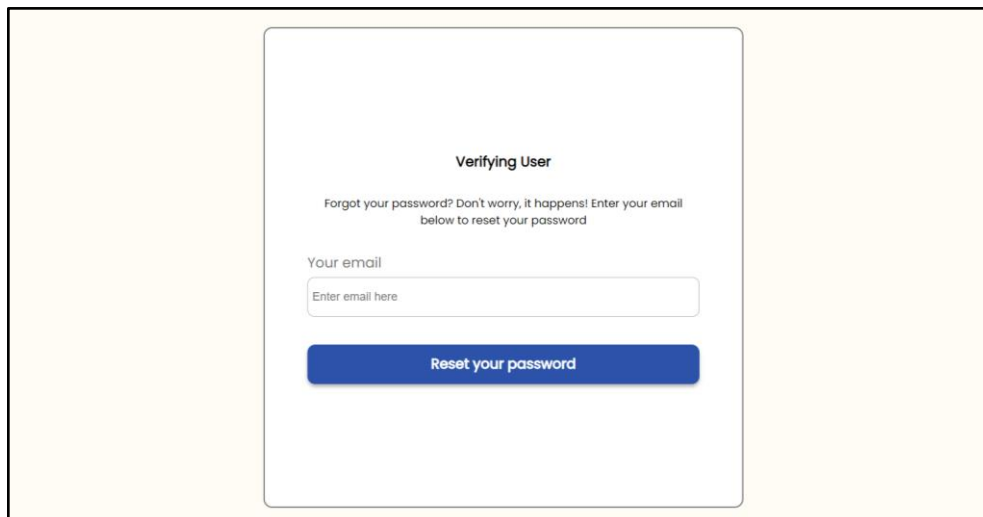
This page allows the user to log in to their account. It can be accessed by clicking the person icon in the Header. It consists of crucial text fields for entering their account email and password. There are also buttons above that allow the user to sign in to their account through social media, thus integrating Google and Facebook authentication. On the right side, a clearly visible and contrasting button guides the user to create an account if they haven't already done so. Lastly, a "Forgot your password?" link is provided for the user to reset their password.

### 2.2.2 Sign Up Page

A screenshot of a 'Create an account' form. The form is centered on a light yellow background. It has a title 'Create an account' at the top. Below the title is a subtitle: 'Join us to access to Tailored Summaries, Analytics, API Integration and more!'. There are three input fields: 'Enter your email' with a placeholder 'Enter email here', 'Enter your password' with a placeholder 'Enter password here', and 'Re-enter password' with a placeholder 'Enter password here'. At the bottom is a blue button labeled 'Create an account'.

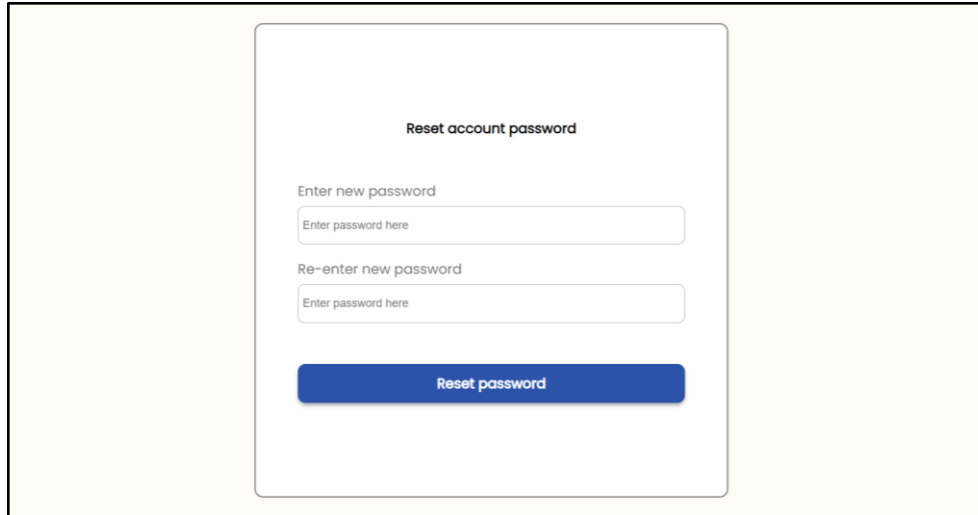
This page allows the user to create a new account. It has a user-friendly interface where users can input the necessary information to set up their accounts. The page includes essential text fields for entering key details such as the user's email address and chosen password. The “Re-enter password” field is essential for data accuracy and privacy during the account creation process.

