Final Report

Web Summarizer and Shortener

COSC 4P02
April 24th, 2024
Naser Ezzati-Jivan

Tanvir Hasan - *Product Owner*	(6599328)	th18ai@brocku.ca
Muhammed Bilal - *Scrum Master*	(6695738)	bb18hb@brocku.ca
Hamza Sidat - *Developer*	(6599591)	hs18so@brocku.ca
Marium Nur - *Developer*	(7327182)	mn21xu@brocku.ca
Anjali Sabu - *Developer*	(7337033)	as21qj@brocku.ca
Abdul-Maalik Siddiqui - *Developer*	(6785828)	as19fa@brocku.ca
Amani Anderson - *Developer*	(6617344)	aa18ex@brocku.ca

Table of Contents

1. Introduction

Within this report, we present a comprehensive overview of the Shorify web application. Utilizing the Agile development process, we crafted the website, breaking down our work into sprints lasting two weeks each. Here, we detail the tasks accomplished throughout each sprint, providing insights into our iterative development approach. Furthermore, we delve into the finalized iteration of our web application, accompanied by essential resources such as a user manual, technical manual, and installation guide. Additionally, comprehensive test documentation for both the frontend and backend components has been included, ensuring the robustness and reliability of our application.

2. Updated Requirements/Features

1. Summarization
    a. Summarization given text
    b. Summarization given a URL
    c. Summarization given a URL to a YouTube video
    d. Ability to copy or export the summarized content
2. (Pro) Custom Summarization
    a. Ability to decide the length level: long, medium or short.
    b. Ability to format the output: paragraph, bullet points, numbered list, etc.
    c. Ability to choose the tone of the summary: standard, formal, sarcastic, etc.
    d. Ability to add references: APA, Chicago, etc.
    e. Ability to choose to summarize the full youtube video or part of a video by inserting timestamps.
    f. Ability to load a saved template into the summarizer when summarizing text or URL.
3. URL Shortener
    a. Standard Link Shortening
    b. (Pro) Ability to add a custom word to the shortened URL
    c. Ability to copy the shortened URL
4. Account creation (to become a pro user)
    a. Authentication with an external service (Google, Facebook)
    b. Manual account creation (email address, username)
5. User Dashboard
    a. History
        i. Ability to view the past five saved summaries and copy the output to the clipboard.
        ii. Ability to view the past five shortened URLs and view the click count for each shortened URL along with copying the shortened URL to the clipboard.
        iii. Ability to delete one row or all of the history.
    b. Templates

        i. Ability to view and edit three custom summarization levels for the shorify summarizer.
    c. API Access
        i. Ability to provide all website services via an API for the summarizer and url shortener.
        ii. Provides users with an API key
    d. Settings
        i. View or update the name associated with the account
        ii. View or update the email associated with the account
        iii. Reset the password associated with the account
        iv. Ability to delete account
    e. Logout of the account
6. Miscellaneous Features
    a. Dark Mode or light mode

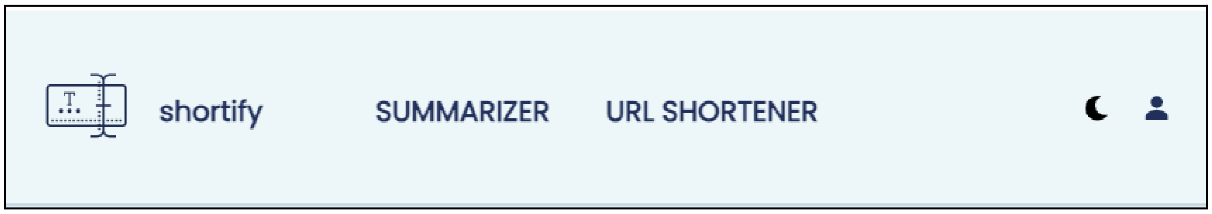
3. User Manuals

3.1. Accessing the web application

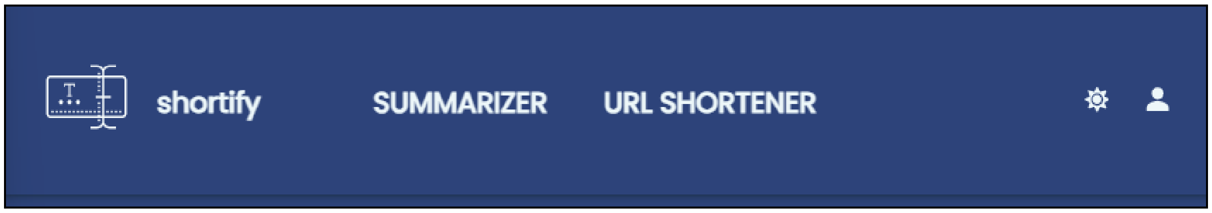
The web application is available at https://4p02shortify.com/ and can also be run locally with the instructions found in the installation manual.

3.2. Header

The header component can be found at the top of every page. It provides users easier access to our main features, the summarizer and URL shortener. It also allows them to click on shorify or the icon to come back to the landing page. Users can choose to create an account, log in or access the user dashboard using the profile icon on the rightmost of the header.



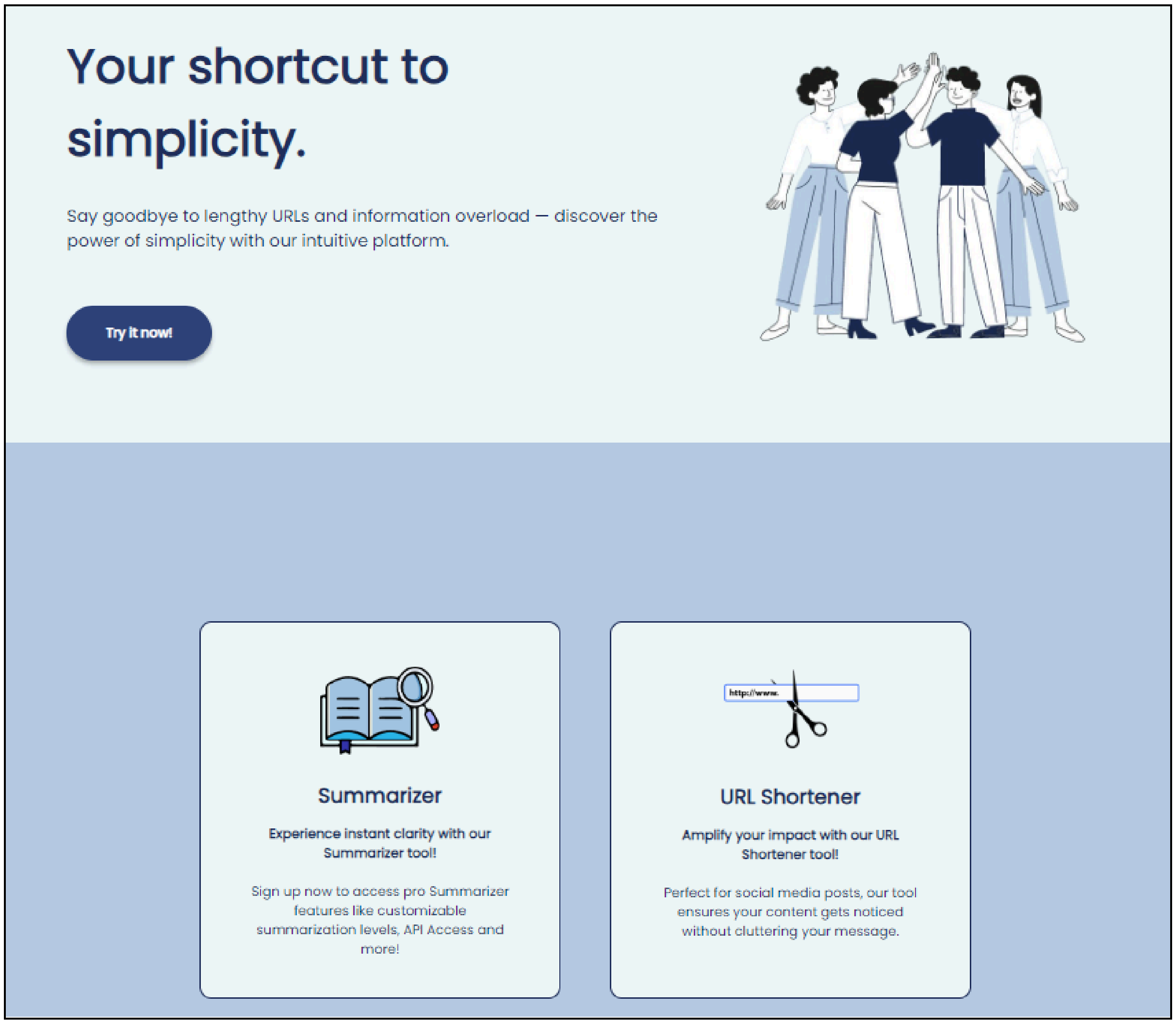
Users can change the theme of the website to dark mode by clicking on the moon icon.



3.3. Landing Page

The simple and easy to user landing page has been created that gives users a simple overview of the summarizer and the URL Shortener along with a reviews and hero component.

Users can also click on the “Try it now!” button to access the summarizer or click on the summarizer or URL shortener descriptions to access the different pages.



Your shortcut to simplicity.

Say goodbye to lengthy URLs and information overload — discover the power of simplicity with our intuitive platform.

Summarizer

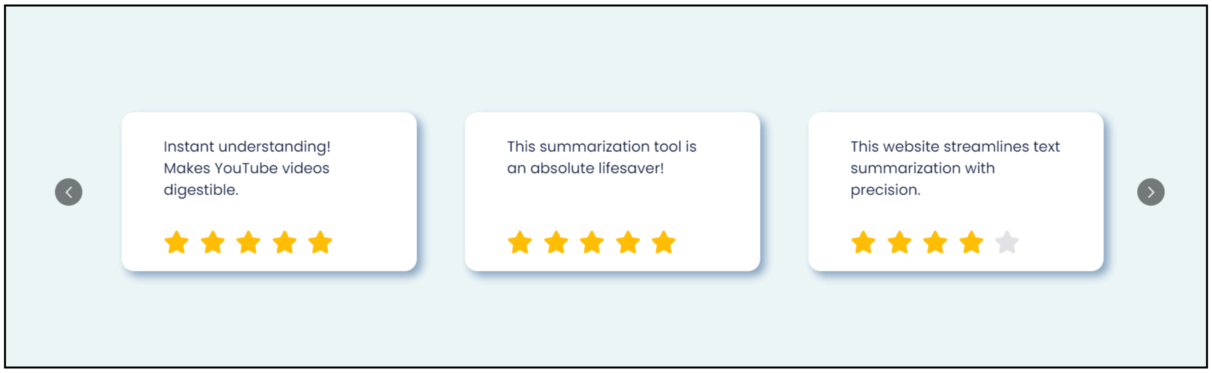
Experience instant clarity with our Summarizer tool!

Sign up now to access pro Summarizer features like customizable summarization levels, API Access and more!

URL Shortener

Amplify your impact with our URL Shortener tool!

Perfect for social media posts, our tool ensures your content gets noticed without cluttering your message.


Instant understanding! Makes YouTube videos digestible.

This summarization tool is an absolute lifesaver!

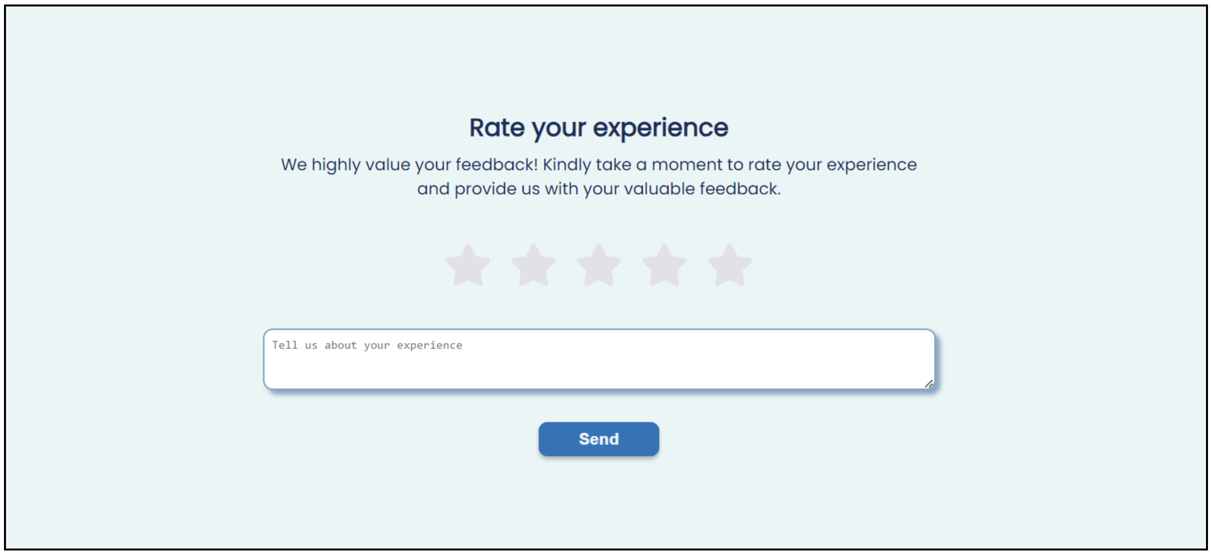
This website streamlines text summarization with precision.

3.4. Footer

A footer component can be found at the bottom of every page, which allows users to access the feedback page of the website.

3.5. Feedback Page

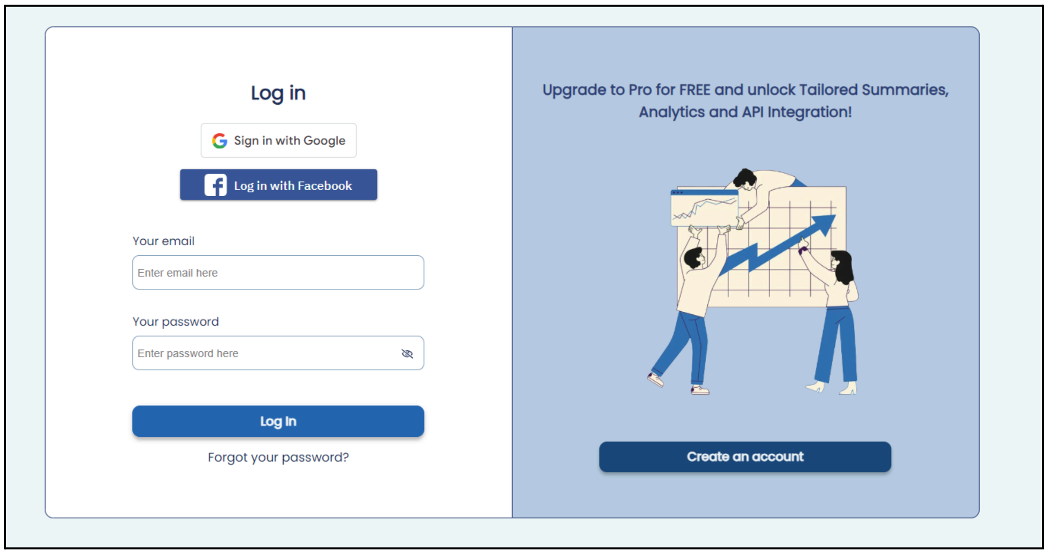
The feedback page allows users to leave an anonymous review of the services offered by the website. They can leave a star rating and/or a text-based review, which is then stored in the database.


Rate your experience

We highly value your feedback! Kindly take a moment to rate your experience and provide us with your valuable feedback.

3.6. Authentication Pages

Free users can click on the profile icon in the header to access the authentication page where they can choose to log into an existing account or create an account to access premium features.


Log in

Sign in with Google

Log in with Facebook

Your email

Enter email here

Your password

Enter password here

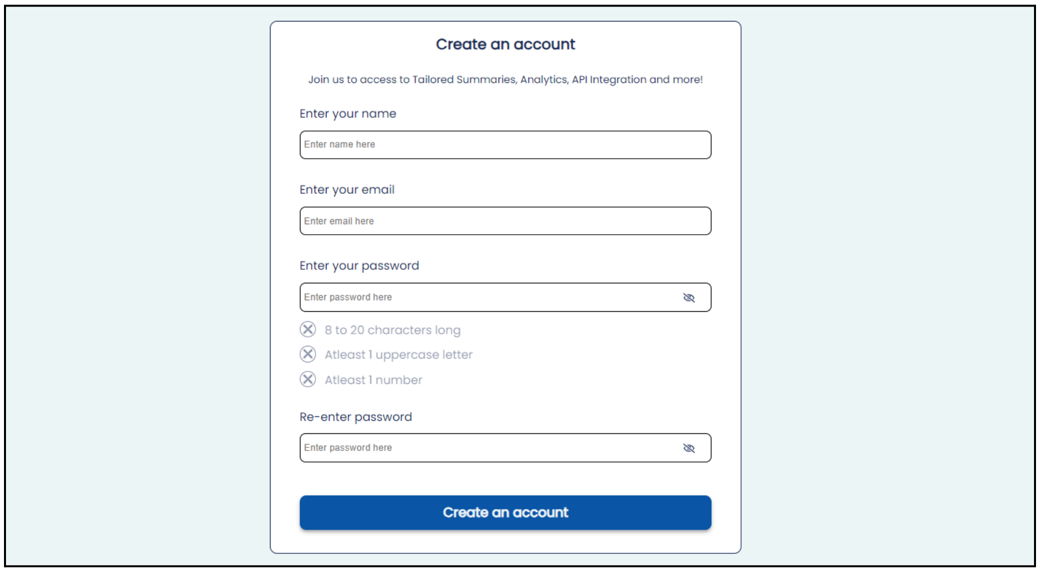
Log In

Forgot your password?

Upgrade to Pro for FREE and unlock Tailored Summaries, Analytics and API Integration!

Create an account

Clicking on the 'Create an account' button leads users to a different page, where they can manually create an account using an email, name and a password.


Create an account

Join us to access to Tailored Summaries, Analytics, API Integration and more!

Enter your name

Enter name here

Enter your email

Enter email here

Enter your password

Enter password here

8 to 20 characters long

Atleast 1 uppercase letter

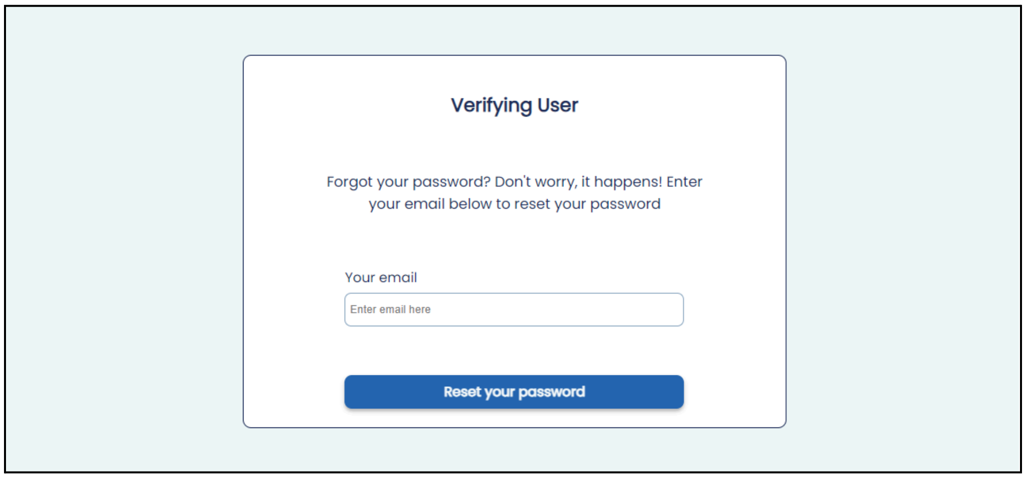
Atleast 1 number

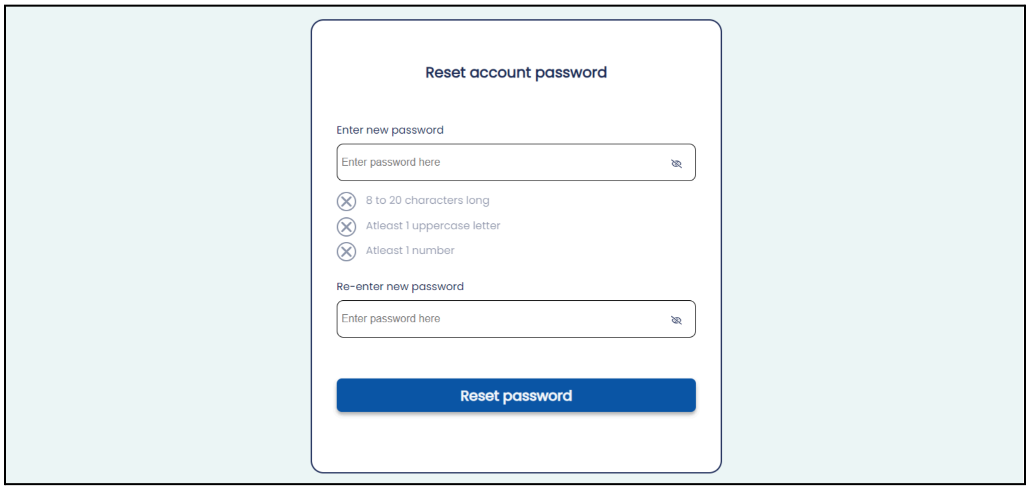
Re-enter password

Enter password here

Create an account

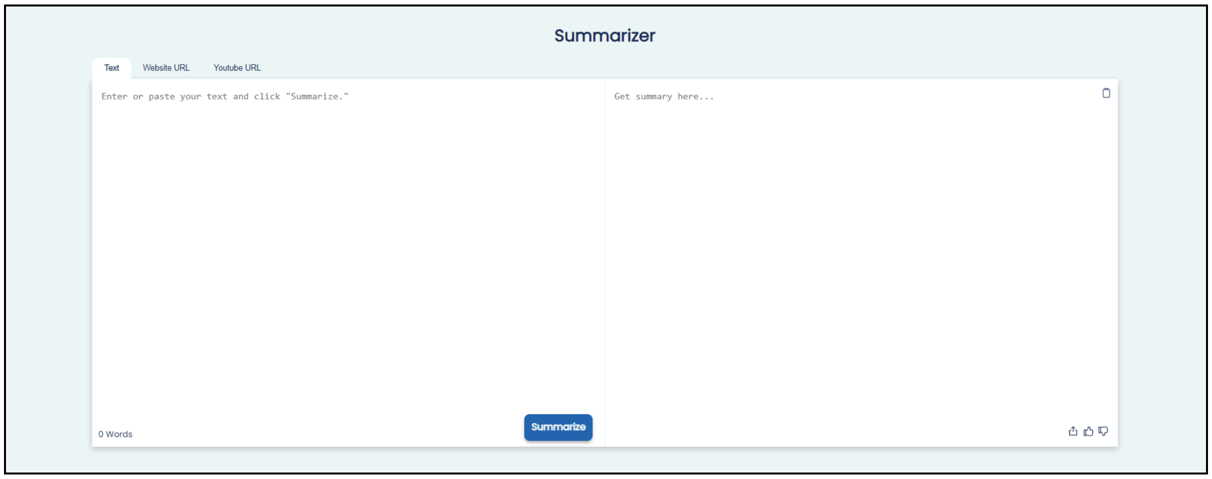
If an existing user has forgotten their password, they can click the forgot password in the login page, 'Forgot your password?' has been placed right below the 'Login' button. Clicking on 'Forgot your password?', leads to a new page to verify the user's email address. Once verified, they can click on the ‘Reset your password’ button leading to a new page allowing them to set a new password for the account.


