BROWSER BUDDY FINAL REPORT

SP-17 Browser Extension

Elliott Brown Nathan Lynes Jacob Germana-McCray

Professor Perry | *04/10/24*
Kennesaw State University | *4850-02/01 Senior Project*
Lines of Code: 440 | **Unique Components: 7**

***GitHub**: https://github.com/Browser-Buddy/browser-buddy*
***Website:** https://browser-buddy.github.io/site/*

Table of Contents

Abstract

This report describes the process by which this team planned, analyzed, designed, implemented, tested, and maintained the Firefox web extension Browser Buddy—which provides users with an AI-powered text summarization tool. Moreover, a system overview will be detailed, and the challenges encountered throughout this process will be explored.

Introduction

Background

With the increasing amount of information available online, efficiently capturing and creating key points from web content has become a major challenge for many users.

The increasing development of generative pre-trained transformer (GPT) models has broadened the user base for tools implementing these models, the most popular of these tools being ChatGPT, which uses the same model featured in this report: GPT-4.

Browser Buddy is one such tool, and this browser extension allows users to efficiently identify key points from large sections of text on webpages.

Objectives

This report will cover the process our team underwent to produce Browser Buddy.

Specifically, we will highlight five areas of interest: our methodology, our testing plan, our implementation, and what we learned from this project.

The methodology consists of every working part defined from the scope. This includes our developmental methods, system design, and technologies used.

We also explain the finished product and how we determined the product was finished—which is further described in the testing plan.

Scope & Limitations

The objective of this project is to deliver a web browser extension for the summarization of web pages—specifically web pages from *wikipedia.com*.

Our system will require no databasing, as there is no planned functionality for saving user data. Browser Buddy will be installed by the user onto any Firefox web browser, and the user will access the functionality through the context menu.

Users will employ Browser Buddy to send data to and from ChatGPT-4 using the servers from OpenAI; the data sent will be user-specified, raw text data, and the data received will be a summary of that raw text data.

Software Development Life Cycle

We implemented the following strategy for developing this project:

1. **Plan**
   We created a stable, logical project plan that foresaw every step of the project's cycle. We used this as our framework.
2. **Analyze**
   This step includes our Requirements document, which may be found in the subsequent section. Our project was analyzed from top-down and down-up to ensure no crucial details were occluded.
3. **Design**
   In developing the Design document for Browser Buddy, we found many nuances that we would have to overcome. This led to a few changes in our requirements, which are appropriate in every step of the SDLC.
4. **Implement**
   This step defined the latter half of the project. We found it necessary to begin programming as soon as possible, as this would give us familiarity with the frameworks and APIs we would be using.
5. **Test**
   This step encompassed nearly the entire project. Each step of developing Browser Buddy called for some tester application—which we would develop and test to see where our requirements and design documents might fail. Towards the end of the project, we were testing the Browser Buddy application directly—this was to verify user stories.
6. **Maintain**
   Like testing, we performed maintenance throughout the entire project.

This aided in reforming the requirements and design documents, as well as in fixing any errors within our code—which was upkept through GitHub source control.

Requirements

Design Constraints

Contained in this section are constraints our team is placing upon the development of Browser Buddy.

Environment

Browser Buddy will work as a browser extension within the Firefox desktop browser on Windows and MacOS devices. Internet connection is a requirement due to the nature of the plugin.

User Characteristics

The target users are students, researchers, professionals, or anyone who frequently reads lengthy online content. Users should have basic web browsing literacy and English reading/writing skills.

System

Browser Buddy will consist of a browser extension frontend using HTML/CSS/JavaScript that uses the WebExtensions API which is shared between Chrome and Firefox, though the main development goal is for a Firefox-specific extension. The stated technologies will be used as a front-end interface for our API connection with OpenAI's GPT-4 model.

Functional Requirements

All functional requirements that this system will implement are detailed below. This will include a description of every function and the level at which they are present.

Users

There is no plan to implement administrative accounts, as we feel this function is unnecessary considering there are no databases, transactions, or logs to be viewed or recorded. Appropriately, every user of the service will be treated the same.