### 2.2.3 Forgot Password / Verify User Page

A screenshot of a 'Verifying User' form. The form is centered on a light yellow background. It has a title 'Verifying User' at the top. Below the title is a subtitle: 'Forgot your password? Don't worry, it happens! Enter your email below to reset your password'. There is one input field labeled 'Your email' with a placeholder 'Enter email here'. At the bottom is a blue button labeled 'Reset your password'.

This page is displayed when the user clicks the “Forgot your password?” link and is designed to verify whether the user has an account created in the database already before redirecting to the page where they can reset their password. It features clear instructions for the user to enter their account email address for verification.

### 2.2.4 Reset Password Page

A UI mockup of a password reset form. The form is centered on a light yellow background. It has a title "Reset account password" at the top. Below the title are two input fields: "Enter new password" and "Re-enter new password", both with placeholder text "Enter password here". At the bottom of the form is a blue button labeled "Reset password".

Reset account password

Enter new password

Enter password here

Re-enter new password

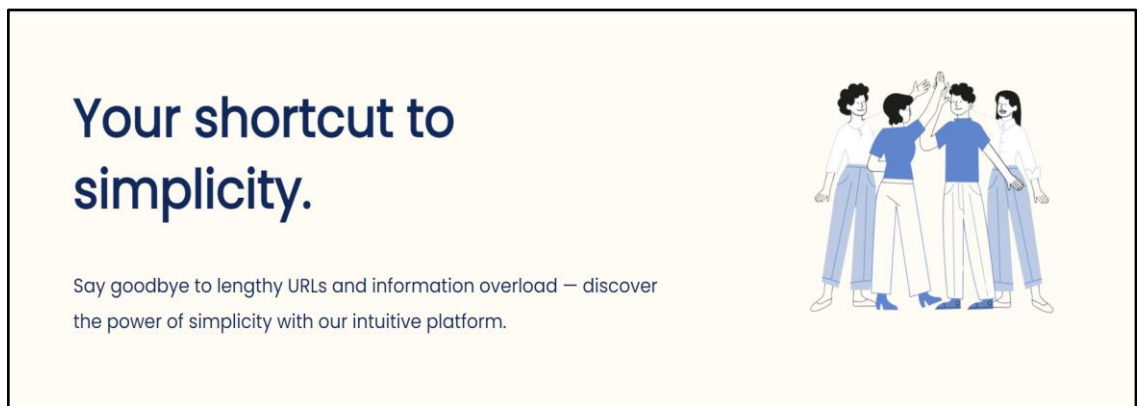
Enter password here

Reset password

This UI is designed to aid the user in resetting their password. It features instructions for the user to enter a new account password and includes a field for re-entering the new password to ensure data accuracy.