Verifying User
Forgot your password? Don't worry, it happens! Enter your email below to reset your password
Your email
Enter email here
Reset your password



3.7. Free Users

3.7.1. Summarizer



Free users have access to all the necessary features for summarization. Nothing needs to be done to become a free user, simply visiting the site.

Content Types You Can Summarize:

- **Plain Text**: Summarize input text up to 125 words.
- **Web Pages**: Provide a URL to summarize the content of a web page.
- **YouTube Videos**: Use a YouTube URL to summarize video content.

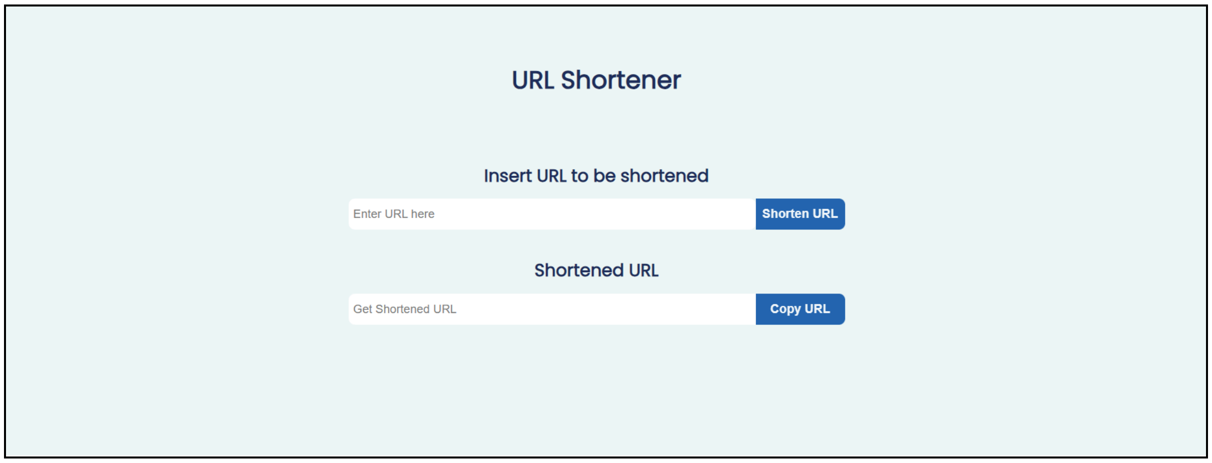
How to Use Shortify Summarizer:

1. **Select the Content Type**: Click on the appropriate tab to select the type of content you wish to summarize (Text, Web Page, YouTube).
2. **Input Your Content**: Type or paste the content or URL into the input field.
3. **Summarize**: Click the 'Summarize' button to generate your summary.
4. **Your Summary**: The summary will appear in paragraph form with a standard tone to the right.
5. **Post-Summary Options**:
    - **Copy**: You can copy the summary to the clipboard.
    - **Export**: Export the summary as a JSON file for additional use. The JSON file will include your input text or URL and the summary.
    - **URL Shortening**: For URL summaries, opt to shorten the URL post-summary.

Summary Rating:

Use the thumbs up or thumbs down buttons to rate the summary. This helps us improve our service.

3.7.2. URL Shortener



Free users have access to shortening a URL. Nothing needs to be done to become a free user, simply visiting the site.

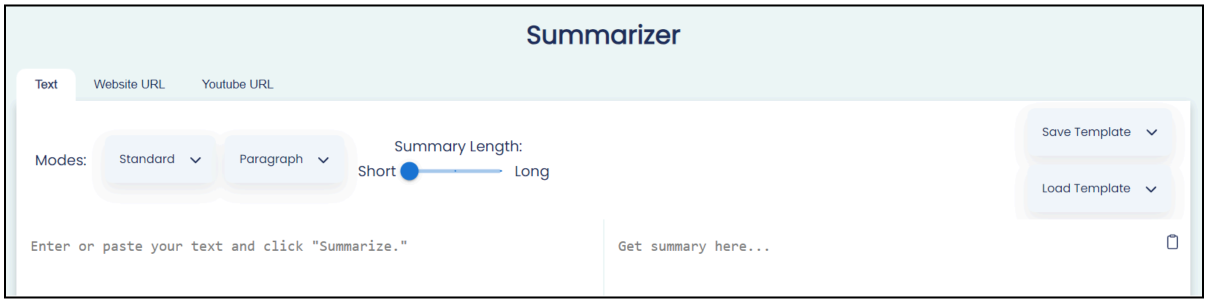
How to Use Shortify URL Shortener:

1. **Input a URL**: Click on the insert URL input field and type or paste a URL there.
2. **Shorten**: Click the ‘Shorten URL’ button to generate the shortened URL.
6. **Your Shortened URL**: The shortened URL will appear in text field box placed on the bottom.
7. **Post-Shorten Options**:
   - **Copy**: You can copy the shortened URL to the clipboard.
   - **Summarizer**: After successful URL shortening, a button will appear that will allow users to summarize the same URL using the shortify web summarizer.

3.8. Premium Users

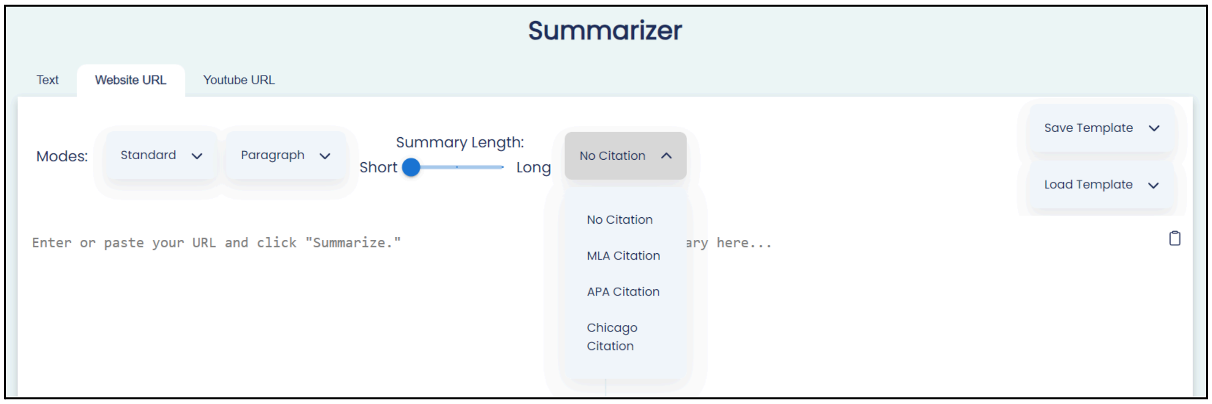
To become a Premium User, all that needs to be done is be logged in, once logged in, all features become available. Signing up or logging in can be done by clicking on the person icon at the top right.

3.8.1. Summarizer

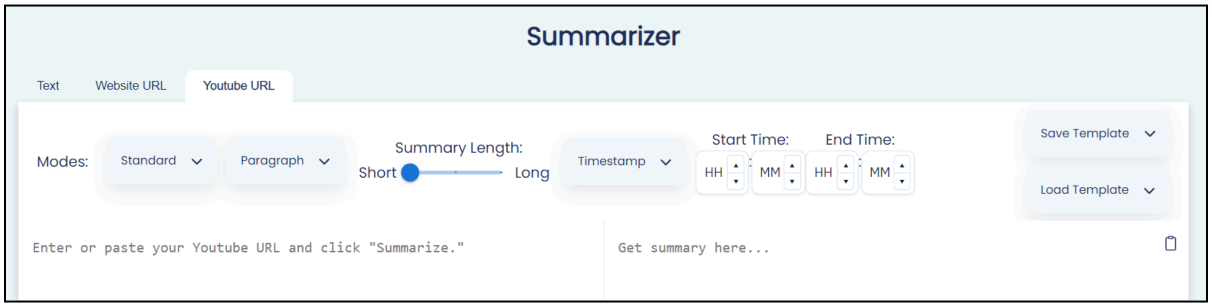


Enter or paste your text and click "Summarize."

Get summary here...



Enter or paste your URL and click "Summarize."



Enter or paste your Youtube URL and click "Summarize."

Get summary here...

Content Types You Can Summarize:

- **Plain Text**: Summarize any length of text.
- **Website URL**: Summarize entire web pages.
- **YouTube URL**: Summarize YouTube video content, including specific segments.

How to Use Shortify Summarizer:

1. **Select the Content Type**: Click the tab corresponding to the content type (Text, Web Page, YouTube).
2. **Input Your Content**: Type or paste the content or URL into the input field.

Customization Options:

- **General Options for All Types**:
  - **Tone**: Choose the summary's tone from the tone dropdown.

    - **Style**: Select the summary's style from the style dropdown.
    - **Length**: Adjust the summary length using the slider.
- **Website URL**: Additionally, select a citation style from the citation dropdown.
- **YouTube URL**:
    - **Summarization Type**: Choose between summarizing the full video or specific timestamps.
    - **Timestamp Inputs**: For timestamp summarization, input the start and end times (hours and minutes) using the provided fields.

Templates:

- Save your customizations as templates for quick reuse. Templates can be saved and loaded from the 'Save Template' and 'Load Template' dropdowns.

Generating Your Summary:

- Click the 'Summarize' button. Your summary will appear in paragraph form with the chosen tone to the right of the screen.

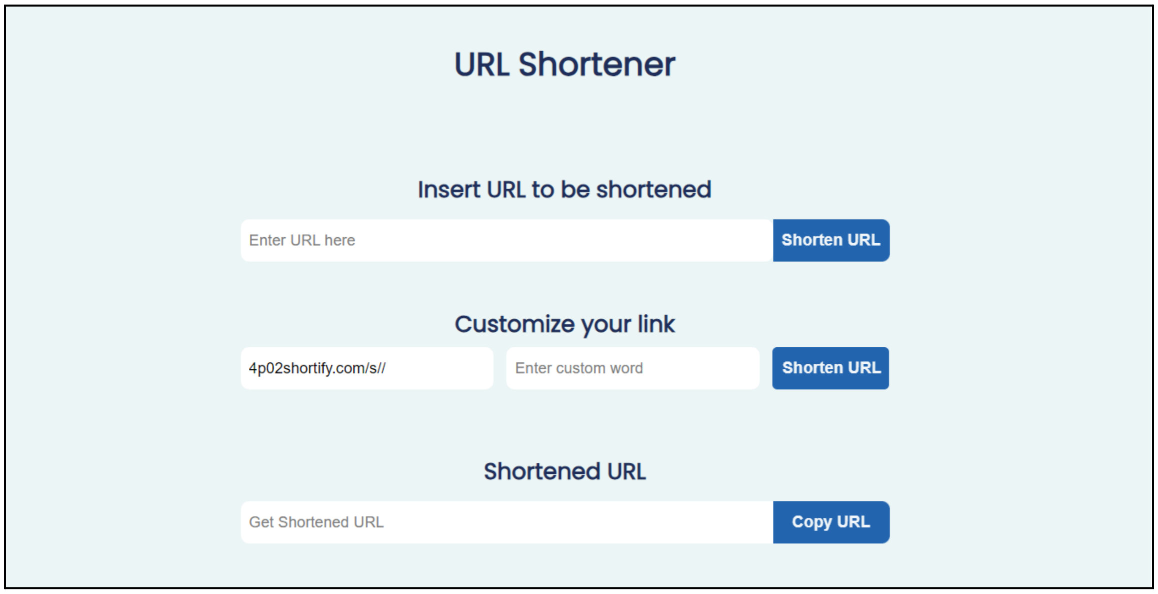
Post-Summary Options:

- **Copy**: Copy the summary to the clipboard.
- **Export**: Export the summary and the original content as a JSON file.
- **URL Shortening**: Opt to shorten a website URL after summarizing it.

Summary Rating:

Provide feedback on your summary with thumbs up or thumbs down.

3.8.2. URL Shortener



How to Use Shortify URL Shortener:

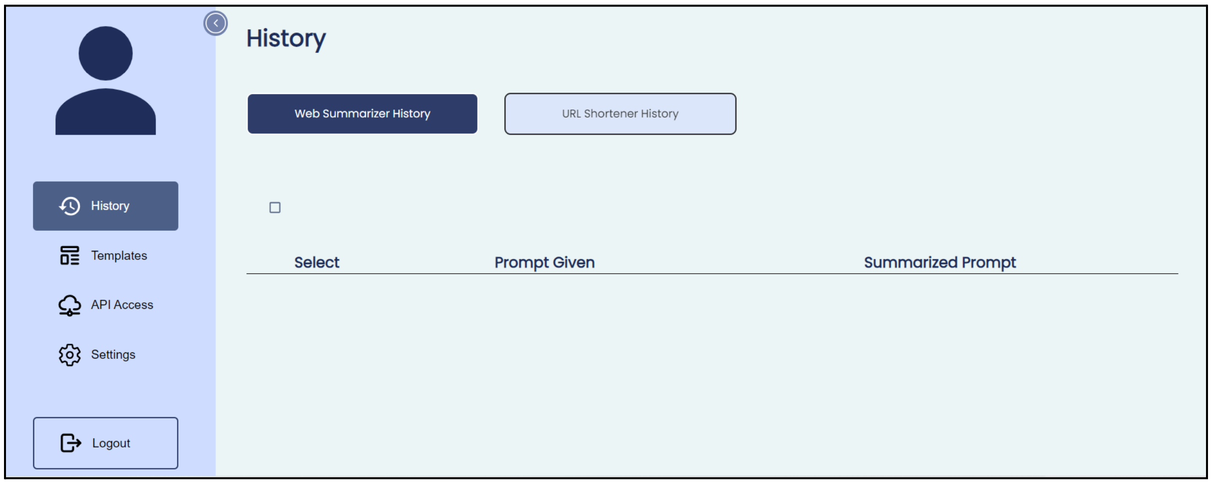
1. **Input a URL**: Click on the insert URL input field and type or paste a URL there.
2. **Customization option**: insert a custom word to be added in the shortened URL.
3. **Shorten**: Click the 'Shorten URL' button to generate the shortened URL.
4. **Your Shortened URL**: The shortened URL will appear in the text field box placed on the bottom. It would contain the username that was automatically assigned to the user when they created an account along with the custom word.
5. **Post-Shorten Options**:
    a. **Copy**: You can copy the shortened URL to the clipboard.
    b. **Summarizer**: After successful URL shortening, a button will appear that will allow users to summarize the same URL using the shortify web summarizer.

3.8.3. User Dashboard

Premium users will have access to the user dashboard. The user dashboard has a collapsible sidebar so that users can access different pages within the user dashboard,

3.8.3.1. History

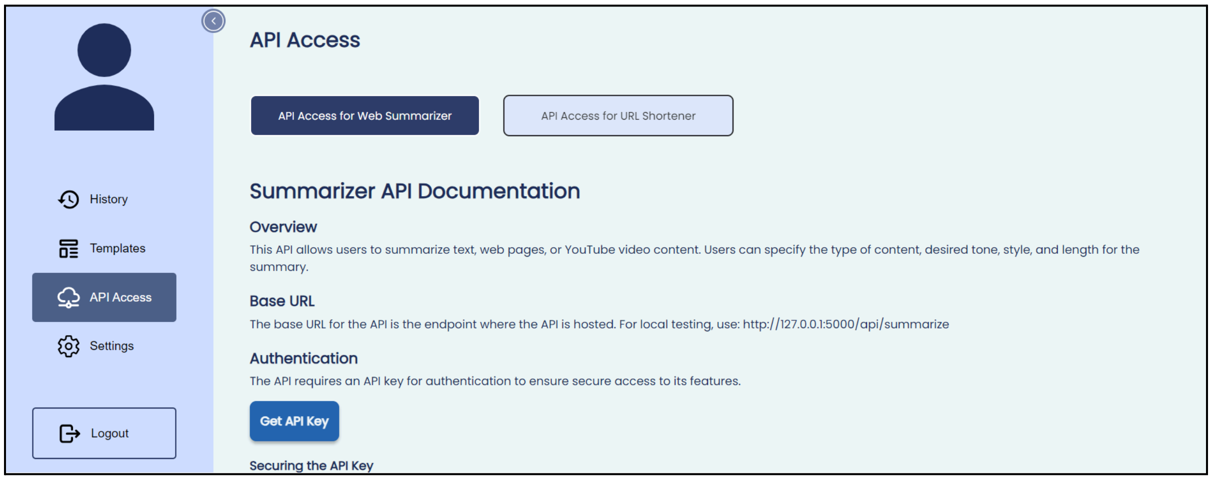
The history page allows users to view the past saved summaries. and shortened URLs. Users can also delete or copy from the history.



3.8.3.2. API Access

Find the API documentation here: https://docs.google.com/document/d/1jqd61hFzwjPLxzOfzYHJyDC9N1FPZLNnyiui4gKeKsE/edit?usp=sharing

Premium users can access the API documentation by clicking on the 'API Access' button in the sidebar. Users can also get an API key by clicking on the 'Get API Key' button placed in the API access component.



API Access

Summarizer API Documentation

Overview

This API allows users to summarize text, web pages, or YouTube video content. Users can specify the type of content, desired tone, style, and length for the summary.

Base URL

The base URL for the API is the endpoint where the API is hosted. For local testing, use: http://127.0.0.1:5000/api/summarize

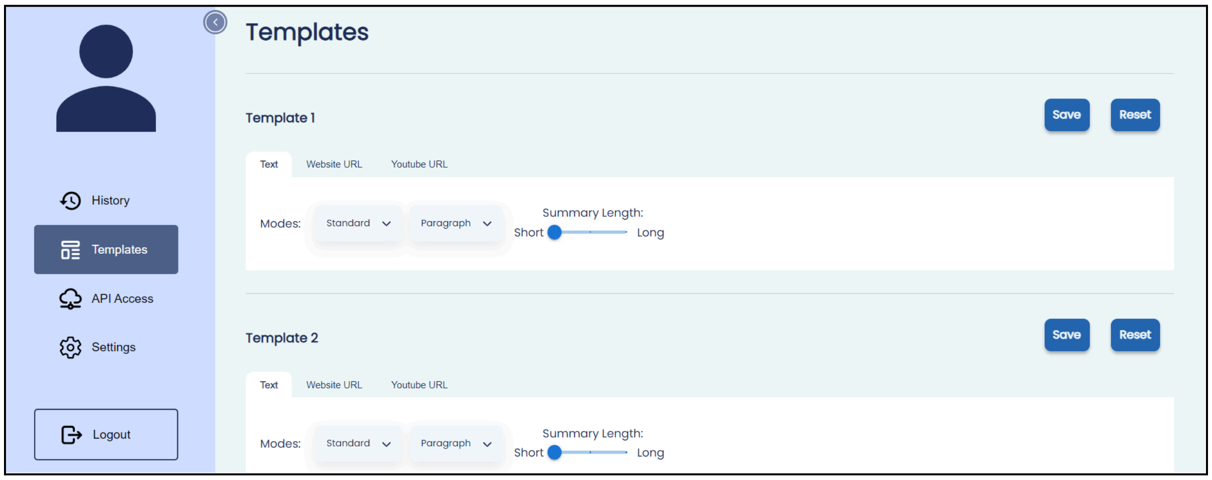
Authentication

The API requires an API key for authentication to ensure secure access to its features.

Securing the API Key

3.8.3.3. Templates

Users can save custom summarisation levels to be accessed later. The custom summarisation levels (3 templates for each user) are stored in the templates part of the user dashboard, where they can also edit a template. The templates can be loaded into the summarizer page for use.



Templates

Template 1

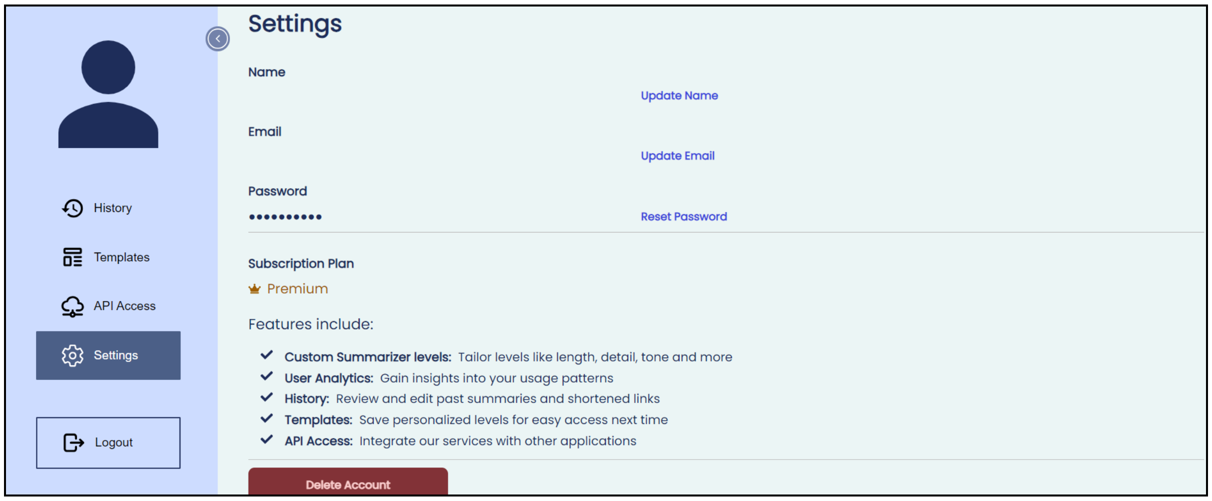
Modes: Standard Paragraph Summary Length: Short Long

Template 2

Modes: Standard Paragraph Summary Length: Short Long

3.8.3.4. Settings

Users can view or update the name, email and password associated with the shortify account and can choose to delete their account.


Settings

Name

Update Name

Email

Update Email

Password

••••••••••

Reset Password

Subscription Plan

Premium

Features include:

- ✓ **Custom Summarizer levels:** Tailor levels like length, detail, tone and more
- ✓ **User Analytics:** Gain insights into your usage patterns
- ✓ **History:** Review and edit past summaries and shortened links
- ✓ **Templates:** Save personalized levels for easy access next time
- ✓ **API Access:** Integrate our services with other applications

4. Technical Manual

Technology Stack

For producing the shortify application, we used the following technologies:

- **React** - Front-end Framework
  - React is a JavaScript library for building user interfaces, mainly for single-page applications. It's used for handling the view layer in web and mobile apps. React also allows you to design simple views for each state in your application, and it will efficiently update and render the right components when your data changes.
- **Flask** - Back-end Framework
  - Flask is a lightweight web application framework written in Python. It's designed to make getting started quick and easy, with the ability to scale up to complex applications. It's designed to provide the user with the basics and lets them pick and choose the rest.
- **PostgreSQL** - Database
  - Due to the nature of our application, the existence of users and the need for a database to support our URL shortening functionality, a database system is imperative. PostgreSQL was chosen for its versatility and ease of implementation. Our database is hosted on the cloud, allowing 24/7 connectivity. The precise architecture is explained later in this report.
- **Server Architecture**
  - **Ubuntu VPS** – An Ubuntu VPS (Virtual Private Server) was chosen as our hosting infrastructure, as it would allow for maximum customization given the diversity of our tech stack.
  - **Nginx** - Nginx installed on our VPS as opposed to Apache, as our web server. Nginx handles the routing and SSL certification for our website;

essentially connecting the users that land on our domain to our pages (index.html).

    - **Gunicorn** - Gunicorn is a WSGI (Web Server Gateway Interface) HTTP server for Python web applications, which allows us to run our multiple Flask servers on their dedicated ports on our VPS without requiring an active terminal. Its worker technology allows for concurrency and should, in theory, make our functionality faster.

- **Testing**
    - **Jest** - Front-end Testing Framework
        - Jest is a JavaScript testing framework with a focus on simplicity. It provides a complete set of features such as zero-configuration, snapshot testing, and test coverage.
    - **Pytest** - Back-end Testing Framework
        - Pytest is a Python testing tool for writing simple and scalable test cases.