Users to the service will receive the following functions:

Install/Uninstall

Clearly, users must possess the ability to add or remove the program from their browser as desired.

The absence of this function would be unethical. Fortunately, our team will have no need to develop such a feature, as this is native to most modern web browsers. For our purposes, Firefox does possess this feature.

Text Highlighting and Context Menu

Browser Buddy is intended to be used within the context menu. The context menu is the pop-up that appears after right-clicking elements within the web browser—again, this is present in most modern web browsers. A simple button will be added into the native context menu, and the button will be appropriately labeled so that the user will understand its purpose.

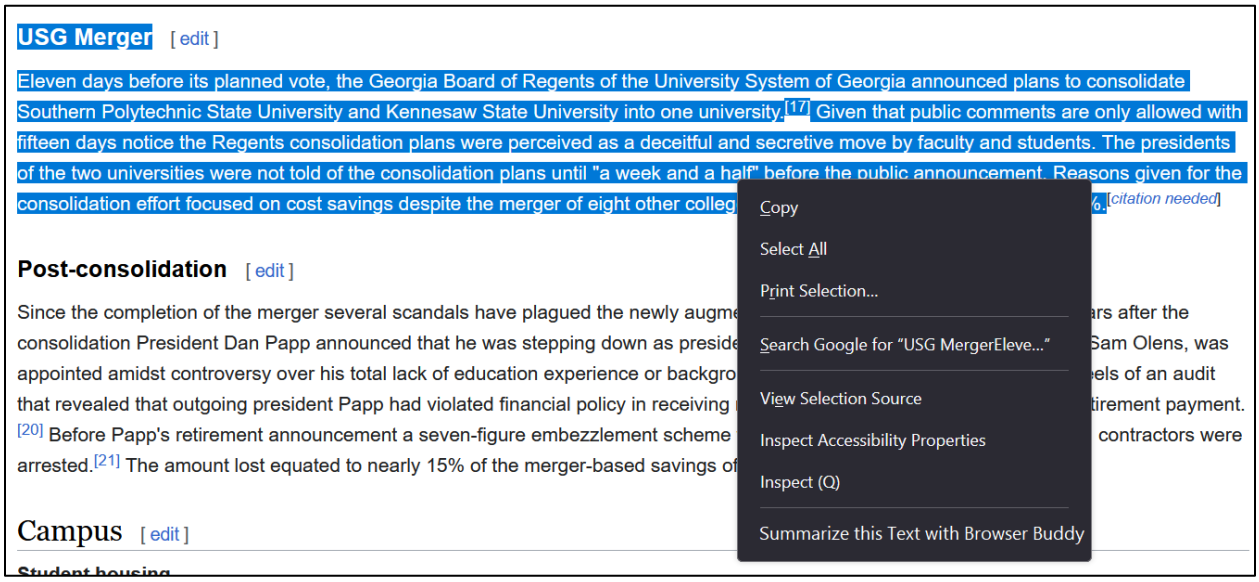

Figure 1: Mockup of Wikipedia page with Context Menu

Moreover, this button will become available after highlighting text within the webpage. It is this text that the extension will send through API calls and retrieve a summary from. Figure 1 depicts what we expect this extension to look like using the Wikipedia page for Southern Polytechnic State University [1].

Summary Window

Upon selecting the button outlined in Section 3.1.2, the summarized text will be displayed to a new pop-up window near the top of the user's screen. Figure 2

provides a mockup of this implementation. This window will feature a text box that holds the summarized text, and the box will disappear upon the user clicking any other element on the page.