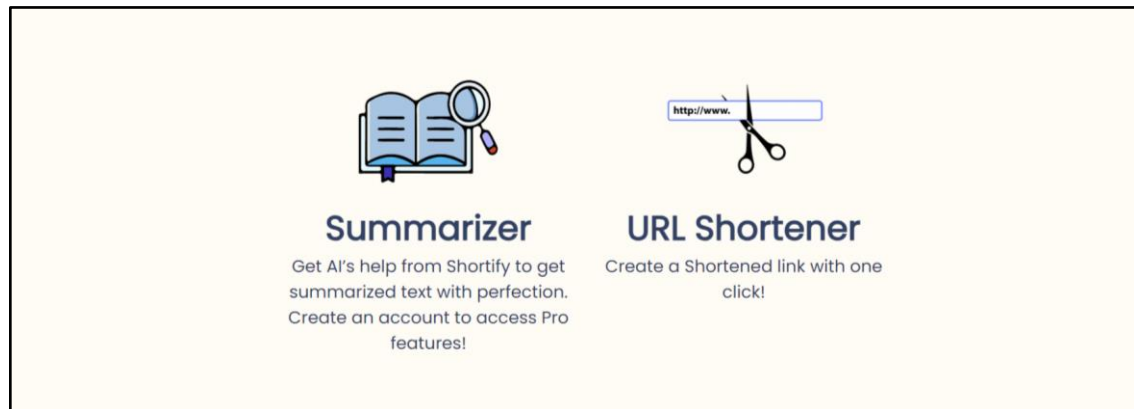
## 2.3. Landing Page

### 2.3.1. Hero Section



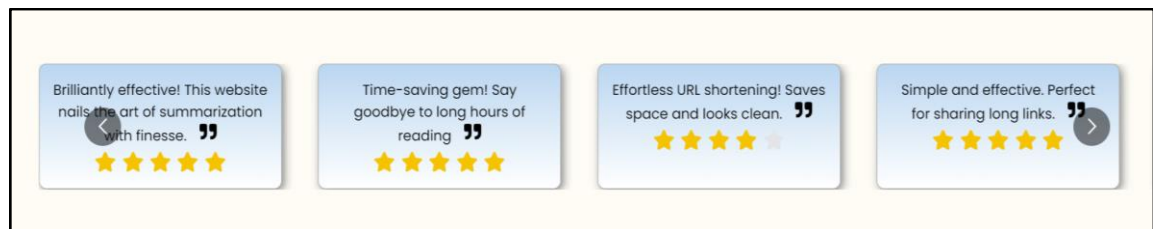
This hero section of our landing page is designed to captivate visitors and immediately convey the value and functionality of our product, through the clear and concise headline, "Your shortcut to simplicity", complemented by a brief description of the key functionalities of our tool, highlighting its ability to create concise summaries and shorten URLs for easy sharing.

### 2.3.2. Services Section



The services sections are clickable to allow users to access the different components. The short descriptions highlight the purpose of the tool to the user.

### 2.3.3. Reviews Section



The react-multi-carousel has been used to create the Reviews Slider. The reviews will always be set to the selected good feedback to be presented to current/future users. Additional properties have been added to the slider which include making it infinite (so that it does not reach an end on each side), and an autoplay feature such that the reviews move automatically every 3 seconds (the user can also manually click the arrows or click on a particular review to move the slider or keep it still so that autoplay pauses). The stars have been used from the react icons. A data variable has been added so that a new review can be added each time without having to change the reviews function.

## 2.4. Summarizer Page

The Summarizer page is a crucial component of the website, as this is where the summarization of the web media and text occurs. This component will consist of custom summarization levels of all types of media, such as text, website URLs, and YouTube video URLs, but for now, we are focusing on producing a basic working component that just accepts text. The selection UI for the premium will also be created in later sprints.

The screenshot shows a web application titled "Summarizer". It features two large text input areas side-by-side. The left area is labeled "Paste your text here..." and the right area is labeled "Get output here...". Below the left input area is a blue button labeled "Summarize". To the right of the "Summarize" button, there are two small icons: a thumbs-up and a speech bubble. The entire interface is set against a light yellow background.

The backend of the summarizer is explained below, however, there are still a couple of functionalities on the front end. The Summarizer uses the “textarea” element and is responsive for different vertical sizes (although the width responsiveness still needs to be adjusted in future sprints). The scroll bar has been made invisible and the “Summarize” button at the bottom remains there across multiple screen sizes. The icons and buttons have been styled similarly to promote consistency and organization throughout the website. Moreover, the output text area has been made read-only, until it has been assigned any text, enabling the user to copy any text that is generated in the output using the copy button. As the user types in the input text area, the delete icon lights up and once clicked, pops a dialog menu that prompts the user to confirm if they wish to delete the content in both the input and output areas.

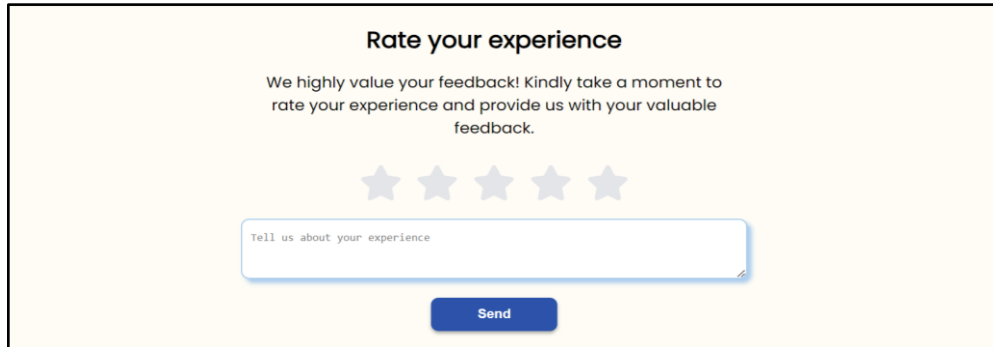
## 2.5. URL Shortener Page

The URL shortener page, currently simulating only the basic level feature of URL shortening, allows a user to input a link to be shortened, and receive a shortened link that redirects them to their original link once they paste it into a browser. The Shorten URL button generates a shortened link which is placed in the Shortened URL text field. The Copy URL button gives the user the option to copy the button directly to their clipboard instead of using CTRL+C.