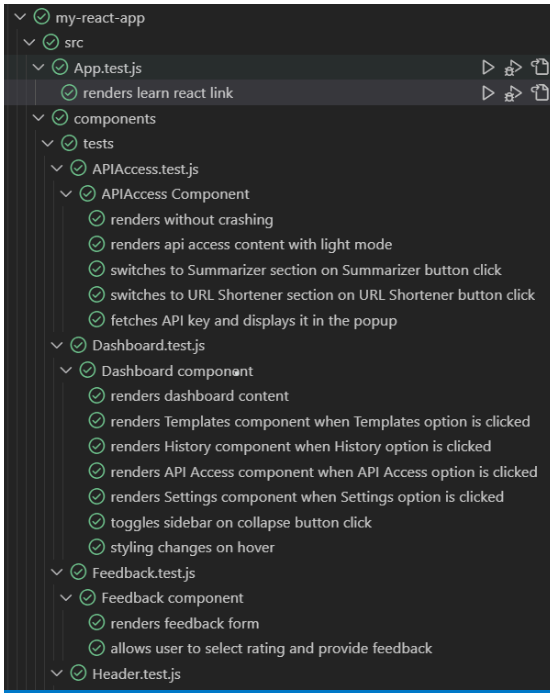
5. Installation Manual

Setting up the Frontend and Backend

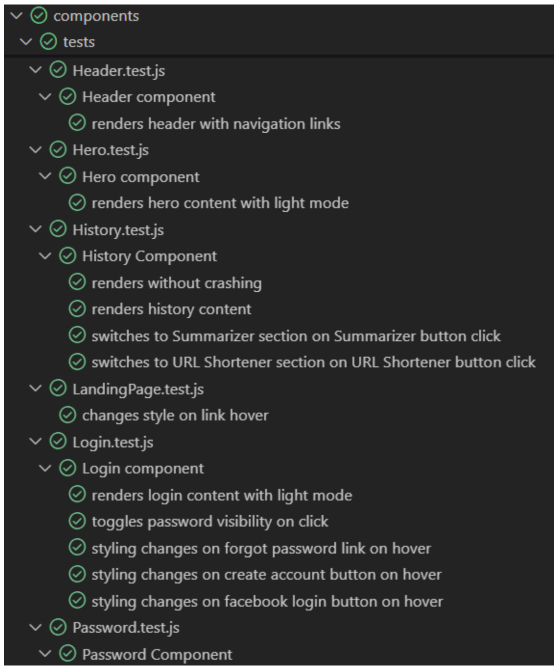
The Front-end & Back-end setup instructions can be accessed through the shortify github link below:

Front-end & Back-end Setup Instructions

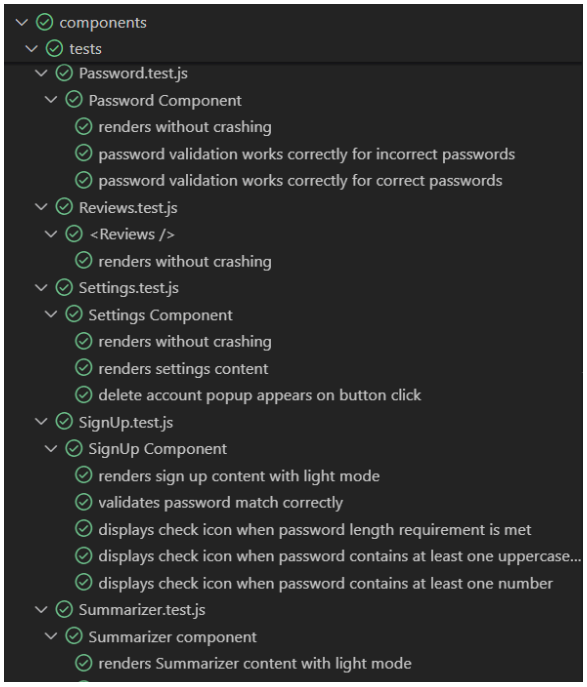
6. Frontend Test Documents/Results

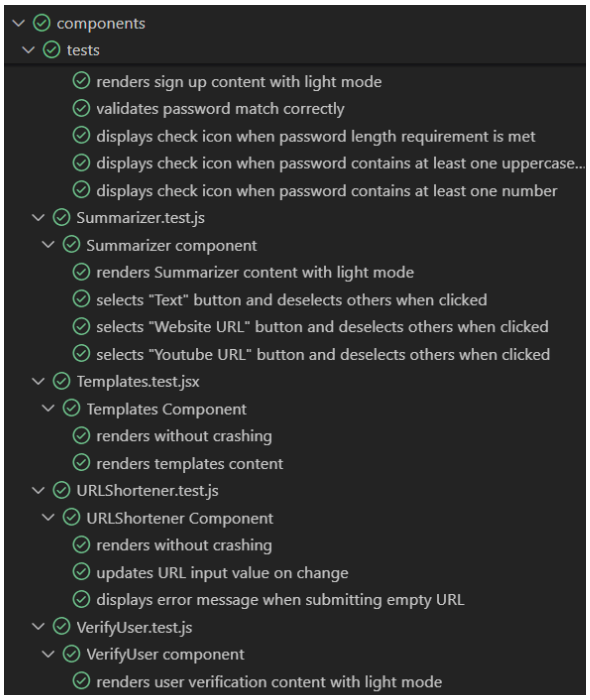


```
my-react-app
  src
    App.test.js
      renders learn react link
    components
      tests
        APIAccess.test.js
          APIAccess Component
            renders without crashing
            renders api access content with light mode
            switches to Summarizer section on Summarizer button click
            switches to URL Shortener section on URL Shortener button click
            fetches API key and displays it in the popup
        Dashboard.test.js
          Dashboard component
            renders dashboard content
            renders Templates component when Templates option is clicked
            renders History component when History option is clicked
            renders API Access component when API Access option is clicked
            renders Settings component when Settings option is clicked
            toggles sidebar on collapse button click
            styling changes on hover
        Feedback.test.js
          Feedback component
            renders feedback form
            allows user to select rating and provide feedback
        Header.test.js
```



```
components
  tests
    Header.test.js
      Header component
        renders header with navigation links
    Hero.test.js
      Hero component
        renders hero content with light mode
    History.test.js
      History Component
        renders without crashing
        renders history content
        switches to Summarizer section on Summarizer button click
        switches to URL Shortener section on URL Shortener button click
    LandingPage.test.js
      changes style on link hover
    Login.test.js
      Login component
        renders login content with light mode
        toggles password visibility on click
        styling changes on forgot password link on hover
        styling changes on create account button on hover
        styling changes on facebook login button on hover
    Password.test.js
      Password Component
```


```
components
  tests
    Password.test.js
      Password Component
        renders without crashing
        password validation works correctly for incorrect passwords
        password validation works correctly for correct passwords
    Reviews.test.js
      <Reviews />
        renders without crashing
    Settings.test.js
      Settings Component
        renders without crashing
        renders settings content
        delete account popup appears on button click
    SignUp.test.js
      SignUp Component
        renders sign up content with light mode
        validates password match correctly
        displays check icon when password length requirement is met
        displays check icon when password contains at least one uppercase...
        displays check icon when password contains at least one number
    Summarizer.test.js
      Summarizer component
        renders Summarizer content with light mode
```


```
components
  tests
        renders sign up content with light mode
        validates password match correctly
        displays check icon when password length requirement is met
        displays check icon when password contains at least one uppercase...
        displays check icon when password contains at least one number
    Summarizer.test.js
      Summarizer component
        renders Summarizer content with light mode
        selects "Text" button and deselects others when clicked
        selects "Website URL" button and deselects others when clicked
        selects "Youtube URL" button and deselects others when clicked
    Templates.test.jsx
      Templates Component
        renders without crashing
        renders templates content
    URLShortener.test.js
      URLShortener Component
        renders without crashing
        updates URL input value on change
        displays error message when submitting empty URL
    VerifyUser.test.js
      VerifyUser component
        renders user verification content with light mode
```

6.1. APIAccess

Positive Conditions:

Test Case 1: Renders without Crashing

- **Objective**: Ensure that the APIAccess component renders without any errors or crashes.

- **Preconditions**: None.
- **Steps**:
    - Render the APIAccess component within a Router.
    - Check if it renders successfully.
- **Expected Results**: The component renders without any errors.

Test Case 2: Renders API Access Content with Light Mode

- **Objective**: Verify that the APIAccess component renders necessary content related to API access in light mode.
- **Preconditions**: None.
- **Steps**:
    - Render the APIAccess component within a Router.
    - Check if specific text elements related to API access are present.
    - Check if the container for buttons has the expected class for light mode.
- **Expected Results**: Relevant text elements and button container are present, and the container has the correct class for light mode.

Test Case 3: Switches to Summarizer Section on Summarizer Button Click

- **Objective**: Ensure that clicking the Summarizer button switches to the Summarizer section within the APIAccess component.
- **Preconditions**: The APIAccess component is rendered.
- **Steps**:
    - Click the Summarizer button.
- **Expected Results**: The Summarizer section is displayed.

Test Case 4: Switches to URL Shortener Section on URL Shortener Button Click

- **Objective**: Ensure that clicking the URL Shortener button switches to the URL Shortener section within the APIAccess component.
- **Preconditions**: The APIAccess component is rendered.
- **Steps**:
    - Click the URL Shortener button.
- **Expected Results**: The URL Shortener section is displayed.

Test Case 5: Fetches API Key and Displays it in the Popup

- **Objective**: Verify that clicking the "Get API Key" button fetches the API key and displays it in the popup.
- **Preconditions**: The APIAccess component is rendered.
- **Steps**:
    - Mock the fetch API to return a test API key.
    - Click the "Get API Key" button.
- **Expected Results**: The popup displays "API key created", "Your API key", and the test API key.

6.2. Dashboard

Positive Conditions:

Test Case 1: Renders Dashboard Content

- **Objective:** Verify that the Dashboard component renders its content without errors.
- **Preconditions:** None.
- **Steps:**
  - Render the Dashboard component within a Router.
- **Expected Results:** The Dashboard content renders without any errors.

Test Case 2: Renders Templates Component when Templates Option is Clicked

- **Objective:** Ensure that clicking the "Templates" option renders the Templates component.
- **Preconditions:** The Dashboard component is rendered.
- **Steps:**
  - Click the "Templates" button.
- **Expected Results:** The Templates component is displayed with its content.

Test Case 3: Renders History Component when History Option is Clicked

- **Objective:** Ensure that clicking the "History" option renders the History component.
- **Preconditions:** The Dashboard component is rendered.
- **Steps:**
  - Click the "History" button.
- **Expected Results:** The History component is displayed with its content.

Test Case 4: Renders API Access Component when API Access Option is Clicked

- **Objective:** Ensure that clicking the "API Access" option renders the History component.
- **Preconditions:** The Dashboard component is rendered.
- **Steps:**
  - Click the "API Access" button.
- **Expected Results:** The API Access component is displayed with its content.

Test Case 5: Renders Settings Component when Settings Option is Clicked

- **Objective:** Ensure that clicking the "Settings" option renders the Settings component.
- **Preconditions:** The Dashboard component is rendered.
- **Steps:**

    - Click the "Settings" button.
- **Expected Results:** The Settings component is displayed with its content.

Test Case 6: Toggles Sidebar on Collapse Button Click

- **Objective:** Ensure that clicking the collapse button toggles the sidebar between large and small views.
- **Preconditions:** The Dashboard component is rendered.
- **Steps:**
    - Click the collapse button.
- **Expected Results:** The sidebar toggles between large and small views accordingly.

Test Case 7: Styling Changes on Hover

- **Objective:** Verify that the styling of options changes on mouse hover.
- **Preconditions:** The Dashboard component is rendered.
- **Steps:**
    - Hover over an option.
- **Expected Results:** The style of the option changes on hover.

6.3. Feedback

Test Case 1. Renders Feedback Form

- **Objective:** Ensure that the Feedback component renders the feedback form correctly.
- **Preconditions:** None.
- **Steps:**
    - Render the Feedback component.
- **Expected Result:** The feedback form includes elements such as "Rate your experience" text, a feedback input field, and a "Send" button.

Test Case 2. Allows User to Select Rating and Provide Feedback

- **Objective:** Verify that the Feedback component allows users to select a rating and provide feedback.
- **Preconditions:** The Feedback component is rendered.
- **Steps:**
    - Render the Feedback component.
    - Select a rating.
    - Enter feedback in the input field.
    - Click the "Send" button.
- **Expected Result:** The component successfully submits the feedback, and a "Thank you for your feedback!" message appears.

6.4. Header

Test Case 1. Renders Header with Navigation Links

- **Objective:** Ensure that the Header component renders navigation links correctly.
- **Preconditions:** None.
- **Steps:**
    - Render the Header component within a BrowserRouter.
- **Expected Result:** The component displays the "shortify" logo along with "SUMMARIZER" and "URL SHORTENER" navigation links.

6.5. Hero

Test Case 1. Renders Hero Content with Light Mode

- **Objective:** Ensure that the Hero component renders its content correctly in light mode.
- **Preconditions:** None.
- **Steps:**
    - Render the Hero component within a Router.
- **Expected Result:** The component displays the appropriate text content and the background color is light.

6.6. History

Test Case 1. Renders Without Crashing

- **Objective:** Ensure that the History component renders without any errors.
- **Preconditions:** None.
- **Steps:**
    - Render the History component within a Router.
- **Expected Result:** The component renders without any errors.

Test Case 2. Renders History Content

- **Objective:** Verify that the History component renders its content correctly.
- **Preconditions:** The History component is rendered.
- **Steps:**
    - Render the History component within a Router.
- **Expected Result:** The component displays the title "History" along with sections for "Web Summarizer History" and "URL Shortener History".

Test Case 3. Switches to Summarizer Section on Summarizer Button Click

- **Objective:** Ensure that clicking the "Web Summarizer History" button switches to the Summarizer section.
- **Preconditions:** The History component is rendered.
- **Steps:**
  - Render the History component.
  - Click the "Web Summarizer History" button.
- **Expected Result:** The Summarizer section is displayed, showing content related to summarized prompts.

Test Case 4. Switches to URL Shortener Section on URL Shortener Button Click

- **Objective:** Ensure that clicking the "URL Shortener History" button switches to the URL Shortener section.
- **Preconditions:** The History component is rendered.
- **Steps:**
  - Render the History component.
  - Click the "URL Shortener History" button.
- **Expected Result:** The URL Shortener section is displayed, showing content related to shortened links.

6.7. Landing Page

Test Case 1. Changes Style on Link Hover

- **Objective:** Verify that the style changes correctly when hovering over the Summarizer and URL Shortener links.
- **Preconditions:** The LandingPage component is rendered.
- **Steps:**
  1. Render the LandingPage component within a Router.
  2. Find the Summarizer and URL Shortener links.
  3. Get the initial computed style of both links.
  4. Simulate a mouse enter event on the Summarizer link.
  5. Get the computed style of the Summarizer link after the mouse enter event.
  6. Assert that the style of the Summarizer link has changed.
  7. Simulate a mouse enter event on the URL Shortener link.
  8. Get the computed style of the URL Shortener link after the mouse enter event.
  9. Assert that the style of the URL Shortener link has changed.
- **Expected Result:** The style of both links changes when hovered over.

6.8. Login

Test Case 1: Renders Login Content with Light Mode

- **Objective:** Verify that the LogIn component renders its content correctly in light mode.
- **Preconditions:** None.
- **Steps:**
    - Render the LogIn component within a Router wrapped with AuthProvider.
- **Expected Results:** The login form, including email and password fields, create account link, login button, and logo, is rendered with appropriate styling for light mode.

Test Case 2: Toggles Password Visibility on Click

- **Objective:** Ensure that clicking the password visibility button toggles the visibility of the password field.
- **Preconditions:** The LogIn component is rendered.
- **Steps:**
    - Click the password visibility button.
- **Expected Results:** The password field toggles between displaying plaintext and password characters.

Test Case 3: Styling Changes on Forgot Password Link on Hover

- **Objective:** Verify that the styling of the "Forgot your password?" link changes on mouse hover.
- **Preconditions:** The LogIn component is rendered.
- **Steps:**
    - Hover over the "Forgot your password?" link.
- **Expected Result:** The style of the link changes on hover.

Test Case 4: Styling Changes on Create Account Button on Hover

- **Objective:** Ensure that the styling of the "Create an account" button changes on mouse hover.
- **Preconditions:** The LogIn component is rendered.
- **Steps:**
    - Hover over the "Create an account" button.
- **Expected Result:** The style of the button changes on hover.

Test Case 5: Styling Changes on Facebook Login Button on Hover

- **Objective:** Verify that the styling of the "Log in with Facebook" button changes on mouse hover.
- **Preconditions:** The LogIn component is rendered.
- **Steps:**
    - Hover over the "Log in with Facebook" button.
- **Expected Result:** The style of the button changes on hover.

6.9. Password

Test Case 1: Renders Without Crashing

- **Objective:** Ensure that the Password component renders without any errors.
- **Preconditions:** None.
- **Steps:**
  - Render the Password component within a Router.
- **Expected Result:** The component renders without any errors.

Test Case 2: Password Validation Works Correctly for Incorrect Passwords

- **Objective:** Verify that password validation works correctly for incorrect passwords.
- **Preconditions:** The Password component is rendered.
- **Steps:**
  - Enter an incorrect password in the password and confirm password fields.
  - Click the "Reset password" button.
- **Expected Result:** The component displays a message indicating that the passwords do not match.

Test Case 3: Password Validation Works Correctly for Correct Passwords

- **Objective:** Verify that password validation works correctly for correct passwords.
- **Preconditions:** The Password component is rendered.
- **Steps:**
  - Enter a correct password in the password and confirm password fields.
  - Click the "Reset password" button.
- **Expected Result:** The component does not display a message indicating that the passwords do not match.

6.10. Reviews

Test Case 1. Renders Without Crashing

- **Objective:** Ensure that the Reviews component renders without any errors.
- **Preconditions:** None.
- **Steps:**
  - Render the Reviews component.
- **Expected Result:** The component renders without any errors.

6.11. Settings

Test Case 1: Renders Without Crashing

- **Objective:** Ensure that the Settings component renders without any errors.

- **Preconditions:** None.
- **Steps:**
  - Render the Settings component within a Router.
- **Expected Result:** The component renders without any errors.

Test Case 2: Renders Settings Content

- **Objective:** Verify that the Settings component renders its content correctly.
- **Preconditions:** The Settings component is rendered.
- **Steps:**
  - Render the Settings component within a Router.
- **Expected Result:** The component displays the title "Settings" and the "Subscription Plan" text.

Test Case 3: Delete Account Popup Appears on Button Click

- **Objective:** Ensure that clicking the "Delete Account" button triggers the appearance of the delete account popup.
- **Preconditions:** The Settings component is rendered.
- **Steps:**
  - Render the Settings component within a Router.
  - Click the "Delete Account" button.
- **Expected Result:** The delete account popup appears with the message "Are you sure you want to delete your account permanently?".

6.12. SignUp

Test Case 1: Renders Sign Up Content with Light Mode

- **Objective:** Ensure that the SignUp component renders its content correctly in light mode.
- **Preconditions:** None.
- **Steps:**
  - Render the SignUp component within a Router.
- **Expected Result:** The component displays the introductory text, name input field, and password length requirement text with appropriate styling for light mode.

Test Case 2: Validates Password Match Correctly

- **Objective:** Verify that the SignUp component correctly validates whether the password and confirm password fields match.
- **Preconditions:** The SignUp component is rendered.
- **Steps:**
  - Enter the same password in both the password and confirm password fields.
  - Click the "Create Account" button.

- **Expected Result:** The component does not display a message indicating that the passwords do not match.

Test Case 3: Displays Check Icon When Password Length Requirement is Met

- **Objective:** Ensure that the SignUp component displays a check icon when the password length requirement is met.
- **Preconditions:** The SignUp component is rendered.
- **Steps:**
  - Enter a password with the required length.
- **Expected Result:** The component displays a check icon next to the password length requirement.

Test Case 4: Displays Check Icon When Password Contains at Least One Uppercase Letter

- **Objective:** Verify that the SignUp component displays a check icon when the password contains at least one uppercase letter.
- **Preconditions:** The SignUp component is rendered.
- **Steps:**
  - Enter a password containing at least one uppercase letter.
- **Expected Result:** The component displays a check icon next to the uppercase letter requirement.

Test Case 5: Displays Check Icon When Password Contains at Least One Number

- **Objective:** Ensure that the SignUp component displays a check icon when the password contains at least one number.
- **Preconditions:** The SignUp component is rendered.
- **Steps:**
  - Enter a password containing at least one number.
- **Expected Result:** The component displays a check icon next to the number requirement.

6.13. Summarizer

Positive Conditions:

Test Case 1: Renders Summarizer Component with Light Mode

- **Objective**: Verify that the component is rendered successfully with light mode
- **Preconditions**: Render the Summarizer component within a Router.
- **Steps**:
  - Call the render function to check if the Summarizer component is returned on the screen as truthy.

- **Expected Results**: The Summarizer component is rendered successfully with light mode.

Test Case 2: Text Summary Mode Switch

- **Objective**: Verify that the ‘Text’ input summary mode is chosen corresponding to the correct color switches.
- **Preconditions**: Render the Summarizer component within a Router.
- **Steps**:
    - Click the ‘Text’ button.
- **Expected Results**: The ‘Text’ button lights up with the right classes attached to its element.

Test Case 3: Website URL Summary Mode Switch

- **Objective**: Verify that the ‘Website URL’ input summary mode is chosen corresponding to the correct color switches.
- **Preconditions**: Render the Summarizer component within a Router.
- **Steps**:
    - Click the ‘Website URL’ button.
- **Expected Results**: The ‘Website URL’ button lights up with the right classes attached to its element.

Test Case 4: Youtube URL Summary Mode Switch

- **Objective**: Verify that the ‘Youtube URL’ input summary mode is chosen corresponding to the correct color switches.
- **Preconditions**: Render the Summarizer component within a Router.
- **Steps**:
    - Click the ‘Youtube URL’ button.
- **Expected Results**: The ‘Youtube URL’ button lights up with the right classes attached to its element.

Test Case 5: Typing into the Input Field Updates InputContent Value

- **Objective**: Verify that typing into the input text field updates the value of the input content that will be used for summarization.
- **Preconditions**: Render the Summarizer component.
- **Steps**:
    - Change the value of the input text field to “Test input”.
- **Expected Results**: The input field’s content is updated to “Test input”.

6.14. Templates

Test Case 1: Renders Without Crashing

- **Objective:** Ensure that the Templates component renders without any errors.
- **Preconditions:** None.
- **Steps:**
    - Render the Templates component within a Router.
- **Expected Result:** The component renders without any errors.

Test Case 2: Renders Templates Content

- **Objective:** Verify that the Templates component renders its content correctly.
- **Preconditions:** The Templates component is rendered.
- **Steps:**
    - Render the Templates component within a Router.
- **Expected Result:** The component displays the titles of all available templates ("Template 1", "Template 2", "Template 3") and the "Summarize" button container with appropriate styling for light mode.

6.15. URL Shortener

Test Case 1. Renders Without Crashing

- **Objective:** Ensure that the URLShortener component renders without any errors.
- **Preconditions:** None.
- **Steps:**
    - Render the URLShortener component.
- **Expected Result:** The component renders without any errors.

Test Case 2. Updates URL Input Value on Change

- **Objective:** Verify that the URL input value updates correctly when changed by the user.
- **Preconditions:** The URLShortener component is rendered.
- **Steps:**
    - Render the URLShortener component.
    - Change the value of the URL input field.
- **Expected Result:** The value of the URL input field updates to the new value provided by the user.