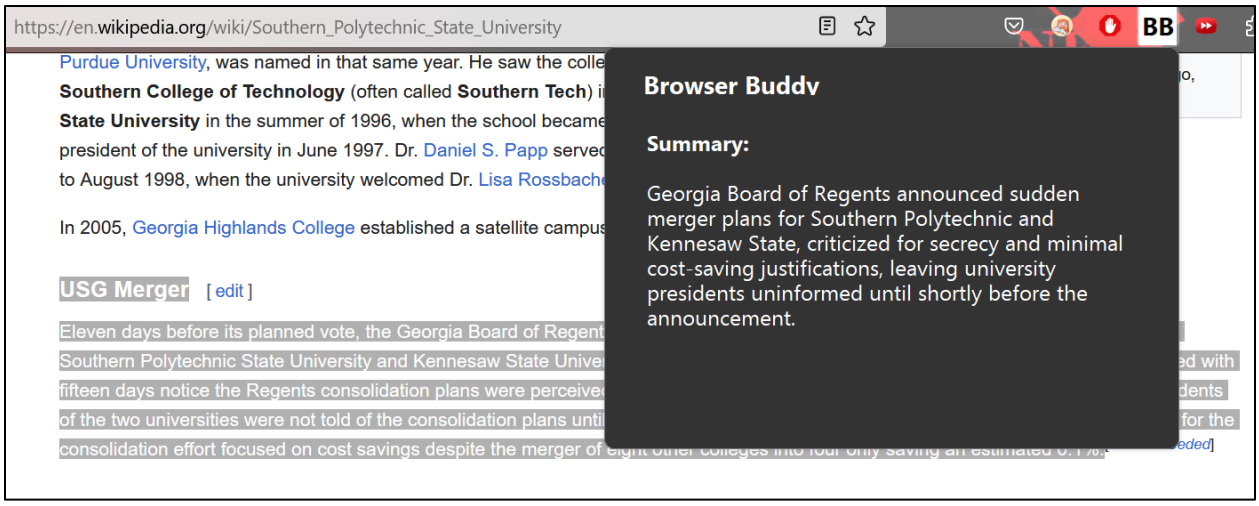

Figure 2: Mockup of Summary Window

Backend

Backend refers to any functions the user does not see or interact with. Of course, the user interacts with features of the GUI, but the functions described here are required for the APIs to function properly.

Obtain API Key

Every instance of the program will require a key to communicate with OpenAI's API. Keys exist so that only verified users have access to proprietary functions defined by companies, organizations, or institutions—such is the case here. A key will be obtained before the program attempts to call the API after a predefined user input is received.

Communicate with OpenAI

Browser Buddy sends highlighted user input to OpenAI so that a summarized text may be returned.

Nonfunctional Requirements

This section lists arbitrary attributes of our program.

Security

Users should not fear a breach of personal information or loss of their private key.

Capacity

As this extension relies upon the availability of the OpenAI servers, the capacity for users to send/receive data is limited to various features like server downtime, Internet speed, location, etc. Users should not expect to wait for a request to be filled.

Moreover, the local speed of the program should not be an issue. Observing the scope of this project, there is no need for memory-intensive or processor-intensive algorithms or tasks; therefore, this program should not require more than the minimum requirements for running just the browser.

Usability

The extension should be usable and understandable to most users. We intend on implementing proper UIE techniques to ensure that our target demographic will require the least amount of assistance or training possible.

Other

Depending on how OpenAI marshals the use of their API, we may or may not need to modify the code of this project so that future users can use the more recent versions of the API. Our maintenance should be very minimal, and it must be noted that every call to the API costs $0.001. Since this project is to remain within the realm of KSU, this cost should be negligible.

External Interface Requirements

User Interface

Users will interact with the software via their web browser. Due to the high variance in web browser layouts and designs, we are designing a GUI that pairs well with the default Firefox web browser layout. Browser Buddy's GUI does not require too much

attention either; a simple popup window does not impact the user's overall interaction with their web browser.

Software Interface

Our application is run exclusively through Mozilla Firefox. Therefore, our application needs to use Firefox's appropriate extensions for usage of extensions.

API Interface

To communicate with OpenAI's ChatGPT-4 model, we require an interface for sending and receiving data. Again, this will be configured to work with Mozilla Firefox.

Testing Plan

Due to the nature of this project, the extent of the testing routine should be as complex as testing every user story. Again, the lack of a database and lack of different levels of users does not open many opportunities to test the project outside of its required functionality.

