Final Deliverable

StudySync Web Extension

Sunidhi Abhange

Gage Fleming

Hashem Ramadan

Mason Waldapfel

University of London

CM2020: Agile Software Projects

Team 68 (Tutor Group 6)

Table of Contents

[1 Background 3](#_Toc159449505)

[1.1 Introduction 3](#_Toc159449506)

[1.2 Literature 4](#_Toc159449507)

[1.3 Scope 5](#_Toc159449508)

[1.4 Group Work 7](#_Toc159449509)

[2 Planning and Research 9](#_Toc159449510)

[2.1 Research 9](#_Toc159449511)

[2.1.1 Google Extension Documentation 9](#_Toc159449512)

[2.1.2 Software Design and Development 10](#_Toc159449513)

[2.1.3 Discussion Prompts 11](#_Toc159449514)

[2.2 Planning and Iteration 12](#_Toc159449515)

[3 Prototyping and Iteration 14](#_Toc159449516)

[3.1 Prototyping 14](#_Toc159449517)

[3.2 Iteration 14](#_Toc159449518)

[4 Design 16](#_Toc159449519)

[4.1 User Interface 16](#_Toc159449520)

[4.2 Software Requirements Specification 16](#_Toc159449521)

[5 System Development 18](#_Toc159449522)

[5.1 Development Process 18](#_Toc159449523)

[5.1.1 Week One 18](#_Toc159449524)

[5.1.2 Week Two 21](#_Toc159449525)

[5.1.3 Week Three 24](#_Toc159449526)

[5.1.4 Week Four 28](#_Toc159449527)

[5.1.5 Week 5 32](#_Toc159449528)

[6 Analysis 33](#_Toc159449529)

[7 Evaluation 34](#_Toc159449530)

[8 Conclusion 35](#_Toc159449531)

[9 Individual Reflection (Name) 36](#_Toc159449532)

[10 Appendix 37](#_Toc159449533)

[11 References 38](#_Toc159449534)

# 1 Background

## 1.1 Introduction

Two months ago, our group proposed a project, StudySync, that would meet the vision statement below. The proposal laid out a quality plan to ensure our team could bring this vision to life. Our team has worked diligently during the past months to bring this project to life through iterative development and a user-centred design philosophy. Section 2.1.3 of the project proposal clearly states the problems and objectives we aim to solve and meet.

“For computer science students enrolled in the program offered via Coursera and the University of London who need help to limit distractions to their studies, StudySync is a web extension that will provide a single point of access to study productivity tools. The web extension will use a whitelist to block all web traffic except for the URLs specified. The system will combine this with time-tracking analytics to provide actionable feedback on the quality of the user’s study sessions. The time tracker will track students' time spent studying

during semesters in each class and each specific task within classes. This extension will increase students’

productivity and provide actionable statistics to help guide their study sessions. This enables students to

create better study habits and become better students. Unlike the current productivity web extensions on the

market, our product will integrate directly with the Coursera website, contain no paywall, and come with out-

of-the-box functionality to combine into a low barrier to entry Coursera productivity extension.”

This final report details the successful implementation of the project and highlights the intricacies our team navigated in completing the project. The reader will understand why our group made certain decisions and how the implementation came to be. Finally, the report will reflect on the overall process from a group and individual perspective. Developing new skills and overcoming setbacks has been challenging and rewarding. Our team is proud of what we built and looks forward to navigating more complex projects throughout the rest of our degree.

Throughout this report, we will heavily refer to the work presented in our project proposal and thus classify it as a dependency. Specific references to relevant sections will be made by referring to the section title instead of repeating ourselves for brevity.

## 1.2 Literature

Our project proposal went through an in-depth analysis of the current products on the market that could solve our project's problems and objectives described in the above vision statement. The market analysis found that none of the current products could match the needs of our target demographic and what they wanted from a tool of this nature. This validated StudySync had a place within the market. We recommend reviewing section 2.1.2 of our project proposal to understand the market picture before StudySync’s development.