Test Case 3. Displays Error Message When Submitting Empty URL

- **Objective:** Ensure that an error message is displayed when the user submits an empty URL.
- **Preconditions:** The URLShortener component is rendered.
- **Steps:**
    - Render the URLShortener component.
    - Leave the URL input field empty.
    - Click the "Shorten URL" button.

- **Expected Result:** The component displays an error message indicating that the user should enter a URL to be shortened.

6.16. Verify User

Test Case 1: Renders User Verification Content with Light Mode

- **Objective:** Ensure that the VerifyUser component renders its content correctly in light mode.
- **Preconditions:** None.
- **Steps:**
    - Render the VerifyUser component within a Router.
- **Expected Result:** The component displays the title "Verifying User", an email input field with a placeholder text, and a message prompting the user to reset the password if forgotten. The component's container has appropriate styling for light mode.

7. Backend Implementation and Test Documents/Results

7.1. Text Extraction

7.1.1. Text Extraction Test Documents

Test Case 1: Text Extraction from a Valid URL

- **Objective**: Verify that text can be successfully extracted from a valid URL using the web text extraction service.
- **Preconditions**: Web text extraction service is properly configured with internet access.
- **Steps**:
    - Submit a POST request to the text extraction endpoint with a valid URL.
- **Expected Result**: The code successfully extracts text and returns it without any errors.

Test Case 2: Text Extraction Handling with Extraction Error

- **Objective**: Validate that the text extraction service can handle and log errors appropriately when an extraction error occurs.
- **Preconditions**: Web text extraction service is equipped to handle and log errors.
- **Steps**:

    - Submit a POST request to the text extraction endpoint with a URL known to cause extraction errors.
- **Expected Result**: The code attempts text extraction, logs the extraction error, and returns an appropriate error message.

Test Case 3: Handling of Invalid URL

- **Objective**: Ensure that the web text extraction service can identify and handle invalid URLs appropriately.
- **Preconditions**: Web text extraction service must validate URL format before processing.
- **Steps**:
    - Submit a POST request to the text extraction endpoint with an invalid URL.
- **Expected Result**: The code validates the URL, identifies it as invalid, and returns an appropriate error code.

Test Case 4: Error Handling When URL is Missing

- **Objective**: Confirm that the web text extraction service correctly handles cases where no URL is provided in the request.
- **Preconditions**: Text extraction endpoint expects a URL parameter in the request.
- **Steps**:
    - Submit a POST request to the text extraction endpoint without a URL.
- **Expected Result**: The code checks for URL presence, finds it missing, and returns an error code indicating that no URL was provided.

Test Case 5: Handling Non-GET Method Usage

- **Objective:** Test that the web text extraction service properly handles and rejects non-GET methods if not configured to accept them.
- **Preconditions:** The endpoint is configured to accept only POST requests for text extraction.
- **Steps:**
    - Submit a GET request to the text extraction endpoint.
- **Expected Result:** The code returns a 405 Method Not Allowed status code.

Test Case 6: Handling Non-HTML URL

- **Objective:** Ensure that the web text extraction service can appropriately handle and respond to URLs pointing to non-HTML content.
- **Preconditions:** Web text extraction service should check the content type of the URL before attempting text extraction.
- **Steps:**
    - Submit a POST request to the text extraction endpoint with a URL linking to a non-HTML resource (e.g., a PDF file).

- **Expected Result:** The code recognizes the non-HTML content and returns an error message about the inability to extract text.

Test Case 7: Response Time for Web Text Extraction

- **Objective:**Verify that the response time for extracting text from a URL meets performance expectations.
- **Preconditions:** The server and network conditions should be optimal for testing to avoid external delays.
- **Steps:**
  - Submit a POST request to the text extraction endpoint with a valid URL and measure the time taken for the response.
- **Expected Result:** The response time from the initiation of the request to the retrieval of text is within the specified range of 15-20 seconds.

7.2. YouTube Caption Extraction

7.2.1. YouTube Caption Extraction Test Documents

1. Test Case: Conversion of Time to Seconds

- **Objective:** Verify that the convert_time_to_seconds function correctly converts time in HH:MM format to seconds.
- **Preconditions:** The function must handle various formats and constraints of time input.
- **Steps:**
  - Test with inputs "02:30", "00:45", "01:00", "24:00", and "30:00".
  - Test with invalid inputs "100", "one:two", "13:61", and "00:-1".
- **Expected Result:** The function returns correct second values for valid inputs and raises ValueError for invalid inputs.

2. Test Case: Validation of Time Within Video Duration

- **Objective:** Ensure the validate_time_within_duration function accurately checks if given start and end times are within the video duration and in the correct order.
- **Preconditions:** Inputs include start time, end time, and total video duration.
- **Steps:**
  - Validate with a correct order of times (start_time=100, end_time=200, duration_seconds=300).
  - Test incorrect scenarios: end time before start time, start time or end time exceeding the video duration.
- **Expected Result:** The function passes without errors for valid timings and raises ValueError for invalid timings.

3. Test Case: Downloading Audio from YouTube

- **Objective:** Test the download_audio function to ensure it can download audio from a YouTube URL and handle the process correctly.
- **Preconditions:** The function utilizes yt_dlp.YoutubeDL for downloading.
- **Steps:**
    - Mock the yt_dlp.YoutubeDL class to simulate the downloading process.
    - Call download_audio with a dummy URL.
- **Expected Result:** Returns the expected filename and correctly uses the yt_dlp.YoutubeDL library.

4. Test Case: Audio Transcription Completion

- **Objective:** Verify that the transcribe_audio function can start a transcription job and handle its completion status correctly.
- **Preconditions:** Use of AWS Transcribe service; mocking of Boto3 and request handling.
- **Steps:**
    - Mock AWS Transcribe client and the requests library to simulate transcription job progression and completion.
    - Call transcribe_audio with a test audio file path.
- **Expected Result:** The function retrieves the transcription result upon job completion.

5. Test Case: Successful Upload of File to S3

- **Objective:** Ensure the upload_file_to_s3 function can successfully upload a file to an AWS S3 bucket.
- **Preconditions:** Mocking of the Boto3 S3 client.
- **Steps:**
    - Mock the S3 client to simulate a successful upload.
    - Call upload_file_to_s3 with a test file path.
- **Expected Result:** The function returns True, indicating a successful upload.

6. Test Case: Handling Failure in Upload to S3

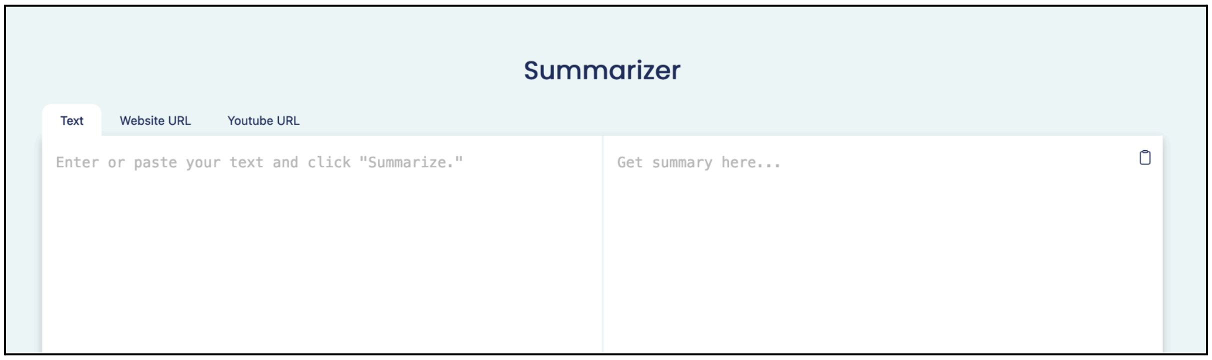
- **Objective:** Test the upload_file_to_s3 function's ability to handle upload failures.
- **Preconditions:** Mocking of the Boto3 S3 client to simulate an upload failure.
- **Steps:**
    - Mock the S3 client to raise an S3UploadFailedError during file upload.
    - Call upload_file_to_s3 with a test file path.
- **Expected Result:** The function returns False, indicating a failed upload.

7.3. Summarizer

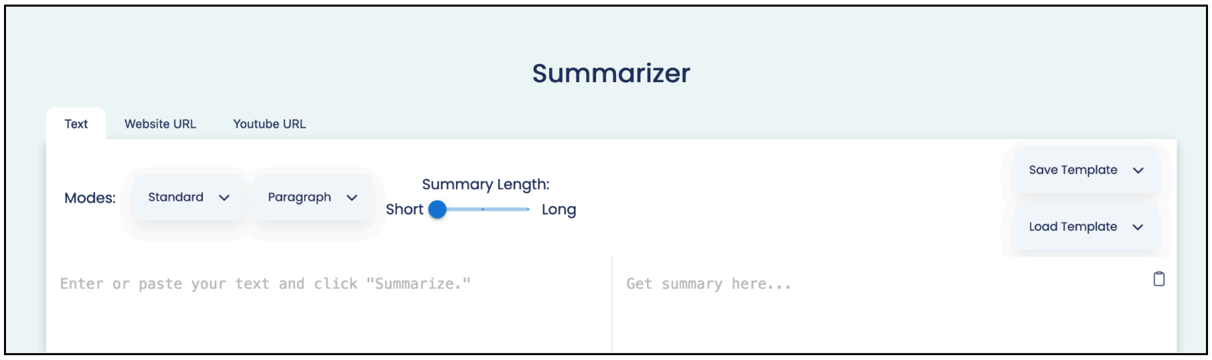
The final additions to the summarizer include premium users access to pro features, the method for YouTube video caption extraction as discussed earlier, timestamps implementation, and the addition of various citations for webpages, with some other smaller features like improved json file exportation.

The many modes the summarizer has is intended for pro users only (logged in users), so the summarizer will not include tones, style, length, citations, timestamp options, and the ability to save the modes as templates for non-pro users. This was implemented by checking whether users were logged in or not, and removing the options all together when not logged in. Logged in users are also not limited by any word count limits.

Logged out users UI (no mode options):



Logged in users UI (all mode options):

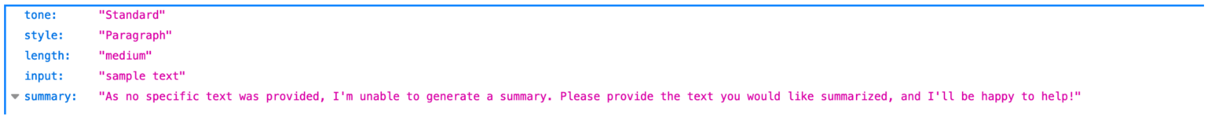


Timestamps have been fully implemented, allowing users to input start time and end time of their videos, and receive a summary between the given times, from the Youtube captions or speech to text transcript, that is then passed to the gpt-4 model for summarization. YouTube captions are used if available, and in cases where they are not, we use the downloaded audio speech to text transcript for the summary.

Citations have also now been included into the summarizer for web page summarization. The user is able to select from 3 different citation styles, MLA, APA, and Chicago. The citations will be generated at the end of each summary, using the appropriate citation term, like “reference” for APA, and “Bibliography” for Chicago style. This was implemented by sending a citation style specific prompt to the AI, since without it, citations were inconsistent.

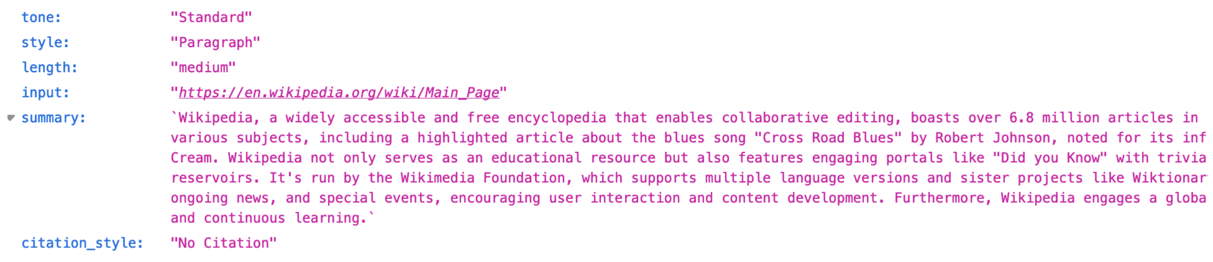
Finally, the summarizer improves summary exportation by fully utilizing json files' ability to include different types of content. The files now include summary type specific content as shown at: https://github.com/Attested-paper51/COSC-4P02-Web-Summarizer/tree/main/Sample%20Exports

Text summary exports include the tone, style length, input (the text to summarize) and the summary itself.



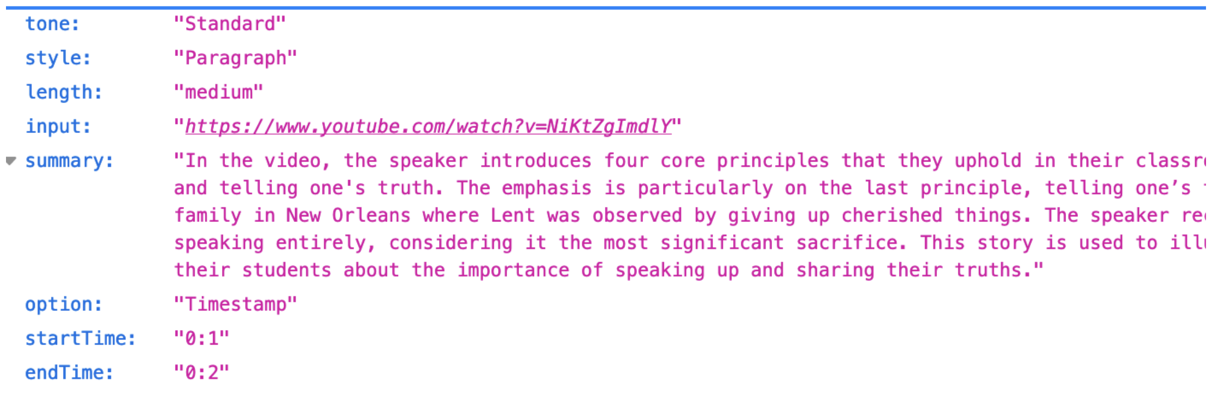
```
tone:       "Standard"
style:      "Paragraph"
length:     "medium"
input:      "sample text"
summary:    "As no specific text was provided, I'm unable to generate a summary. Please provide the text you would like summarized, and I'll be happy to help!"
```

Web page summary exports include the tone, style length, input (the webpage URL to summarize), the summary itself, and the citation style used, if any.



```
tone:            "Standard"
style:           "Paragraph"
length:          "medium"
input:           "https://en.wikipedia.org/wiki/Main_Page"
summary:         `Wikipedia, a widely accessible and free encyclopedia that enables collaborative editing, boasts over 6.8 million articles in
                 various subjects, including a highlighted article about the blues song "Cross Road Blues" by Robert Johnson, noted for its inf
                 Cream. Wikipedia not only serves as an educational resource but also features engaging portals like "Did you Know" with trivia
                 reservoirs. It's run by the Wikimedia Foundation, which supports multiple language versions and sister projects like Wiktionar
                 ongoing news, and special events, encouraging user interaction and content development. Furthermore, Wikipedia engages a globa
                 and continuous learning.`
citation_style:  "No Citation"
```

Lastly, YouTube summary exports include the tone, style length, input (the YouTube video URL to summarize), the summary itself, and the option of full video summarization or timestamp summarization, with timestamp summarization including the times the video was summarized for in HH:MM format for start and end times.



```
tone:       "Standard"
style:      "Paragraph"
length:     "medium"
input:      "https://www.youtube.com/watch?v=NiKtZgImdlY"
summary:    "In the video, the speaker introduces four core principles that they uphold in their classr
            and telling one's truth. The emphasis is particularly on the last principle, telling one’s
            family in New Orleans where Lent was observed by giving up cherished things. The speaker re
            speaking entirely, considering it the most significant sacrifice. This story is used to ill
            their students about the importance of speaking up and sharing their truths."
option:     "Timestamp"
startTime:  "0:1"
endTime:    "0:2"
```

7.3.1. Summarizer Test Cases

The following are the test cases for the backend of the summarizer, all the tests, and test results conducted can be found at:
https://github.com/Attested-paper51/COSC-4P02-Web-Summarizer/tree/main/WebSummarizerShortner/server/tests/BackEndTests

The BackEndTests folder contains the unit tests conducted which all passed successfully, and it also contains the results from the tests, which include the exact parameters passed and the responses received after each test.

Below are the test cases for each test conducted, this has been updated to include the addition of YouTube video summarization (full and timestamps) and all citation styles available. Our frontend uses the API to talk to the backend. For more information on how the API works, the API documentation can be found at:
https://github.com/Attested-paper51/COSC-4P02-Web-Summarizer/blob/main/API%20Documentation%20for%20Text%20Summarization%20Service.pdf

7.3.2. Text Summarization Test Cases

Positive Conditions:

Test Case 1: Formal Tone with Bullet Points

- **Objective**: Ensure API returns a correctly formatted summary with a formal tone in bullet points style.
- **Preconditions**: API is functioning; input text is prepared.
- **Inputs**: Formal tone, bullet points style, short length.
- **Steps**:
    - Send a POST request to the API with the specified parameters.
    - Validate that the response contains appropriate style for summarizing the text in the specified tone.
- **Expected Results**: The API returns a summary with proper tone and style.

Test Case 2: Formal Tone with Numbered List

- **Objective**: Validate the API's capability to summarize text with a formal tone in a numbered list style.
- **Preconditions**: API is available; suitable text for formal tone is provided.
- **Inputs**: Formal tone, numbered list style, short length.
- **Steps**:
    - Send a POST request to the API with the specified parameters.
    - Validate that the response contains appropriate style for summarizing the text in the specified tone.

- **Expected Results**: The API returns a summary with proper tone and style.

Test Case 3: Formal Tone with Paragraph

- **Objective**: Test that the API can generate a summary with a formal tone in paragraph form.
- **Preconditions**: API is responsive; the input text fits a formal context.
- **Inputs**: Formal tone, paragraph style, medium length.
- **Steps**:
    - Send a POST request to the API with the specified parameters.
    - Validate that the response contains appropriate style for summarizing the text in the specified tone.
- **Expected Results**: The API returns a summary with proper tone and style.

Test Case 4: Casual Tone with Bullet Points

- **Objective**: Confirm that the API provides a casual tone summary in bullet points format.
- **Preconditions**: API is operational; input text matches casual style.
- **Inputs**: Casual tone, bullet points style, long length.
- **Steps**:
    - Send a POST request to the API with the specified parameters.
    - Validate that the response contains appropriate style for summarizing the text in the specified tone.
- **Expected Results**: The API returns a summary with proper tone and style.

Test Case 5: Casual Tone with Numbered List

- **Objective**: Test API for summarizing text in a casual tone with a numbered list presentation.
- **Preconditions**: API is live; text for casual tone summary is ready.
- **Inputs**: Casual tone, numbered list style, medium length.
- **Steps**:
    - Send a POST request to the API with the specified parameters.
    - Validate that the response contains appropriate style for summarizing the text in the specified tone.
- **Expected Results**: The API returns a summary with proper tone and style.

Test Case 6: Casual Tone with Paragraph

- **Objective**: Check if the API can summarize text in a casual tone, presented in a paragraph.
- **Preconditions**: API is active; input text suitable for casual tone.
- **Inputs**: Casual tone, paragraph style, short length.
- **Steps**:
    - Send a POST request to the API with the specified parameters.

    - Validate that the response contains appropriate style for summarizing the text in the specified tone.
- **Expected Results**: The API returns a summary with proper tone and style.

Test Case 7: Sarcastic Tone with Bullet Points

- **Objective**: Ensure the API can create a sarcastic tone summary in bullet points style.
- **Preconditions**: API is accessible; text can be interpreted with a sarcastic tone.
- **Inputs**: Sarcastic tone, bullet points style, medium length.
- **Steps**:
    - Send a POST request to the API with the specified parameters.
    - Validate that the response contains appropriate style for summarizing the text in the specified tone.
- **Expected Results**: The API returns a summary with proper tone and style.

Test Case 8: Sarcastic Tone with Numbered List

- **Objective**: Assess the API's ability to summarize text with a sarcastic tone in numbered list format.
- **Preconditions**: API is up; text appropriate for sarcastic tone is used.
- **Inputs**: Sarcastic tone, numbered list style, long length.
- **Steps**:
    - Send a POST request to the API with the specified parameters.
    - Validate that the response contains appropriate style for summarizing the text in the specified tone.
- **Expected Results**: The API returns a summary with proper tone and style.

Test Case 9: Sarcastic Tone with Paragraph

- **Objective**: Verify the API's capacity to deliver a summary with a sarcastic tone in paragraph style.
- **Preconditions**: API endpoint is working; suitable sarcastic text is available.
- **Inputs**: Sarcastic tone, paragraph style, short length.
- **Steps**:
    - Send a POST request to the API with the specified parameters.
    - Validate that the response contains appropriate style for summarizing the text in the specified tone.
- **Expected Results**: The API returns a summary with proper tone and style.

Negative Conditions:

Test Case 4: Empty Text Input

- **Objective**: Test the API's response to an empty text input.

- **Preconditions**: API is operational.
- **Inputs**: An empty string for text input, with valid other parameters.
- **Steps**:
    1. Send a POST request with an empty string for the text input.
    2. Assess the API's response for handling empty input.
- **Expected Results**: The API returns an error message for missing or empty text input.

Test Case 5: Invalid Summarization Type

- **Objective**: Verify the API's error handling for a summarization content type, for example, not on youtube summarization, when should be on text summarization.
- **Preconditions**: API is running; other parameters are set correctly.
- **Inputs**: A valid text input with an invalid content type parameter.
- **Steps**:
    1. Submit a request with an invalid content type.
    2. Check for an appropriate error message response.
- **Expected Results**: The API responds with an error indicating an invalid input type.

Test Case 6: Invalid API Key

- **Objective**: Validate the API's authentication mechanism by using an incorrect API key.
- **Preconditions**: API endpoint is reachable.
- **Inputs**: All necessary request parameters with an invalid API key.
- **Steps**:
    1. Execute a request with an invalid API key.
    2. Evaluate the response for proper authentication enforcement.
- **Expected Results**: The API responds with an error message regarding the invalid or missing API key.

Text Summarization Test Results

Test Name: Test Tone: Formal, Style: Bullet Points, Length: short

```
Request: {
        "key": "frontend",
        "input": "Comprehensive testing of tone, style, and length.",
        "type": 0,
        "tone": "Formal",
        "style": "Bullet Points",
        "length": "short"
}
```

Response: {

"summary": "- Testing of tone, style, and length is essential for ensuring comprehensive evaluation of written content.\n- It helps in determining the appropriateness of writing style and tone for the intended audience.\n- Evaluating the length of the content ensures that it is concise and effectively conveys the message.\n- Comprehensive testing of tone, style, and length is crucial for enhancing the overall quality of written materials."

}

Test Name: Test Tone: Casual, Style: Numbered List, Length: long

```
Request: {
	"key": "frontend",
	"input": "Comprehensive testing of tone, style, and length.",
	"type": 0,
	"tone": "Casual",
	"style": "Numbered List",
	"length": "long"
}
```

Response: {

"summary": "1. Let's break it down, I'm here to summarize! \n2. Basically, they want me to test how I can summarize in different tones, styles, and lengths. \n3. No worries, I'll make sure to keep it casual and easygoing while summarizing the text. \n4. So, I'll be providing summaries in a numbered list format to keep things organized. \n5. Time to show off my summarization skills and have some fun with it!"