Of course, we intend on reinforcing all the requirements. Given there is enough time and resources remaining, we may modify the source of our code to allow for faster execution times and faster communication. Since our project heavily relies on the use of an external API, the speed we will observe should be reliant on various factors outside the control of the program itself—such as internet speed, server uptime, etc.

The results of this testing will be compiled and reported towards the end of development.

Design

Design Considerations

This section highlights determinants for the system's design.

Assumptions and Dependencies

- Summarized content will be sufficiently comprehendible.
- Users have Firefox and a stable internet connection.
- Third-party API will stay consistently available.

General Constraints

- Avoid heavy resource usage and slow performance.
- Support cross-platform between MacOS & Windows.

Goals and Guidelines

- Provide a one-click summarization experience.
- User-friendly interface and UX.
- Modular components for maintainability.
- Ensure responsiveness.

Development Methods

We will abide by a general programming format and style guide; such is defined below.

HTML/CSS

HTML/CSS will conform to the HTML5 & CSS3 standards. Kebab-case styles will be observed. Four-space indentation will be used for nested statements. Maximum column space will be set to 80 characters.

JavaScript

JavaScript will follow ES6 standards. There will be no third-party libraries used. Camel-case style will be used. Four-space indentation will be used for nested statements. Maximum column space will be set to 80 characters.

Architecture

Contained here is a list of directories that our software will be housed in. A basic description of each section is provided.

Root

Root of the browser buddy extension.

- / (root)
  - Manifest.json
  - README.md

Assets

Iconography that will be used throughout the program.

- /assets – static assets
  - Icons/

Source

The logic and styling for the extension. Each file will be explained further in section 5.2.

- /src – Core extension source code
  - Background.js
  - contentScript.js
  - summarize.js
  - api.js
  - cache.js
- /stylesheet.css – style for the popup.

Documentation

Documentation will provide a succinct description of how the program functions, as well as a developer quick-start guide. The user documentation will describe the usual flow of the program understandable to an average user.

- /docs – Documentation
  - Developer Documentation.md
  - System Design.md
  - User Documentation.md

Policies

- Meet browser extension security policies.
- Request only required permissions.
- Transmit no user data externally.
- Cache summaries using browser storage best practice.

Detailed Design of System

Classification and Definition

Browser Buddy is a *web extension*, which is an application that is executed within a web browser. Specifically, this web extension uses external APIs to process user-provided data.

Detailed Overview

The browser buddy extension will consist of several files operating as individual services.

- Background – Adds a right-click option for summarization, listens for a selection and sends the any highlighted text on the browser to summarize.js.
- Summarize – processes highlighted text into a prompt. This consists of appending a hardcoded string designed to guide the LLM into providing a better response, as well as an option string which will enable the ability to change "formats" of the response (i.e. bulleted list, paragraph, simple terms). The fully appended prompt "promptStr" will then be sent to the api_handler.
- API_Handler – prepares a POST request using JavaScript's fetch() method using the prompStr built in the summarize service. After this, the api_handler will wait for a response from OpenAI, process each byte of the stream into readable strings, and push them to the content_script.
- Content_script – Updates the DOM using the response gathered by the API_Handler. Builds the widget that displays the response.