The screenshot shows a web application for URL shortening. At the top, it says "Insert URL to be shortened". Below this is a text input field labeled "Enter URL here" and a blue button labeled "Shorten URL". Below the "Shorten URL" button, it says "Shortened URL". Underneath is another text input field labeled "Get Shortened URL" and a blue button labeled "Copy URL". The interface is set against a light yellow background.



## 2.6. Feedback Page



The feedback form is titled "Rate your experience" and includes a message: "We highly value your feedback! Kindly take a moment to rate your experience and provide us with your valuable feedback." Below the message is a five-star rating system. Underneath the stars is a text input field with the placeholder text "Tell us about your experience". At the bottom of the form is a blue "Send" button.

## 3. Implementation

### 3.1. Text Extraction

In the task of extracting text for summarization, we've employed a Flask-based backend with CORS enabled for web text extraction, utilizing Selenium and the Chrome WebDriver for dynamic page rendering, complemented by BeautifulSoup for HTML parsing to accurately extract text from web pages. This method ensures the comprehensive capture of readable text, even from JavaScript-heavy sites, by dynamically loading the content and parsing the HTML. For YouTube video transcripts, we've adopted the youtube-transcript-api, which allows for straightforward extraction of video IDs via regular expressions and efficient retrieval of transcripts, ensuring a seamless process for obtaining video text. For our project needs we did some modifications of this API and made the application take a YouTube URL as an input.

#### 3.1.1. Text extraction tests

For the web text extraction class, we implemented seven (7) unit tests which included:

1. Given the valid URL, the code successfully extracts the text
2. Given the valid URL, the code extracts the text with the error
3. Given the invalid URL, the code handles the error efficiently by throwing the appropriate error code
4. When no URL is given and the program is being called, the code throws the appropriate error code showing that no URL was provided

- Below is the snapshot of all these seven tests passed:

For the YouTube text extraction class, we implemented six (6) unit tests which included:

- Below is the snapshot for all six test case passes:

### 3.2. Summarizer

The summarization being the core feature of the web application, we chose to use OpenAI's gpt-3.5-turbo-0125. The summarization portion is able to take any text input, and summarize it in a variety of forms, be it simply reducing the amount of text material to a small paragraph, summarizing in point form, or any of the other many ways the user would like it to be summarized through the power of AI. While the AI can summarize any text-based content, the web app itself is able to summarize more than just pasted-in text. We have implemented the ability to summarize any webpage, and YouTube videos. We accomplished this with the previously mentioned text extraction for web pages, by passing in the scraped text from a webpage and handing it to the AI for summarization. The summarizer is able to summarize YouTube videos through the use of youtube-transcript-api 0.6.2, this API is able to extract the captions or auto captions of a YouTube video, and we then pass that to the AI for summarization. One challenge faced by using OpenAI's API is the limit of tokens. Naturally, many websites have a large amount of content that would be ideal candidates for summarization, but the token limit does not allow for such huge prompts. We noticed that many of these web pages are filled with content that may not be relevant to what the user wants to summarize anyway, it often includes large URLs to other pages rather than the main content of the page, so we will implement a way to focus only on what is most relevant to summarize. Overall, we have conducted many tests and have found the quality of the summaries to be of high quality, and we are satisfied with the summarization tool. Below is the testing page for our summarization, it takes in every input mentioned, plus some others we're testing.

## Web Summarizer

Enter Text to Summarize:

GPT-4 is a large multimodal model (accepting text or image inputs and outputting text) that can solve difficult problems with greater accuracy than any of our previous models, thanks to its broader general knowledge and advanced reasoning capabilities. GPT-4 is available in the OpenAI API to paying customers. Like gpt-3.5-turbo, GPT-4 is optimized for chat but works well for traditional completions tasks using the Chat Completions API. Learn how to use GPT-4 in our text generation guide.

Enter a URL to summarize:

Enter a YouTube URL to summarize:

Or upload a video:

No file selected.