}

Test Name: Test Tone: Sarcastic, Style: Paragraph, Length: medium

```
Request: {
	"key": "frontend",
	"input": "Comprehensive testing of tone, style, and length.",
	"type": 0,
	"tone": "Sarcastic",
	"style": "Paragraph",
	"length": "medium"
}
```

Response: {

"summary": "Oh, how delightful it is to spend endless hours meticulously testing the tone, style, and length of a document. What a thrilling and invigorating task, truly a joy to behold. So much excitement in ensuring every word, punctuation mark, and paragraph is perfectly calibrated to meet the highest standards. A truly riveting experience, one that certainly does not feel like a tedious and monotonous chore at all."

}

Test Name: Empty Text Input Test

```
Request: {
        "key": "frontend",
        "input": "",
        "type": 0,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "medium"
}

Response: {

        "error": "Missing or empty text"

}
```

Test Name: Invalid Input Type Test

```
Request: {
        "key": "frontend",
        "input": "This should be treated as text, but is mislabeled.",
        "type": 7,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "medium"
}

Response: {

        "error": "Invalid input type"

}
```

Test Name: Invalid API Key Test

```
Request: {
        "key": "wrong_key",
        "input": "Sample text",
        "type": 0,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "medium"
}

Response: {

        "error": "Invalid or missing api key"

}
```

7.3.3. Web Page Summarization Test Cases

Positive Conditions:

Test Case 1: Standard Tone with APA Citation

- **Objective**: Verify that the API returns a correct summary with APA citation for a webpage.
- **Preconditions**: API is online; the webpage URL is valid and contains citable content.
- **Steps**:
    1. Send a POST request to the API with the parameters set for standard tone, paragraph style, short length, and APA citation format.
    2. Verify that the response contains a summary with APA citation.
- **Expected Results**: The API returns a summary in a standard tone with correctly formatted APA citation.

Test Case 2: Standard Tone with MLA Citation

- **Objective**: Ensure the API correctly summarizes a web page with MLA citation in standard tone.
- **Preconditions**: API is operational; the webpage URL is valid with content that can be cited in MLA format.
- **Steps**:
    1. Issue a POST request to the API with parameters for standard tone, paragraph style, short length, and MLA citation format.
    2. Confirm that the response includes a summary with MLA citation.
- **Expected Results**: The API provides a summary with a standard tone and an MLA-formatted citation.

Test Case 3: Standard Tone with Chicago Citation

- **Objective**: Confirm that the API can generate a summary with Chicago citation for a webpage.

- **Preconditions**: API is active; the webpage URL is valid and contains content eligible for Chicago style citation.
- **Steps**:
    1. Make a POST request to the API with parameters set for standard tone, paragraph style, short length, and Chicago citation format.
    2. Check that the API response includes a summary with Chicago citation.
- **Expected Results**: The API returns a summary in standard tone with a correctly formatted Chicago citation.

Test Case 4: Standard Tone without Citations

- **Objective**: Confirm that the API can generate a summary with no citation for a webpage.
- **Preconditions**: API is active; the webpage URL is valid and no citation parameter is selected.
- **Steps**:
    1. Make a POST request to the API with parameters set for standard tone, paragraph style, short length, and no citation format.
    2. Check that the API response includes a summary without any citation.
- **Expected Results**: The API returns a summary in standard tone with no citations.

Negative Conditions:

Test Case 5: Invalid Webpage URL

- **Objective**: Test the API's error response when provided with an invalid webpage URL.
- **Preconditions**: API is reachable.
- **Steps**:
    1. Submit a POST request to the API with an invalid webpage URL and other valid parameters.
    2. Examine the API's error response.
- **Expected Results**: The API returns an error indicating that the webpage URL is invalid.

Web Page Summarization Test Results

Test Name: Invalid URL Test

```
Request: {
        "key": "frontend",
        "input": "https://thisisdefinitelynotavalidurl.com",
        "type": 1,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "short"
}
Response: {
        "error": "error processing url"
}
```

Test Name: Valid URL with No Citation Citation

```
Request: {
        "key": "frontend",
        "input": "https://en.wikipedia.org/wiki/Main_Page",
        "type": 1,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "short",
        "citation": "No Citation"
}
Response: {
        "summary": "Wikipedia is a free encyclopedia that allows anyone to edit its content.
The article provides information about Blair Peach, a New Zealand teacher and activist who
died in 1979 during an anti-racism demonstration in London. The investigation revealed that
Peach was fatally hit by a police officer, resulting in compensation for his family and the
establishment of an award in his honor. The article also includes snippets of other featured
content on Wikipedia, such as information about recent news, historical events, and notable
individuals. Additionally, it offers links to different sections of Wikipedia and its sister
projects. The platform is maintained by volunteer editors and hosted by the Wikimedia
Foundation, providing access to a wide range of articles in multiple languages."
}
```

Test Name: Valid URL with APA Citation

```
Request: {
        "key": "frontend",
        "input": "https://en.wikipedia.org/wiki/Main_Page",
        "type": 1,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "short",
        "citation": "APA"
}
Response: {
        "summary": "Blair Peach, a New Zealand teacher and activist, tragically died during
an anti-racism demonstration in Southall, London, in 1979. An investigation revealed that he
was fatally struck by a police officer, leading to compensation for his family and the
establishment of awards and a school in his honor. The full report on his death was not
```

released to the public until 2009. (\"Blair Peach,\" n.d.)\n\nReference:\nBlair Peach. (n.d.). In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Main_Page"
}

Test Name: Valid URL with MLA Citation

```
Request: {
        "key": "frontend",
        "input": "https://en.wikipedia.org/wiki/Main_Page",
        "type": 1,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "short",
        "citation": "MLA"
}
Response: {
        "summary": "Blair Peach, a New Zealand teacher and activist, tragically died during
an anti-racism demonstration in Southall, London, in 1979. An investigation revealed he was
fatally hit on the head by a Metropolitan Police officer, leading to compensations and honors
in Peach's name. The full report on his death was not made public until many years later,
highlighting the aftermath and impact of the incident. (\"Blair Peach\")
\n\nCitation\n\"Wikipedia.\" Wikipedia, Wikimedia Foundation, 7 Mar. 2024,
https://en.wikipedia.org/wiki/Main_Page."
}
```

Test Name: Valid URL with Chicago Citation

```
Request: {
        "key": "frontend",
        "input": "https://en.wikipedia.org/wiki/Main_Page",
        "type": 1,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "short",
        "citation": "Chicago"
}
Response: {
        "summary": "Wikipedia is a widely accessible online encyclopedia where users can
edit and contribute to articles across a variety of topics. One notable case featured on the
site is that of Blair Peach, a New Zealand teacher and activist who tragically died during an
anti-racism demonstration in London in 1979. An investigation revealed he was fatally hit on
the head by a police officer. Despite initial obstruction and a delayed release of the full
investigation report, Peach's family received compensation and various honors were
bestowed upon him, including an award by the National Union of Teachers and a school
named after him in Southall.^1\n\nBibliography\nhttps://en.wikipedia.org/wiki/Main_Page"
}
```

7.3.4. YouTube Video Summarization Test Cases

Positive Conditions:

Test Case 1: Full Video Summarization

- **Objective**: Verify that the API can summarize the full content of a YouTube video.
- **Preconditions**: API is online; a valid YouTube video URL is provided.
- **Steps**:
  1. Send a POST request to the API with the parameters for a full video summarization.
  2. Review the response to ensure it contains a summary of the full video content.
- **Expected Results**: The API returns a successful response with a short paragraph summarizing the full video.

Test Case 2: Valid Timestamp Summarization

- **Objective**: Confirm that the API can summarize a specific segment of a YouTube video using valid timestamps.
- **Preconditions**: API is operational; a valid YouTube video URL and valid timestamps are supplied.
- **Steps**:
  1. Submit a POST request with valid timestamps for a specific video segment.
  2. Validate that the response includes a summary of the specified segment.
- **Expected Results**: The API delivers a summary corresponding to the segment of the video defined by the valid timestamps.

Negative Conditions:

Test Case 3: Invalid YouTube URL

- **Objective**: Test the API's response to an invalid YouTube URL for summarization.
- **Preconditions**: API is reachable.
- **Steps**:
  1. Make a POST request with an invalid YouTube URL for summarization.
  2. Assess the API's error response.
- **Expected Results**: The API provides an error message indicating that the YouTube URL is invalid.

Test Case 4: Timestamps Exceeding Video Duration

- **Objective**: Ensure the API correctly handles timestamps that exceed the actual video duration.
- **Preconditions**: API is functional; a valid YouTube video URL is used with timestamps that exceed the video's length.
- **Steps**:
    1. Dispatch a POST request with start and end times that surpass the video duration.
    2. Examine the API's response for proper error handling.
- **Expected Results**: The API returns an error stating that the start or end time exceeds the video's duration.

YouTube Video Summarization Test Results

Test Name: Full Video Test

```
Request: {
        "key": "frontend",
        "input": "https://www.youtube.com/watch?v=UNP03fDSj1U",
        "type": 2,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "short",
        "option": "Full Video"
}
Response: {
        "summary": "The speaker reflects on their experience of trying something new for 30
days, inspired by Morgan Spurlock's concept. By taking on various challenges, such as daily
photo-taking and writing a novel within a month, they discovered increased self-confidence
and a sense of accomplishment. The speaker highlights the importance of making
sustainable changes and shares that even small efforts can lead to significant results.
Encouraging the audience to consider what they have always wanted to try and commit to it
for the next 30 days, the speaker emphasizes the potential for personal growth within this
timeframe."
}
```

Test Name: Invalid URL Test

```
Request: {
        "key": "frontend",
        "input": "notayoutubelink",
        "type": 2,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "short",
        "option": "Full Video"
}
Response: {
        "error": "Invalid YouTube URL"
}
```

Test Name: Exceed Video Duration Test

```
Request: {
        "key": "frontend",
        "input": "https://www.youtube.com/watch?v=UNP03fDSj1U",
        "type": 2,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "short",
        "option": "Timestamp",
        "startTime": "00:00",
        "endTime": "00:05"
}
Response: {
        "error": "Start or end time exceeds video duration."
}
```

Test Name: Valid Timestamps Test

```
Request: {
        "key": "frontend",
        "input": "https://www.youtube.com/watch?v=UNP03fDSj1U",
        "type": 2,
        "tone": "Standard",
        "style": "Paragraph",
        "length": "short",
        "option": "Timestamp",
        "startTime": "00:01",
        "endTime": "00:03"
}
```

Response: {

"summary": "The YouTube video transcript highlights the transformative power of engaging in 30-day challenges. By pushing oneself to complete various challenges such as writing a novel in a month or hiking up Mount Kilimanjaro, the speaker experienced an increase in self-confidence and a shift towards a more adventurous lifestyle. They emphasize the importance of making small, sustainable changes to achieve lasting results. The key takeaway is that committing to short-term challenges can lead to significant personal growth and accomplishments."

```
}
```

7.4. URL Shortener

The URL Shortener's backend implementation remains unchanged from the previous progress report. The URL shortener works in the following manner. Firstly, a user inputs a URL to be shortened. The frontend Javascript looks for URLs that are prefixed with either http, https or www. - if this is not of the correct format, then the shortener outputs an error telling the user to input a valid link. When they click shorten, a POST request is sent to the Flask backend. The backend will receive the link, and create a 6 character hash based on the short link and the time it was received. If the user is a pro user, they have the option of giving a custom word, which would be appended to the end of the link if the custom word has not been taken. To allow for multiple users to use the same custom word, the link contains the username; a unique identifier, before the custom word. This shortened link is then added to the database table called shortened_url, which contains the original URL, the shortened URL and also the click counts representing the number of times the shortened URL has been clicked and resolved. This shortened URL redirects to the original URL through a Flask route which waits for the </s/shortURL> route, which finds the original URL from the database and returns the original URL, finally being redirected by the Flask server when the user resolves this link. When this method/route is called, the URL shortener increments the 'click_counts' row corresponding to the link by one. Links would take the form of https://4p02shortify.com/s/f475cc for example. The URL shortener also contains API routing and handling, which provides the same functionality, except it expects an API key if the user were to attempt to use the API.

7.4.1. URL Shortener Test Documents

The URL Shortener unit tests are as follows, which are run using Pytest, found in testShortener.py

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.

Test Cases

These test cases were investigated during development.

Test Case 1: Valid Link Shortening

- **Objective**: Ensure that the URL shortener can shorten a link.
- **Preconditions**: Website is running, the user enters a valid link.
- **Steps**:
    1. Enter a valid link into the text box for URL shortening.
    2. Click ‘Shorten’.
- **Expected Results**: A shortened link is pasted into the respective text box.

Test Case 2: Short Link Resolving

- **Objective**: Ensure that the shortened link resolves and redirects to the original link.
- **Preconditions**: Flask server is running, the user enters a valid short link.
- **Steps**:
    1. Enter a short link into a browser.
    2. Click enter.
- **Expected Results**: The shortened link redirects and loads the original link.

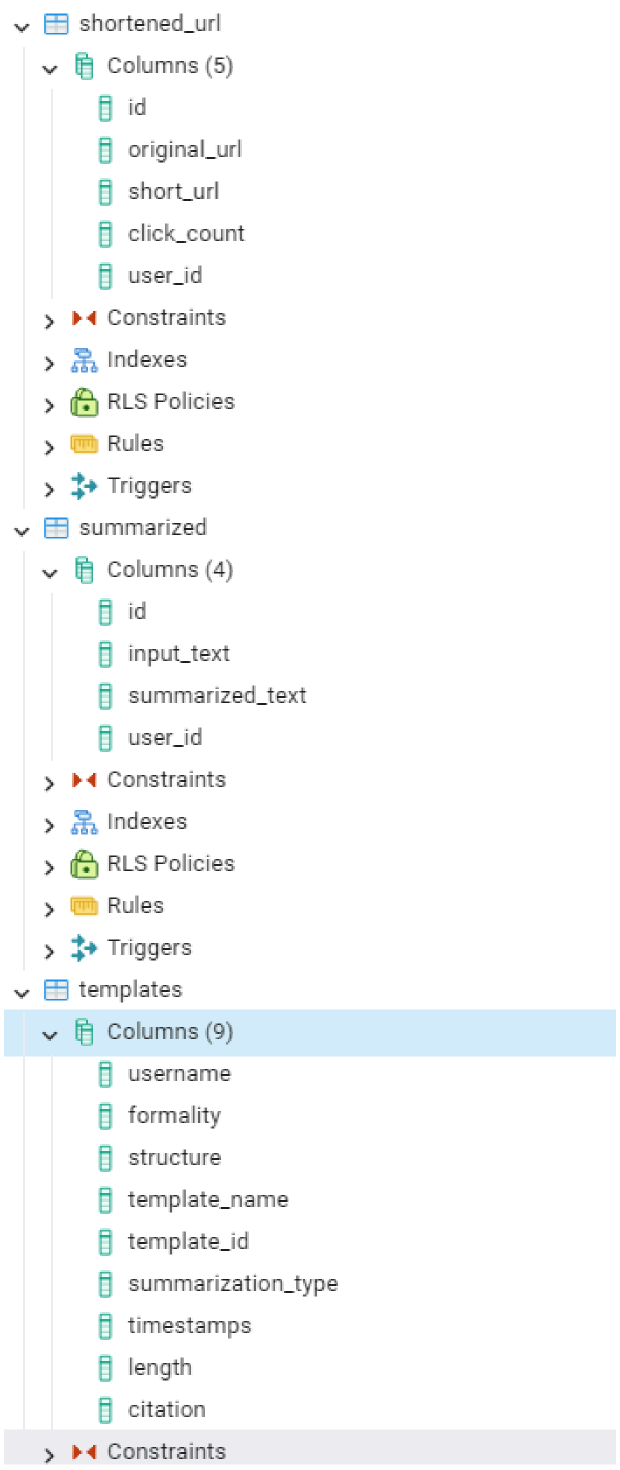
Test Case 3: Invalid Link Shortening

- **Objective**: Ensure that when a user inputs an invalid link, an error appears and no short link is generated.
- **Preconditions**: Flask server is running, the user enters an invalid link.
- **Steps**:
    1. Enter an invalid link into the text box for URL shortening.
    2. Click ‘Shorten’
- **Expected Results**: An error appears, asking the user to enter a valid link.

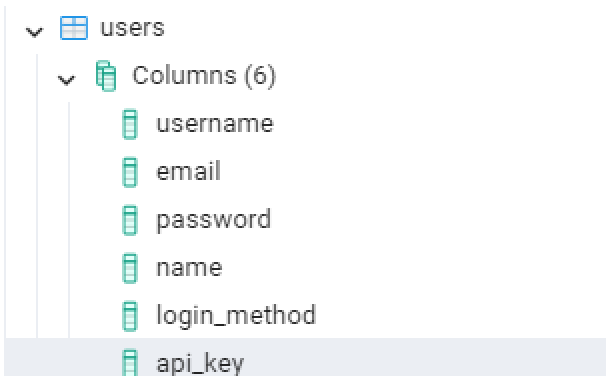
7.5. PostgreSQL Database

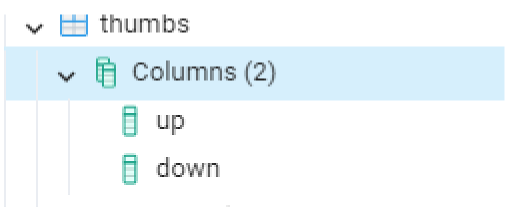
The Postgre Database has evolved since the first Progress Report. Firstly, the database was migrated from a locally hosted Postgre database on Amani’s local machine to a cloud-based database. What this means is, that the functionality that relies on the database can be accessed without relying on a local machine’s internet access or having the database installed and configured on each user’s machine. The cloud solution chosen is https://www.elephantsql.com/ - which offers a free cloud database with 20MB. Along with this storage restriction, comes a limit on the number of connections allowed, being 5. This turned out to be an issue when two people are testing at the same time, so we opted to go for their $5 a month plan to allow 100MB of storage and 10 connections. There are 6 tables in the database; shortened_url - containing all URL shortener logic and information, users - containing all user information, summarized - containing the summarization information, templates - containing the custom summarization levels that a user may save, feedback - where anyone who accesses our website can submit anonymous feedback and thumbs -

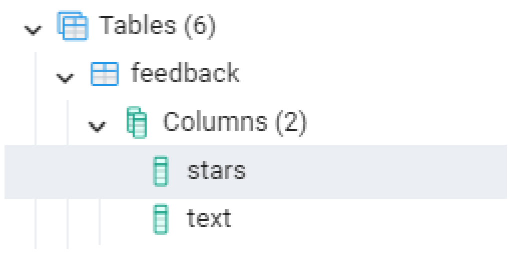
where a user can say either thumbs up or down after summarizing. Below is a screenshot of the database from pgAdmin. It shows the table names and their columns, which are self explanatory.



```
shortened_url
  Columns (5)
    id
    original_url
    short_url
    click_count
    user_id
  Constraints
  Indexes
  RLS Policies
  Rules
  Triggers
summarized
  Columns (4)
    id
    input_text
    summarized_text
    user_id
  Constraints
  Indexes
  RLS Policies
  Rules
  Triggers
templates
  Columns (9)
    username
    formality
    structure
    template_name
    template_id
    summarization_type
    timestamps
    length
    citation
  Constraints
```







This is the final state of our database, as more columns were added and dropped throughout development.

7.6. User History

The history management functionality of our web application provides users with the ability to save, retrieve, and delete their summarized texts and shortened URLs efficiently. This functionality enhances user experience by allowing easy access to past activities and managing these entries effectively within the system's database.

The backend, built with Python and Flask, interfaces with a PostgreSQL database using psycopg2, a PostgreSQL adapter that facilitates database operations securely. Connection to the database is established using credentials stored in environment variables (.env file). The connection string includes parameters for database name, user credentials, host details, and sets `sslmode='require'` to enforce encrypted connections, safeguarding data integrity during transmission. Lastly, to prevent SQL injection, we used parameterized SQL query to perform actions on our database

The user history functionality is only applicable if the user is logged into the web application. For summarization, users are given an option to save a summary, if the user clicks the "Save Summary" button, the Python insert function is called where insert query is executed with required fields. For URL shorteners, the fields (given link and shortened link) are automatically saved if the user is logged in. When a user saves a summary, the `insertHistory` method is triggered. This method first checks the count of existing summaries for the user in the summarized table. If the user has ten entries, which is the limit set to prevent excessive use of storage, the oldest entry is removed. This is achieved by a subquery that identifies the oldest entry (lowest ID) for deletion:

```
DELETE FROM summarized
WHERE id = (
    SELECT id FROM summarized
    WHERE user_id = %s
    ORDER BY id ASC
    LIMIT 1
)
```

Users can retrieve their history through the `retrieveHistory` method, which fetches all entries from the summarized table for the user. This method is called when a user clicks on the relevant history components button on their dashboard (Summarizer or Shortened URL). Deletion of a specific entry is handled by the `deleteHistory` method, where the entry ID is validated and converted to an integer before executing the delete command:

```
DELETE FROM summarized
WHERE id = %s AND user_id = %s
```

This method only performs the deletion if the specified ID belongs to the user, adding more enhancement to authorization.

Similar to text summaries, shortened URLs are automatically saved when created by logged-in users. The `retrieveURLHistory` method provides users with their history of shortened URLs, displaying details such as the original URL, the shortened form, and click counts. Deletion of URLs is handled with the same security considerations as text summaries, ensuring that operations are executed only if the user ID matches and validating the URL ID.

```
DELETE FROM shortened_url
WHERE id = %s AND user_id = %s
```

Each operation commits changes to the database and handles exceptions by rolling back in case of errors, making sure the database state remains consistent and accurate.

All these are implemented through strong history management functionalities in the application that not only improves interaction for the users by giving them an ability to trace their activities but also ensures the safety of the data and efficient use of resources. For this, proper parameterization of SQL commands, secure connections with the database, and attentive control over the information of the user are implemented.

7.6.1. User History Test Documents

Positive Test Cases

Test Case 1: Insert Valid Summary

- **Objective:** Verify that a valid summary can be inserted correctly.
- **Precondition**: User is logged in with fewer than 5 summaries stored.
- **Test Steps:**
  Login to the system.
  Submit a request to insert a summary with valid text.
- **Expected Result:** The summary should be added to the database; the system should confirm successful insertion.
- **Test Data:**
  Username: user1
  Content: “this could be either URL or a text”
  Summary: "This is a test summary of the input text."

Test Case 2: Retrieve Summaries

- **Objective:** Verify that all summaries for a user can be successfully retrieved.
- **Precondition:** User is logged in and has at least one summary stored.
- **Test Steps:**
  Login to the system.
  Submit a request to retrieve all summaries for the user.
- **Expected Result:** All stored summaries for the user are displayed.
- **Test Data:**
  Id of entry: Integer
  Username: user1

Test Case 3: Delete Specific Summary

- **Objective:** Verify that a user can delete a specific stored summary.
- **Precondition:** User is logged in and has summaries stored with known IDs.
- **Test Steps:**
  Login to the system.
  Submit a request to delete a specific summary using its ID.
- **Expected Result:** The summary is deleted; the system confirms deletion success.
- **Test Data:**
  Username: user1
  Summary ID: 123

Negative Test Cases

Test Case 4: Insert Summary Without Login

- **Objective:** Ensure that summaries cannot be inserted without user authentication.
- **Precondition:** User is not logged in/Username is not provided.
- **Test Steps:**
  Attempt to insert a summary without logging in/providing username.
- **Expected Result:** The system rejects the request and displays an error message about authentication failure.
- **Test Data:**
  Summary: "This is a test summary."