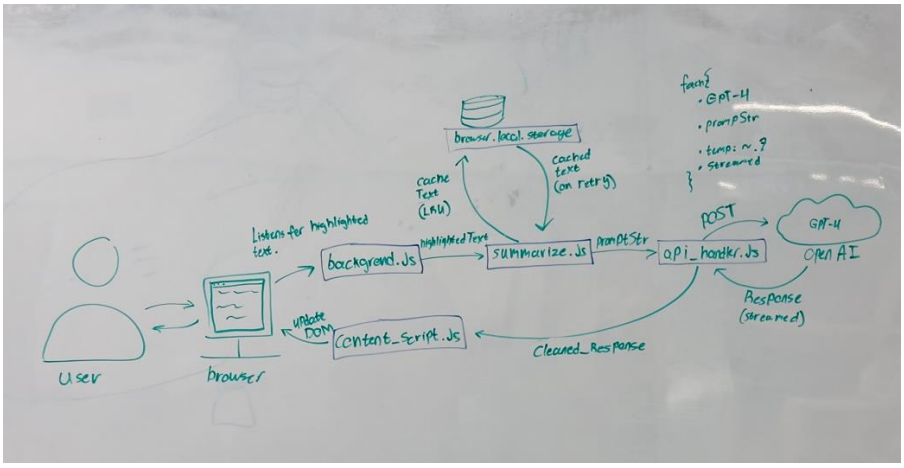

Figure 2. Software Design Diagram of the Browser Buddy extension.

Constraints

If the extension was to operate on a large scale, we would not be able to keep the hardcoded API key that we are currently using.

Future use would require that we publicly share our API key (this is a security problem) or require each user to pay for a private key. This change does open the door to monetizing Browser Buddy, as advertisements may be used to fund the API key payments. This limitation does not imply a significant change of functionality to support a large user base, so we only mildly consider it a limiting factor in our development.

User Stories

"I'm David, and I want to understand as much information on webpages as fast as I can, so that I can quickly participate in conversation about the topic therein."

- David is often late preparing for meetings at his office job, and he often is familiarizing himself with the points of content in the meeting as it happens. David wants to gain a quick and foundational understanding of the topics described on Wikipedia.

"I'm Emmy, and I want to read summaries of my friend's blogs in bullet points, because they talk too much, and I want to read it in specifically bullet point form."

- Emmy's friends post too much on their blogs and she often is expected to know about the things they have posted! Emmy wants to quickly go through their posts for the week and be done with it so that she does not miss valuable information about her friends' lives!

"I'm Eathan, and I want to black box new skills as quickly as possible so that I can complete my assignments in time."

- Eathan's boss often gives him assignments to create something for him that requires skills he has never learned before. Eathan is motivated to meet the expectations placed on him and needs to quickly understand many sources of information which introduce these new skills to him to succeed.

Resources and Processing

Due to the size of this project, the resource requirement is being considered as negligible. Web extensions—especially ones of this magnitude—should not require much memory or processing power. All processing will be done locally, so performance is dependent from machine to machine.

Interface

Our interface will be very briefly covered here. Figures 3 and 4 depict our GUI.

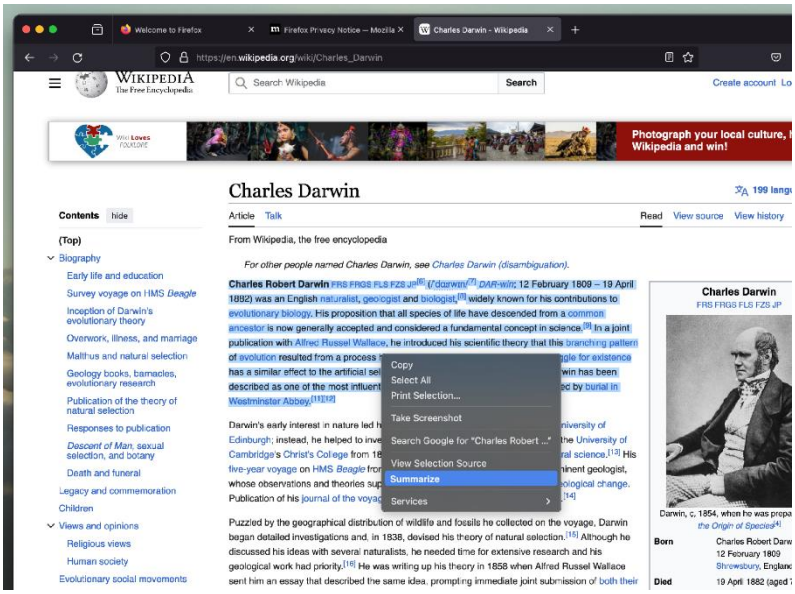

Figure 3. Demonstration of the "Summarize" button in the context menu.