### Summary:

GPT-4 is a high-performing multimodal model capable of accepting both text and image inputs, and producing text outputs. It surpasses earlier models in accuracy due to its extensive general knowledge and enhanced reasoning abilities. It is accessible to paying customers through the OpenAI API, similar to gpt-3.5-turbo. GPT-4 is primarily optimized for chat applications but can also handle traditional completion tasks using the Chat Completions API. Users can find guidance on how to utilize GPT-4 in the text generation guide.

### 3.2.1. Summarizer Tests

While the summarization itself was not of concern for tests since that all is handled by an API, we have ensured tests work with our implementation. Test cases encompass a variety of scenarios, including assessing the summarizer's ability to handle large text bodies, its effectiveness in distilling key points from diverse topics, and its performance in maintaining the coherence and relevance of summaries across different languages and domains. Additionally, edge cases, such as extremely lengthy texts or those with minimal content, have been thoroughly tested to ensure the summarizer is working as intended. The test cases include tests for web pages, YouTube videos, and pasted content.

## 3.3. URL Shortener

The URL shortener's implementation evolved throughout the sprints and throughout development. The current approach, which will persist for the project's lifespan, is based on a Python implementation, with no use of external APIs for the shortening task, but makes use of the Flask framework to handle the redirection of the links. The shortener works by accepting a link and then generating a random unique hash based on the given link. This hash is then appended to the Flask server's domain (currently localhost:5000), generating a full link with a 6-character hash as the shortened link. The ability to create a custom shortened link has been implemented but will evolve as users are implemented. This short link and the original link are stored in a Postgres database table upon shortening. The shortener's implementation also allows for the URL to be resolved, which works by searching the database for the shortened URL when given as an argument and returning the original URL associated with it in the database. It also increases the click count associated with the short link whenever the URL is resolved. When the short link is clicked (or pasted into the browser), if the Flask server is running, it will route the link to the redirection method which resolves the shortened URL and redirects the user to the original URL. This Python implementation communicates with the front end's JavaScript implementation through POST. When a user clicks the 'Shorten' button on the website that is rendered using the React framework, a fetch call is made using the POST method, which points towards the Flask server hosting the shortener's functionality. Flask uses the route given by the JavaScript POST call to call the shortening method, shortening the link, storing it in the database and returning a shortened link to the JavaScript file which is then displayed to the user.

### 3.3.1. URL Shortener Tests

1. Test that the table exists in the database.
2. Test that a link is successfully shortened.

3. Test that a link is resolved (returns the original)
4. Test that the click count of a link is initially set to 0.
5. Test that the click count is updated when it is clicked.
6. Test that URLs are unique given that the same original URL is inputted.
7. Test that a custom URL is made.

```
collected 7 items
testShortener.py ..... [100%]
===== 7 passed in 3.44s =====
```

### 3.4. Postgres Database

A Postgres database has been created for two main purposes as of now. Firstly, to handle the URL shortening capability, and also for us to store user information when users are implemented. Naturally, there exist two tables in this database, users, and shortened\_urls. These tables will likely be evolved in the coming sprints, and are unlikely to be the only two tables that we use on this database. The only table being used is the shortened\_urls database, which contains the following columns: id, original\_url, short\_url, and click\_count. As soon as we tackle the ability for users to sign in, we will include a column for users to be associated with shortened links if they are signed in so that the pro features can be fulfilled.

## 4. Release Planning

### 4.1. Sprint 1 (26 Jan - 9 Feb)

During the first sprint, we made some initial progress on both the front end and back end. On the front side, we got started by drafting a basic UI design in Figma and then diving into React to bring key components like the landing page, summarizer, hero section, and login module to life. However, there's still some work that needs to be done, especially to ensure everything is responsive for different screen sizes, particularly for the larger screens. On the backend, after trying out a few APIs without much luck, we turned to OpenAI to help us summarize text extracted from website URLs. Alongside that, we managed to set up a basic URL shortener with some statistics features and began laying the groundwork for integrating PostgreSQL to handle our database needs. The feedback we received in our scrum meetings resulted in the addition of new features like a feedback button for the summarized outputs and separate user histories for both the URL shortener and summarizer functionalities.