Test Case 5: Delete Non-existent Summary

- **Objective:** Verify that the system handles attempts to delete a non-existent summary gracefully.
- **Precondition:** User is logged in.
- **Test Steps:**
  Login to the system.
  Attempt to delete a summary with an ID that does not exist.
- **Expected Result:** The system should respond that no matching summary was found and deletion was unsuccessful.
- **Test Data:**
  Username: user1
  Summary ID: 999

Test Case 6: Exceed Summary Storage Limit Without Deletion

- **Objective:** To test if the system improperly allows more than the maximum number of summaries without deleting the oldest entry.
- **Precondition:** User is logged in and has the maximum allowed number of summaries (e.g., 10) already stored.
- **Test Steps:**
  Log into the system as the user.
  Attempt to insert a new summary beyond the storage limit.
  Check if the new summary is added without deleting the oldest summary.
- **Expected Result:** The system should either reject the new entry or delete the oldest summary to make space for the new one.
- **Test Data:**
  Username: user1
  Summary: "New summary to test the system's handling of storage limits."

Test Case 7: Retrieve Summaries With Invalid User

- **Objective:** To ensure that summaries cannot be retrieved using an invalid or non-existent username.
- **Precondition:** User attempts to access data with a username that does not exist in the database.
- **Test Steps:**
  Use an invalid username to request a retrieval of summaries.
  Observe the system's response.
- **Expected Result:** The system should return an error message indicating that the username is invalid or does not exist.
- **Test Data:**
  Username: nonExistentUser

Boundary Test Cases

Test Case 8: Maximum Length Summary Input

- **Objective:** Test the system's handling of the longest possible summary input.
- **Precondition:** User is logged in.
- **Test Steps:**
  Login to the system.
  Submit a summary that is at the maximum acceptable length.

Expected Result: The system accepts the summary and stores it correctly.

- **Test Data:**
  Username: user1
  Summary: [Insert maximum length text]

Test Case 9: Boundary of Deletion (FIFO Test)

- **Objective:** Verify that the system deletes the oldest summary when a new summary is added beyond the storage limit.
- **Precondition:** User is logged in and has exactly 5 summaries stored.
- **Test Steps:**
  Login to the system.
  Insert a new summary, triggering the FIFO deletion mechanism.
- **Expected Result:** The oldest summary is deleted; the new summary is added, and the total does not exceed 5.
- **Test Data:**
  Username: user1
  New Summary: "This is a new summary to test FIFO."

7.7. Account Creation

Account creation is a core part of this project. Our pro features are essentially simulated by having a user create an account. If you create an account, you are a pro-user and you have access to all of the features. There are three ways a user can create an account; manually, using an email address, with their password and a selected nickname, or using Facebook/Google authentication. This section will focus on manual creation. When a user attempts to create an account, they are prompted to enter an email address, a name and a password. There are several cases to consider, all of which are handled. For instance, we integrated a free email-verification API (https://api.hunter.io/v2/email-verifier) that will check to see if an email is valid when a user attempts to register with it. If a user enters an email that's already registered, an error will appear, if a user enters an invalid password, given that it doesn't meet the requirements (8-20 chars, 1 uppercase, 1 number), an error will appear. If the password and re-entered password don't match, an error will appear, etc. When a user enters the correct information, a fetch is made to our Flask backend running on a localhost port 5001 (if running locally, otherwise it will fetch to the server on port 5001). Here, the information is processed and fed to the database, with JSON return statements that allow for error handling and routing. Currently, the user does not enter a 'username', but rather, a 'name', which is not necessarily unique to the user. The user's only unique identifier (that is known to them) is their email. In the backend, each email that is entered is given a unique 6-character hash as a username, which allows for more fluid backend processing. This serves as the backend's unique identifier for most cases. When a user account is created, three rows are created in the templates table in the database, corresponding to that user account. This is because each user is given three custom templates - corresponding to three sets of permutations of summarization levels that they can save into a given template. When an account is deleted, these rows are naturally deleted as well.

7.8. Login with Google and Facebook

7.8.1. Google

Logging in with Google makes use of the Google Cloud Console's OAuth API. The API was integrated into our front end through its provided functions to create a button and allow user sign-ins. In the front end, we make use of the renderButton function that the API provides, which upon clicking, allows a user to select which account they want to login with. This sends a callback response, which contains an encoded JWT, from which the email and name is extracted. This information is then recorded in our database, and the user is then logged in by caching their email and name in local storage until they log out. The Google

OAuth app is currently in testing mode, however it seems that anyone can log in, even on our hosted website.

7.8.2. Facebook

Facebook authentication works similarly to Google authentication, but with a few differences. Like Google authentication, only the user's email and publicly accessible name is taken. We make use of two React libraries; reactjs-social-login and react-social-login-buttons to make Facebook authentication work. The app was created on Facebook's developer console, and given the app ID, when the 'Login with Facebook' button is clicked, it asks the user to login to their Facebook account and asks if they want to allow the login, and just as with Google, the email and name is stored to simulate logging in. This is how it works in theory. The main challenge with Facebook authentication is that we could not publish the app, because it requires business verification, and we are not a business. It will also only allow the developer/owner of the app to log in when in testing mode, and you cannot add other emails. This made it much less developer friendly than Google's, however it would work if we were able to publish.

7.9. User Management

Associated with account creation, we allow users to make several changes to their accounts. They can delete their account, change their name, change their email address, change their password, or reset their password if they forgot it. Users of the platform can also save templates; custom summarization levels, three for each user, they can save and access their summarization (10 summaries max per user) and shortening history, as well as gain access to the API key and documentation for the system's API. As such, our pro-users are represented by signing up. All of these cases have been implemented and tested. Each capability is represented in the front end, and when the user interacts with a given button, fetches are made to the Flask server, which is then routed to one of several defined functions that handle the logic behind the desired functionality. All user management functionality is handled by one Python file, which runs on a dedicated Flask server on a given port (currently 5001). These functions that are routed by Flask, all rely on the Authentication class within the Python file (authentication.py). There, SQL queries are defined and executed, which is where the database interaction takes place. Naturally, once a result is returned, this is fed back in the opposite direction and handled by the front end.

7.9.1. User Management Test Documents

Unit Testing

Testing is imperative with a complex system such as this. Like the URL Shortening functionality, unit tests using Pytest were implemented to test the functionality of the user management class. The user management functionality is not complete, however, so the implemented tests are not final. Below are some of the tests that have been implemented so far:

1. Test that a user is registered
2. Test the ‘checkIfAlreadyRegisted’ method
3. Test the ‘findEmail’ method
4. Test that a user can ‘login’
5. Test the ‘isPasswordValid’ function
6. Test account deletion
7. Test password-changing functionality
8. Test password reset functionality
9. Test change email functionality
10. Test getName functionality
11. Test name-changing functionality
12. Test get username functionality
13. Test isPasswordCorrect functionality
14. Test template row creation
15. Test template addition
16. Test template clearance
17. Test template deletion
18. Test template fetching
19. Test the ‘check if a template is in use’ functionality
20. Test add feedback functionality
21. Test add summarization to history

Test Cases

Test Case 1: Manual Account Creation

- **Objective**: Ensure that a user can create an account with an email and password.
- **Preconditions**: Website is running, the user enters a valid email and password.
- **Steps**:
    1. Enter an email, password and confirm password.
    2. Click Sign up.
- **Expected Results**: The user is logged in, with access to the user dashboard and the user’s information is stored in the database.

Test Case 2: Invalid Email Account Creation

- **Objective**: Ensure that users cannot create an account with an invalid email address.
- **Preconditions**: Website is running, the user enters an invalid email.
- **Steps**:

    1. Enter an invalid email, and a valid password.
    2. Click Sign up.
- **Expected Results**: An error appears explaining to the user that their email is not valid.

Test Case 3: Passwords Don't Match - Cannot Login

- **Objective**: Ensure that users cannot log in if the passwords provided during sign-up do not match.
- **Preconditions**: User attempts to log in with mismatched passwords.
- **Steps**:
    1. Enter a valid email and password.
    2. Enter a different password in the confirm password field.
    3. Click Sign up.
- **Expected Results**: An error message appears indicating that the passwords provided do not match, and the user cannot log in.

Test Case 4: Invalid Passwords - Cannot Login

- **Objective**: Ensure that users cannot log in with invalid passwords.
- **Preconditions**: User attempts to log in with an invalid password format.
- **Steps**:
    1. Enter a valid email.
    2. Enter an invalid password format (e.g., less than 8 characters, no numbers)
    3. Click Sign up.
- **Expected Results**: An error message appears indicating that the password does not meet the required format, and the user cannot log in.

Test Case 5: Successful Email Address Change

- **Objective**: Ensure that users logged in manually can successfully change their email address.
- **Preconditions**: User is logged in and navigates to the account settings page.
- **Steps**:
    1. Navigate to the account settings page.
    2. Enter a new valid email address in the appropriate field.
    3. Click Change Email.

  Expected Results: The user's email address is successfully updated in the database.

Test Case 6: Successful Name Change

- **Objective**: Ensure that users can successfully change their name.
- **Preconditions**: User is logged in and navigates to the account settings page.
- **Steps**:

    1. Navigate to the account settings page.
    2. Enter a new name in the appropriate field.
    3. Click Change Name.
- **Expected Results**: The user's name is successfully updated in the database, and the page reloads to match their name.

Test Case 7: Successful Password Change

- **Objective**: Ensure that users can successfully change their password.
- **Preconditions**: User is logged in and navigates to the account settings page.
- **Steps**:
    1. Navigate to the account settings page.
    2. Enter the current password and a new valid password in the appropriate fields.
    3. Click Change Password.
- **Expected Results**: The user's password is successfully updated in the database

Test Case 8: Successful Password Reset

- **Objective**: Ensure that users can successfully reset their password.
- **Preconditions**: User enters a valid email in the Verification and reaches the Password Reset page after clicking ‘Forgot password’
- **Steps**:
    1. Enter a valid email address in the password reset form.
    2. Click Submit.
    3. Check the email for the password reset link and follow the instructions.
- **Expected Results**: The user receives a password reset link via email, and upon clicking it, they can successfully reset their password.

Test Case 9: Unsuccessful Password Reset (Invalid Email Address)

- **Objective**: Ensure that users cannot reset their password with an invalid email address.
- **Preconditions**: User attempts to reset their password with an invalid email address.
- **Steps**:
    1. Enter an invalid email address in the Verification form.
    2. Click Submit.
- **Expected Results**: An error message appears indicating that the provided email address is invalid.

Test Case 10: Cannot Change Email/Password of Google/FB Account

- **Objective**: Ensure that users cannot change the email/password of accounts authenticated via Google/Facebook.
- **Preconditions**: User attempts to change the email/password of a Google/Facebook authenticated account.

- **Steps**:
    1. Navigate to the account settings page.
    2. Attempt to change the email/password of the Google/Facebook authenticated account.
    3. Click Submit.
- **Expected Results**: An error message appears indicating that the email/password cannot be changed for Google/Facebook authenticated accounts.

Test Case 11: Trying to Save a Template in a Slot That's Already Taken

- **Objective**: Ensure that users are notified when they are about to save a template that is already saved.
- **Preconditions**: User attempts to save a template in a slot that is already occupied.
- **Steps**:
    1. Navigate to the Templates section in User Dashboard or on Summarizer
    2. Attempt to save a template in a slot that is already occupied.
    3. Click Save.
- **Expected Results**: An error message appears indicating that the selected slot is already taken, asking the user if they want to overwrite the template.

Test Case 12: Trying to Save More Than 10 Summaries

- **Objective**: Ensure that users cannot save more than 10 summaries.
- **Preconditions**: User attempts to save more than 10 summaries.
- **Steps**:
    1. Navigate to the summarizer
    2. Input a summary, summarize and click save.
    3. Repeat the process.
- **Expected Results**: An error message appears indicating that the maximum limit of 10 summaries has been reached, and no further summaries can be saved.

7.10. API Access

API access is another key functionality in our pro-version of our service. We provide API functionality for the shortening and summarization. Every user, upon creating their account, is given an API key; a hash generated based on their username and the timestamp associated with their account creation, which they can access in the API Access section of the application. For summarization, we provide the endpoint /api/summarize with instructions on how to perform the summarization functionality without accessing our code. This route, given the correct parameters, will allow users to summarize with just as much capability as our custom summarization levels. For the shortening, three functionalities are provided. Shortening, resolving, and click count retrieval. As with the summarizer, the URLs and instructions are given to the user. For example, http://4p02shortify.com:5002/apishorten will contain the functionality for the shortening feature, but will only work if given a valid API key.

8. Release Planning

Weekly meetings every Tuesdays from 3:00 PM to 4:30 PM and Fridays from 4:00 PM to 5:30 PM.

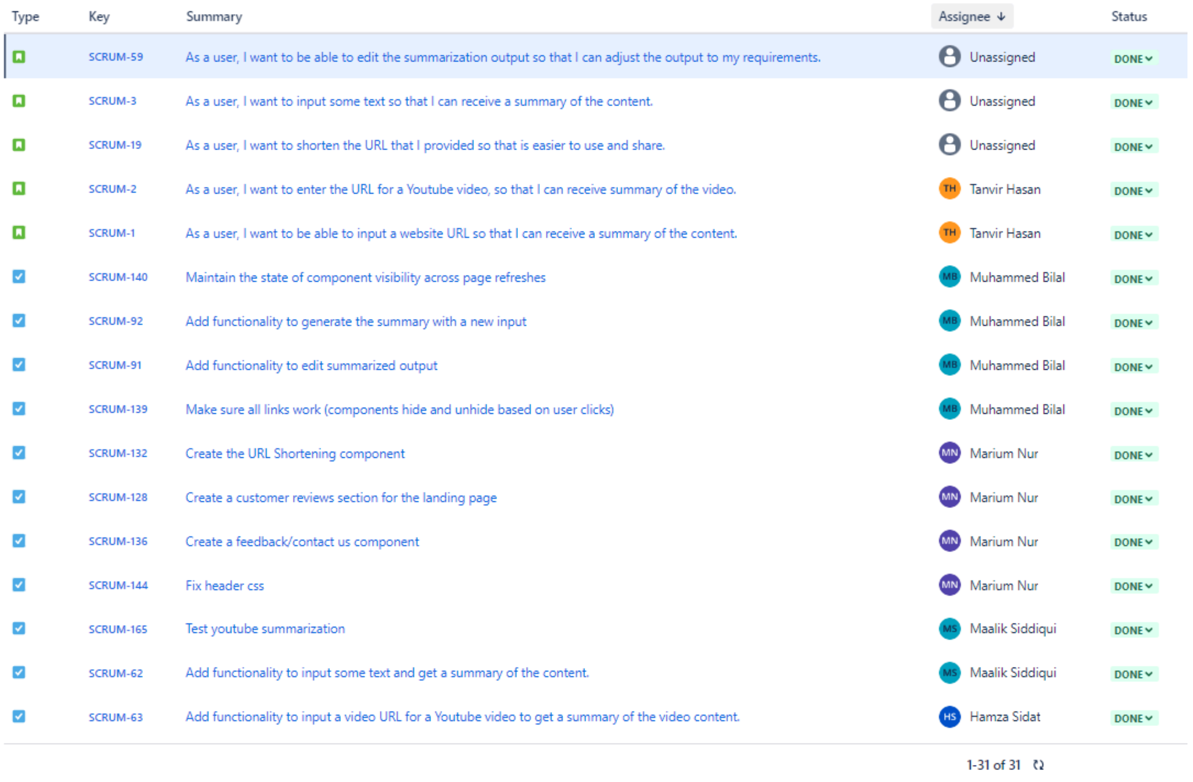
- *Daily scrum meetings to be simulated through communications on Discord (with the exception of in-person Scrums above)*

8.1. 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.

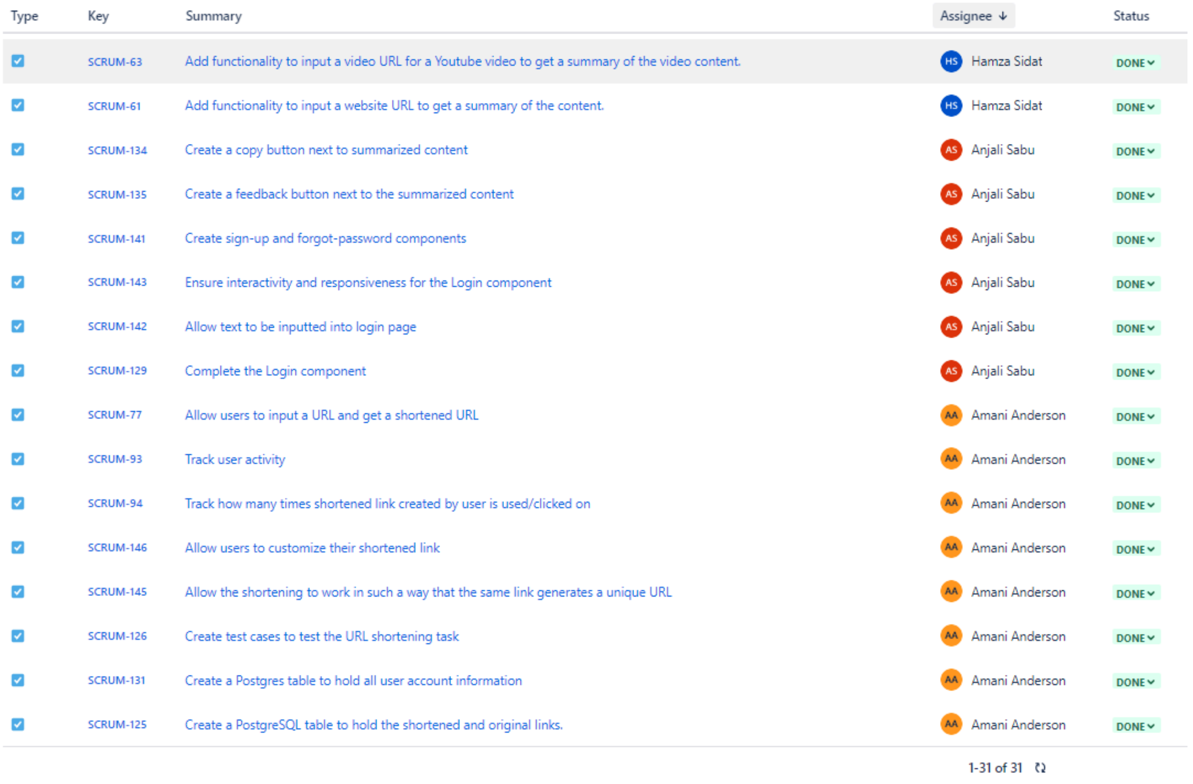
8.2. Sprint 2 (9 Feb - 23 Feb)

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.



Type	Key	Summary	Assignee ↓	Status
🟩	SCRUM-59	As a user, I want to be able to edit the summarization output so that I can adjust the output to my requirements.	Unassigned	DONE
🟩	SCRUM-3	As a user, I want to input some text so that I can receive a summary of the content.	Unassigned	DONE
🟩	SCRUM-19	As a user, I want to shorten the URL that I provided so that is easier to use and share.	Unassigned	DONE
🟩	SCRUM-2	As a user, I want to enter the URL for a Youtube video, so that I can receive summary of the video.	TH Tanvir Hasan	DONE
🟩	SCRUM-1	As a user, I want to be able to input a website URL so that I can receive a summary of the content.	TH Tanvir Hasan	DONE
☑	SCRUM-140	Maintain the state of component visibility across page refreshes	MB Muhammed Bilal	DONE
☑	SCRUM-92	Add functionality to generate the summary with a new input	MB Muhammed Bilal	DONE
☑	SCRUM-91	Add functionality to edit summarized output	MB Muhammed Bilal	DONE
☑	SCRUM-139	Make sure all links work (components hide and unhide based on user clicks)	MB Muhammed Bilal	DONE
☑	SCRUM-132	Create the URL Shortening component	MN Marium Nur	DONE
☑	SCRUM-128	Create a customer reviews section for the landing page	MN Marium Nur	DONE
☑	SCRUM-136	Create a feedback/contact us component	MN Marium Nur	DONE
☑	SCRUM-144	Fix header css	MN Marium Nur	DONE
☑	SCRUM-165	Test youtube summarization	MS Maalik Siddiqui	DONE
☑	SCRUM-62	Add functionality to input some text and get a summary of the content.	MS Maalik Siddiqui	DONE
☑	SCRUM-63	Add functionality to input a video URL for a Youtube video to get a summary of the video content.	HS Hamza Sidat	DONE

1-31 of 31

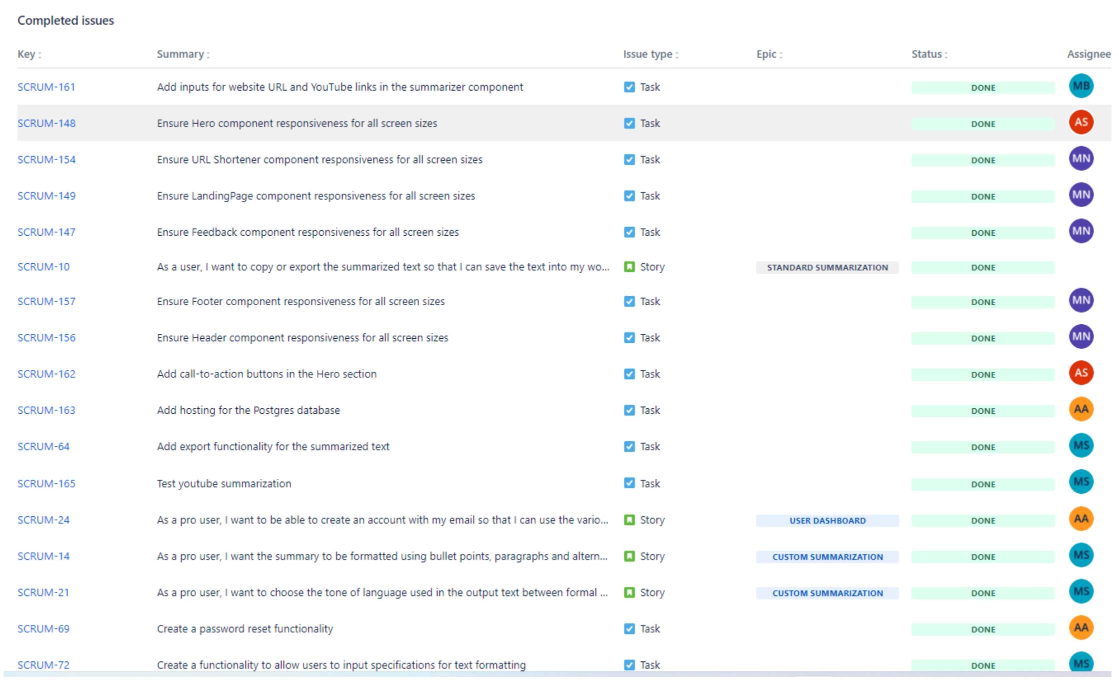


Type	Key	Summary	Assignee ↓	Status
☑	SCRUM-63	Add functionality to input a video URL for a Youtube video to get a summary of the video content.	HS Hamza Sidat	DONE
☑	SCRUM-61	Add functionality to input a website URL to get a summary of the content.	HS Hamza Sidat	DONE
☑	SCRUM-134	Create a copy button next to summarized content	AS Anjali Sabu	DONE
☑	SCRUM-135	Create a feedback button next to the summarized content	AS Anjali Sabu	DONE
☑	SCRUM-141	Create sign-up and forgot-password components	AS Anjali Sabu	DONE
☑	SCRUM-143	Ensure interactivity and responsiveness for the Login component	AS Anjali Sabu	DONE
☑	SCRUM-142	Allow text to be inputted into login page	AS Anjali Sabu	DONE
☑	SCRUM-129	Complete the Login component	AS Anjali Sabu	DONE
☑	SCRUM-77	Allow users to input a URL and get a shortened URL	AA Amani Anderson	DONE
☑	SCRUM-93	Track user activity	AA Amani Anderson	DONE
☑	SCRUM-94	Track how many times shortened link created by user is used/clicked on	AA Amani Anderson	DONE
☑	SCRUM-146	Allow users to customize their shortened link	AA Amani Anderson	DONE
☑	SCRUM-145	Allow the shortening to work in such a way that the same link generates a unique URL	AA Amani Anderson	DONE
☑	SCRUM-126	Create test cases to test the URL shortening task	AA Amani Anderson	DONE
☑	SCRUM-131	Create a Postgres table to hold all user account information	AA Amani Anderson	DONE
☑	SCRUM-125	Create a PostgreSQL table to hold the shortened and original links.	AA Amani Anderson	DONE

1-31 of 31

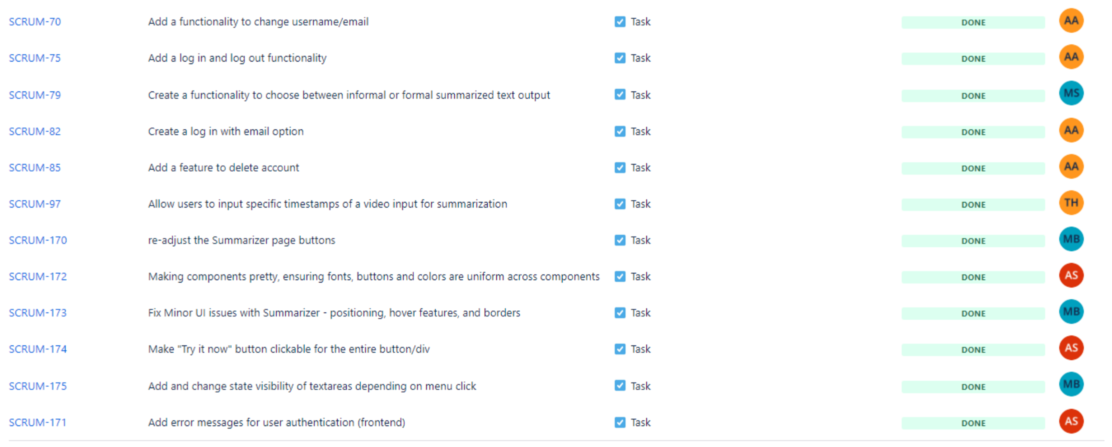
8.3. Sprint 3 (23 Feb - 8 Mar)

During Sprint 3, significant progress was made for various assigned tasks. We began with a productive meeting with the TA to review our progress and completed the first progress report, during which it was suggested to include upcoming features for future sprints in our report. Our focus on responsiveness paid off as most pages and components are now fully responsive across different screen sizes. The summarizer component saw notable enhancements, including custom features for premium users, additional input options such as text, website, and YouTube URLs, as well as hover effects and tooltips. Backend improvements were made to enable summarization with bullet points and other formatting options. Furthermore, users can now add timestamps for video summarization. Additionally, a URL shortener was successfully implemented and the branch was then merged with the User Interface branch. ElephantSQL is being used for the database, with numerous tables added to store history, summarizer inputs and outputs, URL Shortener inputs and output, and much more. The backend functionalities for history access, editing, and deleting are also working as of now alongside the addition of account management features like changing email and password, with enforced password requirements. Backend logic was refined to generate a hash-based username from user emails. Frontend error messages were implemented in the user account authentication process for better user guidance, supported by corresponding backend functionalities.