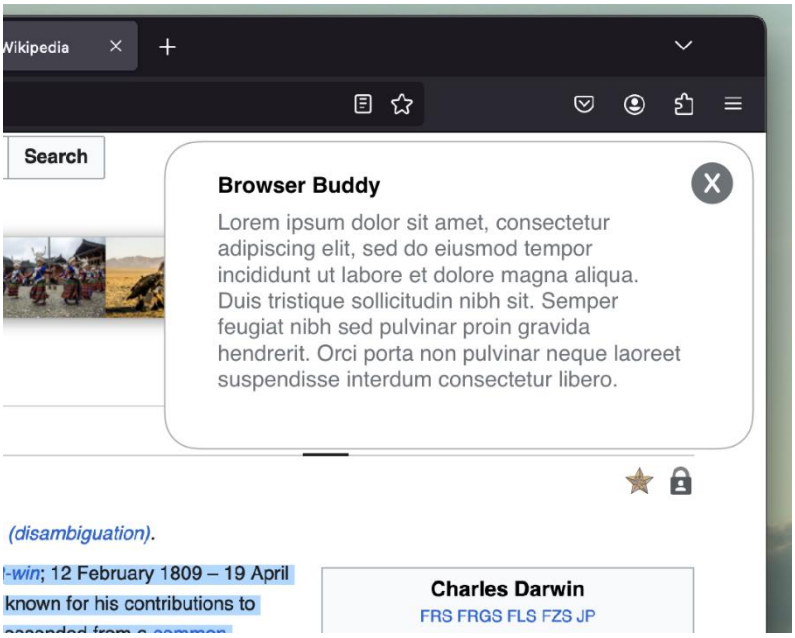

Figure 4. Demonstration of the styling of a summarization after the "Summarize" button is pressed.

Development

Development Approach

We approached this project in order of the SDLC outline defined previously. How we approached the Implement epoch is expounded upon here.

Development was done in an independent manner, with each team member doing their own solo work and then convening every week or so to integrate their ideas and code into the actual project repository.

During our development, we encountered a multitude of issues, which led to us reforming our requirements and adjusting any previous documents. Weekly, we would meet to discuss the state of the project, what we had been working on, and where we needed to be. This approach greatly helped us identify issues with each member's code, and our Git merges were better orchestrated, as explained in the Git and Source Control subsection.

The code was written and rewritten until the requirements were either met or determined to be infeasible. Our nonfunctional requirements were also addressed throughout this cycle. As listed in the requirements document, we focused on maintaining security, ample capacity, and usability. We did this by carefully analyzing the code for memory leaks, which may lead to cases where the given requirements are breached.

Process

Initially, our project used the OpenAI GPT-3-Turbo model. This model did the job, but there was a significant lack of quality. The summaries we received had no substantial information in them, and this is where we started looking for other options.

We moved to OpenAI GPT-4 and noticed a significant increase in quality. However, this increase in quality came at a cost, and our API cost increased.

Towards the beginning of the project, we had high hopes for Browser Buddy. However, since this style of program is unfamiliar to our team, we could not fully grasp the undertaking that Browser Buddy was. Our project planning was very meticulously orchestrated, and we chose to forego many features, as we were unsure if our ideas were possible. In doing so, we ended with a handful of quality features instead of innumerable mediocre features.

For example, our initial idea for Browser Buddy was a browser extension that directly modified the HTML elements of a given webpage. This idea was ambitious, and we correctly foresaw this feature being an impossibility.

Since our application comprises of mostly dynamically generated HTML elements, development revolved around ensuring these elements matched our stylistic needs. Thus, testing was performed consistently throughout the development of Browser Buddy. Our intermittent and final testing data is compiled in the Testing section of this report.