## **4.2. Sprint 2 (9 Feb - 23 Feb)**

During the second sprint, frontend development continued with React, focusing on completing components such as the URL Shortener, Feedback, Forget Password, Create Account, and Reviews components. Additionally, separate pages for the website were created and the Link component was used. Efforts were made to ensure button consistency across all pages. However, it was discovered late in the sprint that many components lacked responsiveness, especially for larger screens, which will be addressed in the next sprint. On the backend, progress was made using PostgreSQL to store inputs, outputs, and statistics for the URL shortener. Challenges arose with token usage for summarization and text extraction visibility issues. Integration of the URL shortener branch with the UI was done, and testing with frontend design began. Issues with URL redirection also needed to be addressed. Work commenced on using YouTube URLs for summarization input, and features like a custom shortened link option for pro users and test cases were added to the URL shortener functionality.

## **5. Problems**

### **5.1. URL Shortener**

The URL shortener's implementation evolved throughout the development process. Initially, the Bitly API for URL shortening was used as it seemed to be free, and it provided several usage tracking and analytical information. This was implemented in Python and JavaScript. However, it became clear upon further inspection, that after a few free uses, this API would become expensive. The idea was scrapped, and a no-API implementation was explored. Python was chosen as the language as we were investigating the use of Flask to handle our entire website. As a result, a fully working implementation of the URL shortener was created using Flask-rendered pages. However, as the team decided to use React for the front end due to its superior UI and customizability, this implementation was changed to allow for the shortening capability to be functional without Flask-rendered pages, but solely routing. This way, the Flask server does not interfere and can interface with the React server. In testing the integration between the React-generated pages and the Flask server redirection, the shortening was successful, but the redirection kept returning a URL not found error from the database. This was fixed by adding the Flask server's domain before the hash, then resolving the full URL as the database stores the domain and the hash as the shortened link.

## **5.2. Site Responsiveness**

Several challenges arose during the design components of the website. This mainly included the responsiveness of many components, as they failed to adjust to different screen sizes. This resulted in the creation of a few extra tasks mid-sprint, addressing size and responsiveness concerns, creating a setback in our sprint timeline. Additional challenges regarding the UI included the numerous re-evaluations of the website header and overall colour scheme. Lastly, a lot of problems arose around the repeated use of the same CSS classes, where changing one component would affect many others. This caused a lot of confusion and duplication while minimizing code reusability.

## **5.3. Github**

Learning the ropes of using source control like Github was especially challenging for the majority of the team. The team has had to learn all the commands for pushing, pulling, and merging as well as understanding how branches work in the development space. As a result, some issues included node modules being pushed onto the main repository multiple times, making it difficult for other members to clone the repository.

## **5.4. API Tokens**

As briefly mentioned in the summarizer section 3.2, API tokens can be a limiting factor when scraping some websites. Initially, when we first started development, we were using OpenAI's gpt-3.5-turbo, this had a context window of 4,096 tokens, as of recently, gpt-3.5-turbo is being retired and currently points to the gpt-3.5-turbo-0613 model. OpenAI has upgraded its users with gpt-3.5-turbo-0125, and it has a context window of 16,385 tokens. This new token limit increase has made it so very few web pages ever hit the token limit, solving our problem. If the 16,385 token limit is ever an issue, we have other options such as gpt-4-0125-preview, which has a context window of 128,000 tokens, much more than what we will ever need, with the only drawback being higher pricing. As for the youtube-transcript-api, there is no token limit.

## **5.5. Text Extraction**

In the process of developing a backend for text extraction, we encountered significant challenges, for both web pages and YouTube videos. Initially, the search for an API capable of extracting comprehensive and relevant text from websites proved fruitful, as we were unable to find a solution that met our criteria for completeness and accuracy. This necessitated a pivot to a more manual approach,

utilizing BeautifulSoup along with Selenium for dynamic content rendering, which, while effective, introduced complexity in handling JavaScript-heavy sites and ensuring the reliability of the extracted text. Similarly, for YouTube text extraction, we faced difficulties in finding an API that could seamlessly integrate with our framework and reliably provide transcripts. Despite testing multiple APIs, we faced integration challenges and inconsistencies in transcript availability, prompting us to settle on the youtube-transcript-api for its relative ease of use and integration capability.

## **6. Future Steps**

### **6.1. Frontend**

- Plan to make all components fully responsive (can use the clamp() function in CSS for font size, height, etc)
- Start working on the pro user pages for Summarizer (including custom pro features) and URL Shortener (including the custom link)
- Start working on the user dashboard for the pro user account

### **6.2. Backend**

- Plan to allow for custom URL shortening
- The inclusion of users; creating, and deleting accounts
- Work on custom features for Summarizer

## **7. Meeting Notes**

[All Meeting Notes](#)