Completed issues

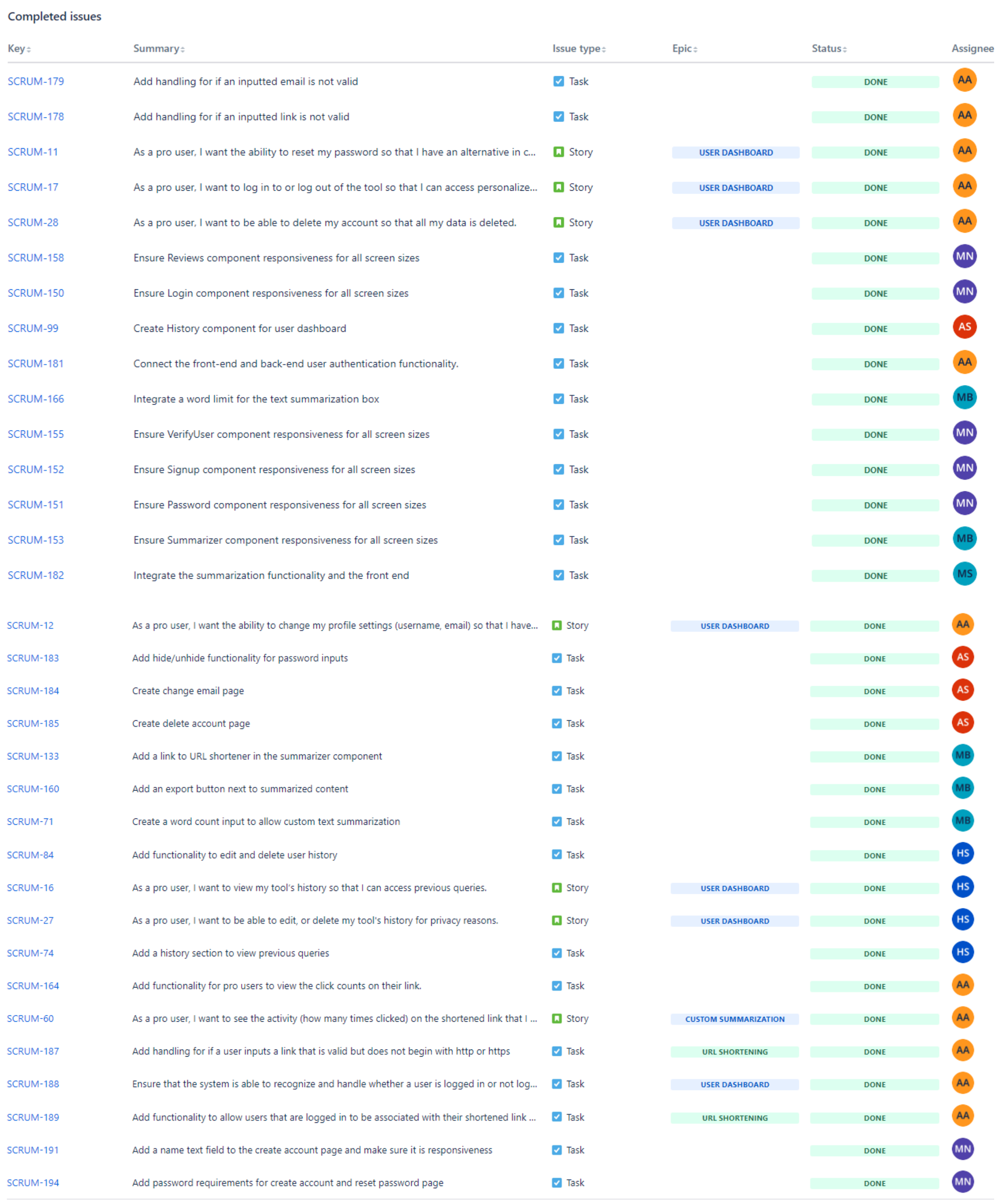
Key	Summary	Issue type	Epic	Status	Assignee
SCRUM-161	Add inputs for website URL and YouTube links in the summarizer component	Task		DONE	MB
SCRUM-148	Ensure Hero component responsiveness for all screen sizes	Task		DONE	AS
SCRUM-154	Ensure URL Shortener component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-149	Ensure LandingPage component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-147	Ensure Feedback component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-10	As a user, I want to copy or export the summarized text so that I can save the text into my wo...	Story	STANDARD SUMMARIZATION	DONE	
SCRUM-157	Ensure Footer component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-156	Ensure Header component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-162	Add call-to-action buttons in the Hero section	Task		DONE	AS
SCRUM-163	Add hosting for the Postgres database	Task		DONE	AA
SCRUM-64	Add export functionality for the summarized text	Task		DONE	MS
SCRUM-165	Test youtube summarization	Task		DONE	MS
SCRUM-24	As a pro user, I want to be able to create an account with my email so that I can use the vario...	Story	USER DASHBOARD	DONE	AA
SCRUM-14	As a pro user, I want the summary to be formatted using bullet points, paragraphs and altern...	Story	CUSTOM SUMMARIZATION	DONE	MS
SCRUM-21	As a pro user, I want to choose the tone of language used in the output text between formal ...	Story	CUSTOM SUMMARIZATION	DONE	MS
SCRUM-69	Create a password reset functionality	Task		DONE	AA
SCRUM-72	Create a functionality to allow users to input specifications for text formatting	Task		DONE	MS



SCRUM-70	Add a functionality to change username/email	Task	DONE	AA
SCRUM-75	Add a log in and log out functionality	Task	DONE	AA
SCRUM-79	Create a functionality to choose between informal or formal summarized text output	Task	DONE	MS
SCRUM-82	Create a log in with email option	Task	DONE	AA
SCRUM-85	Add a feature to delete account	Task	DONE	AA
SCRUM-97	Allow users to input specific timestamps of a video input for summarization	Task	DONE	TH
SCRUM-170	re-adjust the Summarizer page buttons	Task	DONE	MB
SCRUM-172	Making components pretty, ensuring fonts, buttons and colors are uniform across components	Task	DONE	AS
SCRUM-173	Fix Minor UI issues with Summarizer - positioning, hover features, and borders	Task	DONE	MB
SCRUM-174	Make "Try it now" button clickable for the entire button/div	Task	DONE	AS
SCRUM-175	Add and change state visibility of textareas depending on menu click	Task	DONE	MB
SCRUM-171	Add error messages for user authentication (frontend)	Task	DONE	AS

8.4. Sprint 4 (8 Mar - 22 Mar)

During Sprint 4, the frontend development began with the user dashboard design (to allow users to access profile settings, history, templates for summarization and API access). A word count feature was implemented in the summarizer component, restricting usage for texts below 125 words unless users opt for premium features with the creation of an account. Backend enhancements for YouTube video summarization included audio extraction using Amazon services. The User Interface branch was merged with the main branch, solidifying backend-frontend integration. While some error messages and user dashboard functionalities are still under development, functionalities for storing usernames in the database and identifying logged-in users for premium feature access have been successfully added. Test cases for account authentication were also incorporated.

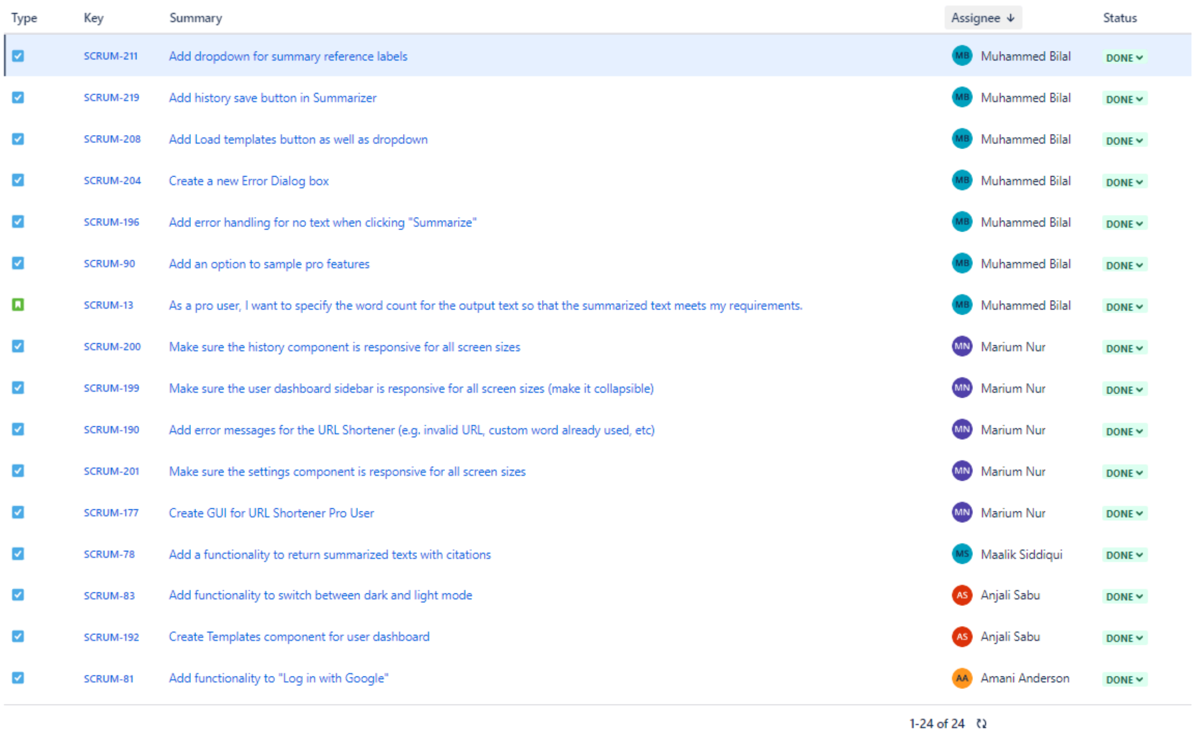


Completed issues

Key	Summary	Issue type	Epic	Status	Assignee
SCRUM-179	Add handling for if an inputted email is not valid	Task		DONE	AA
SCRUM-178	Add handling for if an inputted link is not valid	Task		DONE	AA
SCRUM-11	As a pro user, I want the ability to reset my password so that I have an alternative in c...	Story	USER DASHBOARD	DONE	AA
SCRUM-17	As a pro user, I want to log in to or log out of the tool so that I can access personalize...	Story	USER DASHBOARD	DONE	AA
SCRUM-28	As a pro user, I want to be able to delete my account so that all my data is deleted.	Story	USER DASHBOARD	DONE	AA
SCRUM-158	Ensure Reviews component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-150	Ensure Login component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-99	Create History component for user dashboard	Task		DONE	AS
SCRUM-181	Connect the front-end and back-end user authentication functionality.	Task		DONE	AA
SCRUM-166	Integrate a word limit for the text summarization box	Task		DONE	MB
SCRUM-155	Ensure VerifyUser component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-152	Ensure Signup component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-151	Ensure Password component responsiveness for all screen sizes	Task		DONE	MN
SCRUM-153	Ensure Summarizer component responsiveness for all screen sizes	Task		DONE	MB
SCRUM-182	Integrate the summarization functionality and the front end	Task		DONE	MS
SCRUM-12	As a pro user, I want the ability to change my profile settings (username, email) so that I have...	Story	USER DASHBOARD	DONE	AA
SCRUM-183	Add hide/unhide functionality for password inputs	Task		DONE	AS
SCRUM-184	Create change email page	Task		DONE	AS
SCRUM-185	Create delete account page	Task		DONE	AS
SCRUM-133	Add a link to URL shortener in the summarizer component	Task		DONE	MB
SCRUM-160	Add an export button next to summarized content	Task		DONE	MB
SCRUM-71	Create a word count input to allow custom text summarization	Task		DONE	MB
SCRUM-84	Add functionality to edit and delete user history	Task		DONE	HS
SCRUM-16	As a pro user, I want to view my tool's history so that I can access previous queries.	Story	USER DASHBOARD	DONE	HS
SCRUM-27	As a pro user, I want to be able to edit, or delete my tool's history for privacy reasons.	Story	USER DASHBOARD	DONE	HS
SCRUM-74	Add a history section to view previous queries	Task		DONE	HS
SCRUM-164	Add functionality for pro users to view the click counts on their link.	Task		DONE	AA
SCRUM-60	As a pro user, I want to see the activity (how many times clicked) on the shortened link that I ...	Story	CUSTOM SUMMARIZATION	DONE	AA
SCRUM-187	Add handling for if a user inputs a link that is valid but does not begin with http or https	Task	URL SHORTENING	DONE	AA
SCRUM-188	Ensure that the system is able to recognize and handle whether a user is logged in or not log...	Task	USER DASHBOARD	DONE	AA
SCRUM-189	Add functionality to allow users that are logged in to be associated with their shortened link ...	Task	URL SHORTENING	DONE	AA
SCRUM-191	Add a name text field to the create account page and make sure it is responsiveness	Task		DONE	MN
SCRUM-194	Add password requirements for create account and reset password page	Task		DONE	MN

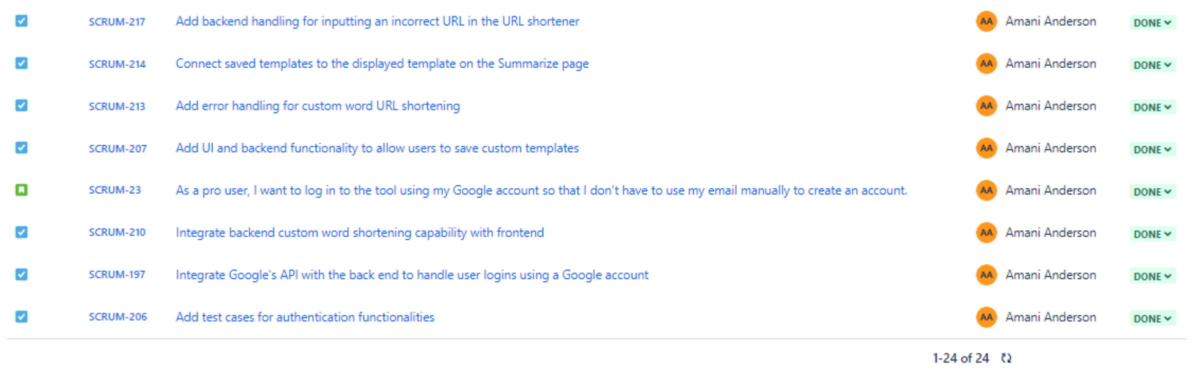
8.5. Sprint 5 (22 Mar - 5 Apr)

In the fifth sprint, we completed the implementation of Google login and strengthened backend and database functionality to prevent password changes via the dashboard for users logging in with Google. Comprehensive test cases were added for YouTube transcription, authentication, and summarization processes along with error handling for URL shortening with a custom word. The frontend of the URL shortener for premium users now includes an extra text field for custom words and the user dashboard's frontend had an addition of a collapsible sidebar to accommodate for smaller screen sizes. The dark mode functionality was implemented for all pages throughout the website. On the backend, the premium URL shortener now incorporates the custom word in the shortened URL, ensuring premium users are recognized within Shorify accounts. We introduced a templates feature for custom summarizations, enabling users to save, edit, or load templates, which were saved in the user dashboard (stored in the database). Additionally, both frontend and backend now include citations for web summarizations, available to premium users. The API access documentation was also written for the summarizer and url shortener. The history backend was also integrated with the frontend to allow users to save selected summaries in the user dashboard history and the last five URL's that were shortened.



Type	Key	Summary	Assignee ↓	Status
☑	SCRUM-211	Add dropdown for summary reference labels	MB Muhammed Bilal	DONE
☑	SCRUM-219	Add history save button in Summarizer	MB Muhammed Bilal	DONE
☑	SCRUM-208	Add Load templates button as well as dropdown	MB Muhammed Bilal	DONE
☑	SCRUM-204	Create a new Error Dialog box	MB Muhammed Bilal	DONE
☑	SCRUM-196	Add error handling for no text when clicking "Summarize"	MB Muhammed Bilal	DONE
☑	SCRUM-90	Add an option to sample pro features	MB Muhammed Bilal	DONE
🟩	SCRUM-13	As a pro user, I want to specify the word count for the output text so that the summarized text meets my requirements.	MB Muhammed Bilal	DONE
☑	SCRUM-200	Make sure the history component is responsive for all screen sizes	MN Marium Nur	DONE
☑	SCRUM-199	Make sure the user dashboard sidebar is responsive for all screen sizes (make it collapsible)	MN Marium Nur	DONE
☑	SCRUM-190	Add error messages for the URL Shortener (e.g. invalid URL, custom word already used, etc)	MN Marium Nur	DONE
☑	SCRUM-201	Make sure the settings component is responsive for all screen sizes	MN Marium Nur	DONE
☑	SCRUM-177	Create GUI for URL Shortener Pro User	MN Marium Nur	DONE
☑	SCRUM-78	Add a functionality to return summarized texts with citations	MS Maalik Siddiqui	DONE
☑	SCRUM-83	Add functionality to switch between dark and light mode	AS Anjali Sabu	DONE
☑	SCRUM-192	Create Templates component for user dashboard	AS Anjali Sabu	DONE
☑	SCRUM-81	Add functionality to "Log in with Google"	AA Amani Anderson	DONE

1-24 of 24

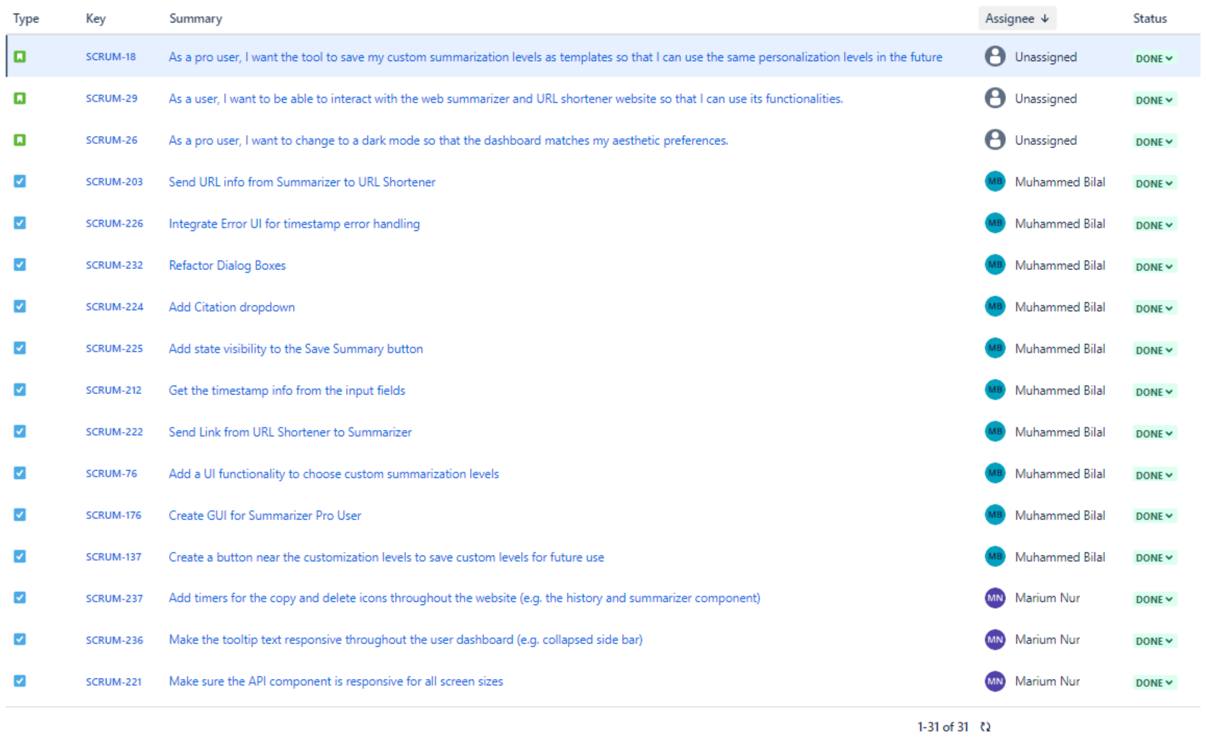


✓	SCRUM-217	Add backend handling for inputting an incorrect URL in the URL shortener	AA Amani Anderson	DONE
✓	SCRUM-214	Connect saved templates to the displayed template on the Summarize page	AA Amani Anderson	DONE
✓	SCRUM-213	Add error handling for custom word URL shortening	AA Amani Anderson	DONE
✓	SCRUM-207	Add UI and backend functionality to allow users to save custom templates	AA Amani Anderson	DONE
🟩	SCRUM-23	As a pro user, I want to log in to the tool using my Google account so that I don't have to use my email manually to create an account.	AA Amani Anderson	DONE
✓	SCRUM-210	Integrate backend custom word shortening capability with frontend	AA Amani Anderson	DONE
✓	SCRUM-197	Integrate Google's API with the back end to handle user logins using a Google account	AA Amani Anderson	DONE
✓	SCRUM-206	Add test cases for authentication functionalities	AA Amani Anderson	DONE