Technologies/Tools Used

Each project member found use in a different IDE, and we saw no reason to restrict ourselves to one environment. Our team employed the following IDEs:

- VSCode
- Notepad++
- NeoVim

Each team member did brief research on the APIs that we would need to finish the browser extension. Since we used legacy programming languages (HTML, CSS, and JavaScript), there were no certificates to be obtained prior to development. The following APIs were used:

- Mozilla Firefox's WebExtensions
- Node.js
- npm
- OpenAI GPT-3.5 and GPT-4

Of course, we tested the browser extension using Firefox. However, Firefox provides a dedicated environment for testing browser extensions, and this may be accessed prior to running the command `web-ext run` in the terminal. From here, we were able to directly see the contents of the cache, the status of our dynamically created HTML elements, and the status of our API calls.

Feature Description

We define a feature as any purposefully implemented portion of code that serves a purpose to the front and/or back end of the browser extension. During development, we considered any requirement written into the functional and nonfunctional requirements to be a feature.

Browser Buddy contains various features that were carefully planned and articulated. Certain features are locked behind other features. For example, a user is unable to summarize text without first entering an API key.

Our final implementation contains the features found in Table 1.

Feature	Usage
Context menu access.	Right click anywhere in the webpage after highlighting text, and the Summarize option should be clickable.
Type OpenAI API.	User obtains an API key and pastes it into the prompt window.
Submit OpenAI API.	User pressed Submit in the prompt window to save the API key to the cache.
Summarize text.	Select the Summarize option in the context menu. After an API key is submitted, then an HTML element displaying the summarized text will appear.
Programmatically append HTML elements to the webpage.	Prompt window and summarize windows (alongside their respective child elements) are created with JavaScript scripts. These are displayed in the top-right.
Send API call to OpenAI.	After clicking Summarize with an API key in the cache. This requires an Internet connection, but the program does successfully send/receive data.
Retry the summary.	Click the retry button in the summary window.
Copy the summary.	Click the copy button in the summary window.
Exit Browser Buddy.	Click the X in the top right of the prompt or summary window. Removes the current summary.

Table 1: List of features and their respective uses.

Firefox Browser Cache

We chose Firefox as the environment for our web extension, and because of this, we may take advantage of the features provided in Firefox through the WebExtensions API. One feature that greatly improves the quality of life of Browser Buddy is the cache.

In Firefox, every browser extension that is active and running will be given its own portion of the cache. Before we needed the user to manually upload their OpenAI API key, we saw no need for this cache, but after our change in approach, we realized that this cache could be used to store the API key. Without storing the key in Browser Buddy's cache, the user would need to resubmit the API key every time they needed to summarize text. We wrote code such that the browser extension saves the provided API key into the cache.

For our purposes, the cache serves as a repository for the API key and nothing else.

Git and Source Control

For source control, we used Git, and this was hosted on GitHub. We upkept a policy of unanimous agreement to merge code; this was to ensure that no one person was behind a branch. Moreover, we ensured our code matched requirements at every step of development, and our meetings greatly assisted this restriction.

We use an organizational account to host the code for the web extension as well as the website, which may be found at https://browser-buddy.github.io/site/index.html.

Browser Buddy's source code and history may be viewed with the following link: https://github.com/Browser-Buddy/browser-buddy.

Testing & Validation

Testing Strategy

To determine that our product was finished, we referenced the requirements document for Browser Buddy—which defines every feature our product should have.

Using the document as a sort-of checklist, we ensured that every requirement from the document was indeed met. This validation process was carried out through evaluation of the code, sample executions of the program, and reenacting user stories.

Results & Issues Identified

Our requirements match our final product, and we have not identified any issues with the program.

An unfortunate outcome from the project was our inability to port Browser Buddy to Google Chrome. We believe cross-browser support would greatly increase the user base, but we did not have the time to reconfigure the code to work with any non-Firefox browser.

The current state of Browser Buddy is verifiably complete, as we can see in the next section.

Final Validation

Overall, the program works as intended. Every user story we have defined is possible. Table 2 provides every test performed on the final model and the results.

Test Description	Test Notes	Result
Context menu displays “Summarize”.	Displays as expected.	Pass
When no API key is provided, prompt for an API key after selecting the Summarize option.	API key prompt screen displays properly.	Pass
X button closes the prompt window.	Prompt window successfully closes.	Pass
Type API key into prompt window.	Text may be entered.	Pass
Submit API key with Submit button.	API key is successfully saved to the cache.	Pass

After providing API key, the summary window is displayed with summarized text.	Summary window displays and provides a summary of the highlighted text.	Pass
Highlighted text is appended to the OpenAI GPT-4 prompt.	Text is clearly added, as it is successfully summarized.	Pass
Copy button in the summary window copies the summary to the clipboard.	Successfully clipped.	Pass
Retry button resubmits the summarized text.	Successfully re-summarized.	Pass
X button closes the summary window.	Successfully closed the summary window.	Pass
Summarize text in different languages.	Proven to work in Chinese, Italian, and Spanish.	Pass
Works in MacOS.	Loads successfully.	Pass
Works in Windows.	Loads successfully.	Pass

Table 2: Test report.

Lessons Learned

Challenges & Solutions

We experienced various challenges throughout the planning and development of Browser Buddy, but each problem was appropriately and succinctly addressed. These challenges and their solutions are listed below.

1. WebExtension API development is inconsistent between Firefox and Chrome. We decided to focus entirely on Firefox because our team has the most experience with this browser.
2. Developing around a secret API key complexified development. We had not expected our API key to get deleted by OpenAI after publishing the code to GitHub. To work around this, we created a prompt window. This way, the user can provide their own API key; in a sense, this only allows users of OpenAI to access the program.
3. Firefox disallows programmatically generating pop-up windows. This was difficult to overcome, as we had to change the code to create new HTML elements that sit on top of the already-loaded webpage.

Best Practices & What Worked

Properly planning every step of a project is a crucial step to a successful, within-budget implementation. This team found great success in this strategy.

Source/Version control greatly improved the speed with which the project was implemented.

Areas for Improvement

As a team, we have learned the importance of proper project planning and management. Our project planned for using 194 work hours. We ended the project with 196 work hours used, which is slightly over our goal. In the future, we intend on assigning more realistic work requirements such that these numbers match.

Additionally, we did not practice the greatest source control behavior. Projects should be updated in relatively small increments, and Browser Buddy was compiled in our GitHub in relatively large increments. In the future, we intend on pushing changes to GitHub with every new function we implement.

Conclusion

In this report, we discussed the process our team underwent to plan, analyze, design, implement, test, and maintain the browser extension Browser Buddy.

Our planning phase consisted of creating a Gantt chart, appropriately assigning work hours, and foreseeing the future steps of the SDLC; we attribute this planning phase to the success of our project. Additionally, our analysis of the project established suitable requirements that were both useful and within reason. The designing of Browser Buddy performed conjointly with the analysis, as we had to design a system that both fit the requirements and was apt to our scope.

As for the implementation, we successfully implemented everything detailed within the requirements and design. We practiced source control to prevent any loss of work, and this practice marshals itself well within our maintenance epoch, which was enacted throughout development. Alongside maintenance, we began testing our implementation as it was being developed, and this allowed for smooth development.

In all, we learned that proper planning and analysis is crucial to a successful development epoch. We greatly benefitted from practicing source control through Git. The success of Browser Buddy's implementation has provided each of us with a greater insight into the proper steps to incorporate into a production-level software development life cycle.

References

[1] Wikipedia contributors. (2023, October 26). Southern Polytechnic State University. In *Wikipedia, The Free Encyclopedia*. Retrieved 19:35, February 9, 2024, from https://en.wikipedia.org/w/index.php?title=Southern_Polytechnic_State_University&oldid=1181918685

Appendix

Glossary of Terms

Application Programming Interface (API) - A set of protocols, routines, and tools for building software applications that specifies how components should interact.

Minimum Viable Product (MVP) - A product with just enough core features to satisfy early customers and provide feedback for future product development.

Integrated Development Environment (IDE) – A software program used for developing, testing, and debugging code of a desired programming language.

Software Development Life Cycle (SDLC) – A form of project management by which epochs are staged to be completed in a certain order, allowing for some overlap.