1-24 of 24

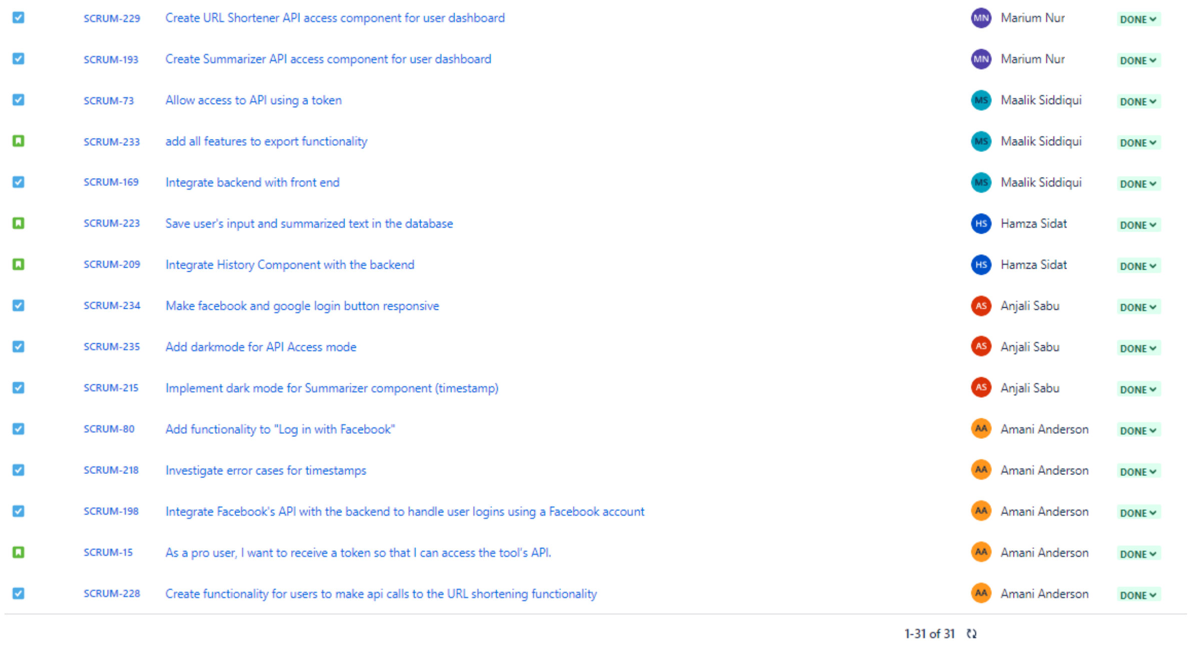
8.6. Sprint 6 (5 Apr - 19 Apr)

During the sixth sprint, we introduced API keys, enabling users to access Shorify's API, integrated with the frontend. Backend support for YouTube timestamp summarization was integrated into the frontend. Facebook authentication was implemented, accompanied by robust error handling to prevent multiple logins with the same email. Export functionality for summarization inputs and outputs as JSON files was successfully developed. An API access frontend was added into the user dashboard, ensuring full responsiveness across all screen sizes. We procured a VPS and domain name for website hosting, launching the frontend on the VPS' IP address. Additionally, the history page was enhanced with functional select, delete, and copy buttons for improved user experience.



Type	Key	Summary	Assignee ↓	Status
🟩	SCRUM-18	As a pro user, I want the tool to save my custom summarization levels as templates so that I can use the same personalization levels in the future	Unassigned	DONE
🟩	SCRUM-29	As a user, I want to be able to interact with the web summarizer and URL shortener website so that I can use its functionalities.	Unassigned	DONE
🟩	SCRUM-26	As a pro user, I want to change to a dark mode so that the dashboard matches my aesthetic preferences.	Unassigned	DONE
✓	SCRUM-203	Send URL info from Summarizer to URL Shortener	MB Muhammed Bilal	DONE
✓	SCRUM-226	Integrate Error UI for timestamp error handling	MB Muhammed Bilal	DONE
✓	SCRUM-232	Refactor Dialog Boxes	MB Muhammed Bilal	DONE
✓	SCRUM-224	Add Citation dropdown	MB Muhammed Bilal	DONE
✓	SCRUM-225	Add state visibility to the Save Summary button	MB Muhammed Bilal	DONE
✓	SCRUM-212	Get the timestamp info from the input fields	MB Muhammed Bilal	DONE
✓	SCRUM-222	Send Link from URL Shortener to Summarizer	MB Muhammed Bilal	DONE
✓	SCRUM-76	Add a UI functionality to choose custom summarization levels	MB Muhammed Bilal	DONE
✓	SCRUM-176	Create GUI for Summarizer Pro User	MB Muhammed Bilal	DONE
✓	SCRUM-137	Create a button near the customization levels to save custom levels for future use	MB Muhammed Bilal	DONE
✓	SCRUM-237	Add timers for the copy and delete icons throughout the website (e.g. the history and summarizer component)	MN Marium Nur	DONE
✓	SCRUM-236	Make the tooltip text responsive throughout the user dashboard (e.g. collapsed side bar)	MN Marium Nur	DONE
✓	SCRUM-221	Make sure the API component is responsive for all screen sizes	MN Marium Nur	DONE

1-31 of 31

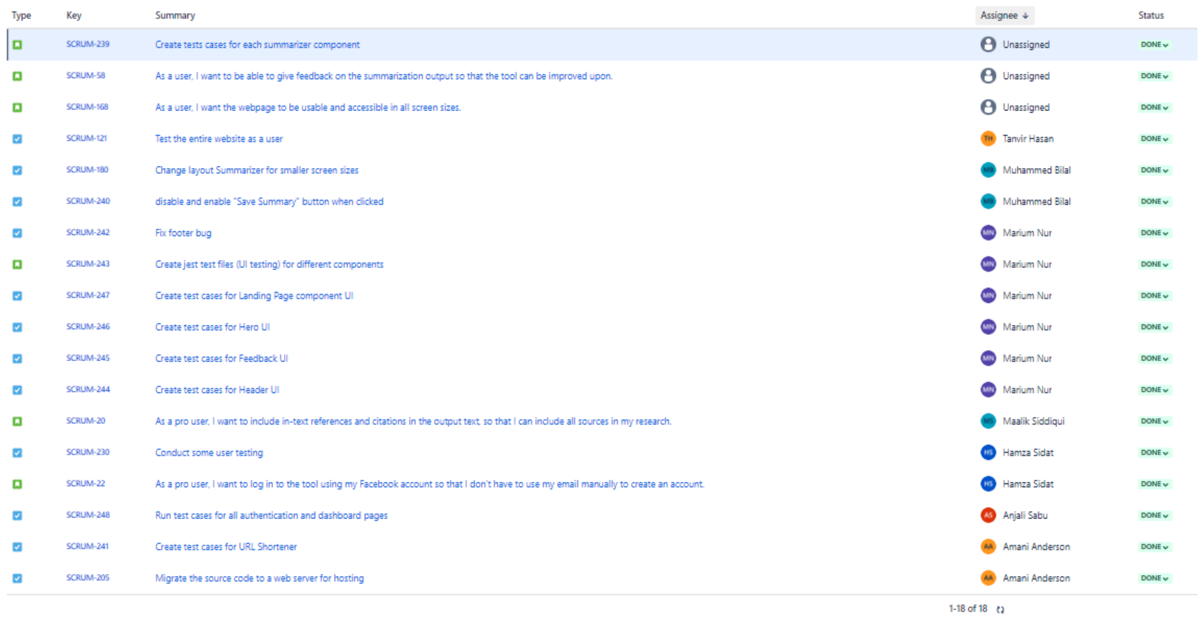


✓	SCRUM-229	Create URL Shortener API access component for user dashboard	Marium Nur	DONE
✓	SCRUM-193	Create Summarizer API access component for user dashboard	Marium Nur	DONE
✓	SCRUM-73	Allow access to API using a token	Maalik Siddiqui	DONE
🟩	SCRUM-233	add all features to export functionality	Maalik Siddiqui	DONE
✓	SCRUM-169	Integrate backend with front end	Maalik Siddiqui	DONE
🟩	SCRUM-223	Save user's input and summarized text in the database	Hamza Sidat	DONE
🟩	SCRUM-209	Integrate History Component with the backend	Hamza Sidat	DONE
✓	SCRUM-234	Make facebook and google login button responsive	Anjali Sabu	DONE
✓	SCRUM-235	Add darkmode for API Access mode	Anjali Sabu	DONE
✓	SCRUM-215	Implement dark mode for Summarizer component (timestamp)	Anjali Sabu	DONE
✓	SCRUM-80	Add functionality to "Log in with Facebook"	Amani Anderson	DONE
✓	SCRUM-218	Investigate error cases for timestamps	Amani Anderson	DONE
✓	SCRUM-198	Integrate Facebook's API with the backend to handle user logins using a Facebook account	Amani Anderson	DONE
🟩	SCRUM-15	As a pro user, I want to receive a token so that I can access the tool's API.	Amani Anderson	DONE
✓	SCRUM-228	Create functionality for users to make api calls to the URL shortening functionality	Amani Anderson	DONE

1-31 of 31

8.7. Sprint 7 (19 Apr - 25 Apr)

In the seventh sprint, the frontend team implemented Jest test cases to thoroughly assess the UI's functionality. The VPS was configured with essential packages and installations to support all backend features seamlessly. Ensuring robust performance, we optimized the backend code to run efficiently on the VPS for extended durations, managing multiple Flask servers concurrently through worker threads. Additionally, we enhanced code readability and maintainability by adding comments to JavaScript and Python files, while also addressing and resolving React warnings for a polished final product.



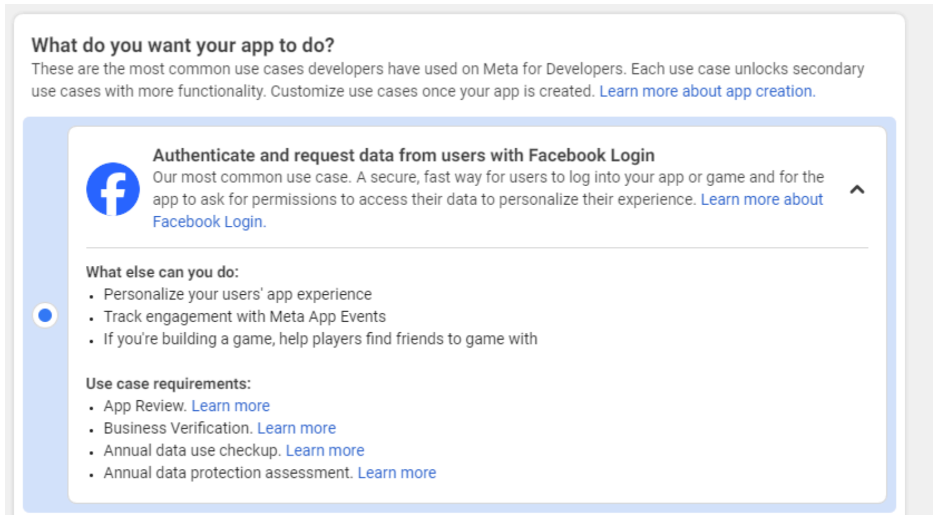
Type	Key	Summary	Assignee ↓	Status
🟩	SCRUM-239	Create tests cases for each summarizer component	Unassigned	DONE
🟩	SCRUM-58	As a user, I want to be able to give feedback on the summarization output so that the tool can be improved upon.	Unassigned	DONE
🟩	SCRUM-168	As a user, I want the webpage to be usable and accessible in all screen sizes.	Unassigned	DONE
☑	SCRUM-121	Test the entire website as a user	Tanvir Hasan	DONE
☑	SCRUM-180	Change layout Summarizer for smaller screen sizes	Muhammed Bilal	DONE
☑	SCRUM-240	disable and enable "Save Summary" button when clicked	Muhammed Bilal	DONE
☑	SCRUM-242	Fix footer bug	Marium Nur	DONE
🟩	SCRUM-243	Create jest test files (UI testing) for different components	Marium Nur	DONE
☑	SCRUM-247	Create test cases for Landing Page component UI	Marium Nur	DONE
☑	SCRUM-246	Create test cases for Hero UI	Marium Nur	DONE
☑	SCRUM-245	Create test cases for Feedback UI	Marium Nur	DONE
☑	SCRUM-244	Create test cases for Header UI	Marium Nur	DONE
🟩	SCRUM-20	As a pro user, I want to include in-text references and citations in the output text, so that I can include all sources in my research.	Maalik Siddiqui	DONE
☑	SCRUM-230	Conduct some user testing	Hamza Sidat	DONE
🟩	SCRUM-22	As a pro user, I want to log in to the tool using my Facebook account so that I don't have to use my email manually to create an account.	Hamza Sidat	DONE
☑	SCRUM-248	Run test cases for all authentication and dashboard pages	Anjali Sabu	DONE
☑	SCRUM-241	Create test cases for URL Shortener	Amani Anderson	DONE
☑	SCRUM-205	Migrate the source code to a web server for hosting	Amani Anderson	DONE

1-18 of 18

9. Problems

9.1. Unable to publish Facebook Login App

As mentioned in the Facebook Login section, the Facebook Login option is not functional to external users at this time. The reason behind this being that Facebook REQUIRES business verification for the app to be published. The app cannot be accessed by anyone other than the developer that owns the app during testing, so we could not even add other developers to the test. There may be ways around this, but in the week or so that we worked on this, we couldn't find a workaround, as it seems Facebook is strict on requiring business verification. Below is a screenshot of this requirement.



What do you want your app to do?

These are the most common use cases developers have used on Meta for Developers. Each use case unlocks secondary use cases with more functionality. Customize use cases once your app is created. Learn more about app creation.

● **Authenticate and request data from users with Facebook Login**

Our most common use case. A secure, fast way for users to log into your app or game and for the app to ask for permissions to access their data to personalize their experience. Learn more about Facebook Login.

What else can you do:

- Personalize your users' app experience
- Track engagement with Meta App Events
- If you're building a game, help players find friends to game with

Use case requirements:

- App Review. Learn more
- Business Verification. Learn more
- Annual data use checkup. Learn more
- Annual data protection assessment. Learn more

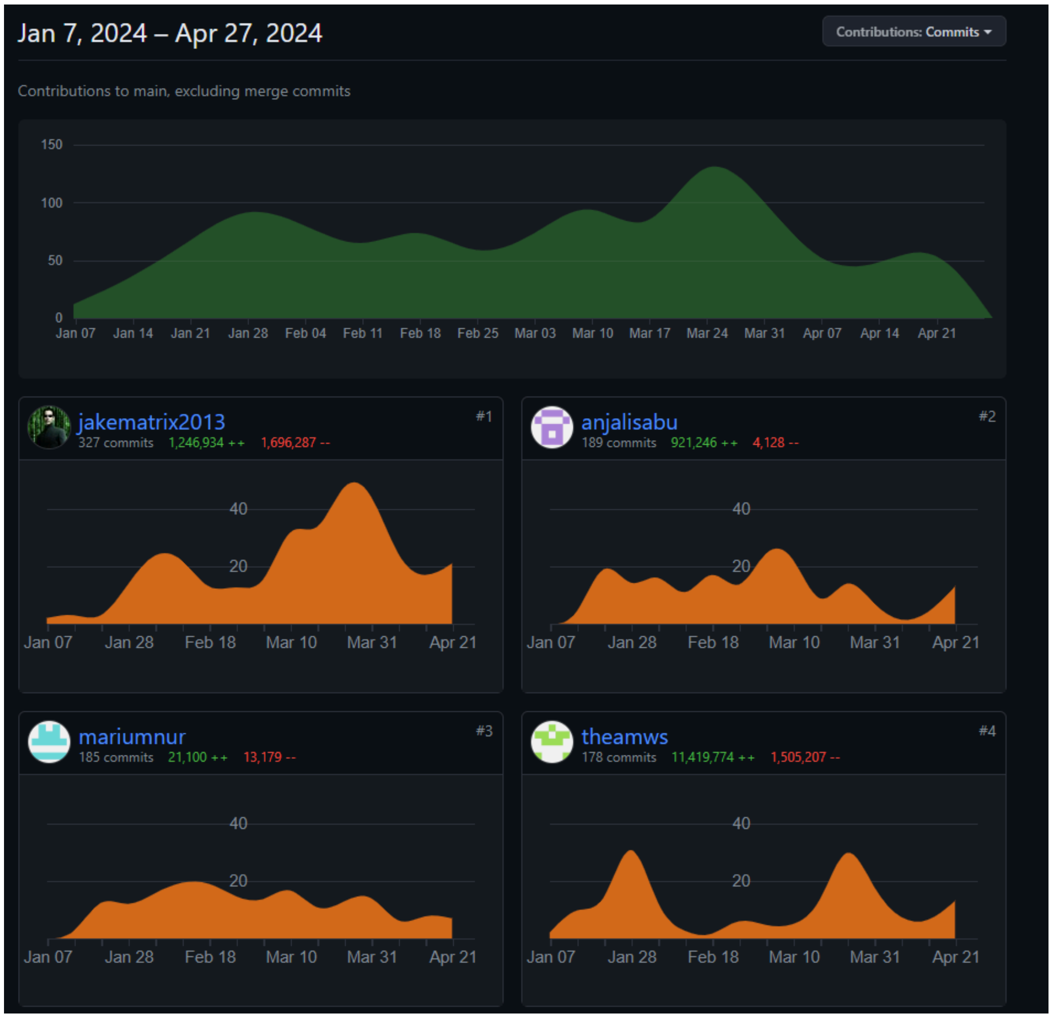
9.2. Website & No Caption Youtube video Summarization on hosted VPS

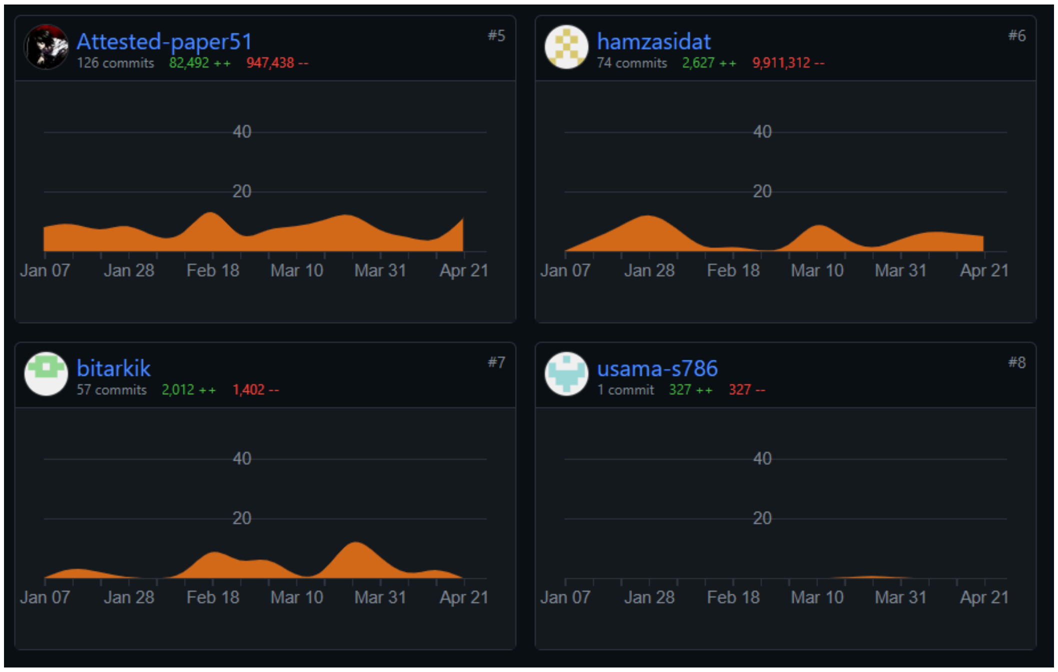
There were a multitude of hurdles to overcome when migrating the website to our Virtual Private Server. Eventually, we were able to have the front-end render, and communicate with the backend with SSL compliance. Due to the fact that we have multiple Flask servers on their own ports running, and the fact that these Flask servers must continue to run without requiring an active terminal, we utilized Gunicorn to run these apps on their dedicated ports. Gunicorn is a python Web Server Gateway Interface HTTP server which allows us to run python web servers concurrently and without using an active terminal. This is a new system to us, so we expected that every functionality may not perform optimally. There are two known issues with this approach. Firstly, web page summarization seems to not work on the VPS with the use of Gunicorn. We are not certain of the root cause, but the suspicion is that because our web page scraper relies on the web page being opened in Chrome, or some browser in order for it to scrape the content, it does not work without an active terminal being opened. In our code, we attempted to make the browser open in 'headless' mode, without a visual browser displaying, but we noticed that the browser still opens even when ssh'ed into the server and while Gunicorn is running. So, if the web page summarization does not work, it could be because there's no active ssh connection and the webpage doesn't open. This is, however, just a hypothesis. The other problem we encountered is with Youtube videos that have no captions/transcripts available. In this case, we rely on ffmpeg, a multimedia framework that allows us to download the audio and then transcribe it manually. The problem encountered initially is that when running the server with Gunicorn, we could not explicitly state the 'path' of where ffmpeg was installed on the VPS. This was thought to have been solved by adding the location of ffmpeg to the environmental variables path on the VPS, but this also seems to be causing issues. These are issues that are certainly resolvable, but we did not have the time to tackle given how late we had migrated our website to the server.

9.3. Summarization Length Accuracy

The length slider feature of our summarizer posed significant challenges during development, particularly in accommodating the diverse lengths of input texts. Implementing fixed word counts for short, medium, and long summaries proved impractical. For example, setting a 'short' summary to 100 words works well for longer texts, but if the original text is already under 100 words, the summary could inadvertently exceed the length of the input. To address this, we opted for a more dynamic approach, allowing the AI to adjust the summary length based on the content. The AI evaluates the context and content density to decide the appropriate summary length, but even then, the summaries would sometimes be of the same length, or slightly shorter.

10. Github Contributions


Jan 7, 2024 – Apr 27, 2024
Contributions: Commits
Contributions to main, excluding merge commits
150
100
50
0
Jan 07 Jan 14 Jan 21 Jan 28 Feb 04 Feb 11 Feb 18 Feb 25 Mar 03 Mar 10 Mar 17 Mar 24 Mar 31 Apr 07 Apr 14 Apr 21
jakematrix2013
327 commits 1,246,934 ++ 1,696,287 --
#1
anjalisabu
189 commits 921,246 ++ 4,128 --
#2
mariumnur
185 commits 21,100 ++ 13,179 --
#3
theamws
178 commits 11,419,774 ++ 1,505,207 --
#4
40
20
Jan 07 Jan 28 Feb 18 Mar 10 Mar 31 Apr 21



Username	Team Member
jakematrix2013	Amani
anjalisabu	Anjali
mariumnur	Marium
theamws	Maalik
Attested-paper51	Bilal
hamzasidat	Hamza
bitarkik	Tanvir

11. Links

Github Repository:
https://github.com/Attested-paper51/COSC-4P02-Web-Summarizer
Website Link: https://4p02shortify.com/
Meeting Notes: All Meeting Notes