This market analysis significantly aided in developing the functionality and design of StudySync. The minimalist nature of these web extensions and the simplistic functionality were critical design heuristics our team saw as valuable within this market. The images below show how this market analysis influenced the overall outcome of StudySync. The simple colour scheme, the visual distinction of call to actions, and the input/output forms were some of the many items influenced by our research.

// TODO IMAGES

Reviewing this section of our project proposal proved to be critical for the development of StudySync. The market analysis was completed effectively, which helped guide our designs, which our users widely accepted throughout the development process. In the evaluation section of the report, the SUS survey’s actively verified how our proposed design led to high usability satisfaction when functionality was added—a direct consequence of our market analysis.

Further literature review involved direct research regarding specific design patterns or functionality. This is better reserved for the planning and research section of the report, where sources and their influence will be discussed within their domain to show their effect on StudySync better.

## 1.3 Scope

Section 2.2 of the project proposal defined the project's scope. We defined significant features and limitations, scoped the development process via a Gantt chart, and explicitly defined the context in which the project was being built. Throughout the latter half of this class, this section was heavily referenced to ensure our team stayed within the scope and ultimately ensured the MVP met our stakeholder's expectations.

Within section 2.2.1 of the project proposal, we defined the significant features of our initial release. Through a robust testing system, we confirmed that the MVP produced by our team met the major features defined for release. The testing also demonstrated that the final MVP met our defined scope requirements.

We also defined a Gantt chart within this section of the report. The development half of this Gantt chart had to be revised to account for a change in the development process. We initially wanted to map sprints to user stories. However, we found that mapping a sprint to a system feature would allow the testing team to quickly verify the validity of the functionality implemented by the technical team. It also shortened our development time and allowed us some leeway in the sprints should one take a little longer than expected. While agile development usually focuses on user stories for sprints, we found this methodology worked for the team and led to a more streamlined development process.

The updated Gantt chart, which accurately displays the scope of the development process, is below. While there were some variations, the team largely followed this Gantt chart to completion. Please refer to section five for a deep dive into the complete development process.

A screenshot of a project

Description automatically generated

Figure 1: Gantt chart 1/2.

A screenshot of a calendar

Description automatically generated

Figure 2: Gantt chart 2/2.

## 1.4 Group Work

We split the working tasks for the latter half of the course. While there was much overlap within the project, we assigned major roles to individuals, which served as a guideline. Still, all four of us were crucial to implementing the MVP and the deliverable.

* Technical
  + Hashem was the technical lead. He oversaw the implementation of the web extension. He completed this role with his strong technical background and efficiently divided up roles related to the process. He was pivotal to the success of the project.
* Testing
  + Sunidhi and Mason were the lead quality assurance and user liaisons. They ensured the project stayed within scope and completed much of the research and user testing required to guide and validate it. They kept the stakeholders within the loop and efficiently guided the project toward a sound completion.
* Report
  + Gage managed the implementation of the report. He effectively divided roles and managed the report process throughout the latter half of the course.

# 2 Planning and Research

## 2.1 Research

Throughout the development process, many resources were used. Ranging from Google extension documentation to other BSc courses to discussion prompts. A high-level overview of these three resources follows.

### 2.1.1 Google Extension Documentation

Our project proposal identified Google Chrome as the sole supported web browser for StudySync in section 4.2.2. This was done in response to a survey that indicated that around 70% of our target demographic use Google Chrome as their web browser. To ensure StudySync met the standards set by Google, we heavily referred to the Chrome extension developer documentation: <https://developer.chrome.com/docs/extensions/develop>. No one in the group has built a web extension before, and this resource proved invaluable to guide our project implementation.

To highlight this, one can see how the whitelist storage was implemented. We knew storage would need to be used when researching the project, and Hashem found the Google Chrome Storage API referenced in section 4.6.2 in the project proposal. However, we did not anticipate it being difficult to get the API to store the extension's data effectively and consistently. These web docs enabled us to navigate this trouble efficiently.

A screenshot of a computer program

Description automatically generated

Figure 3: The documentation used to help guide storage implementation.

### 2.1.2 Software Design and Development

The software design and development course offered by the University of London was another great resource which helped the testing team navigate that process. The testing Excel sheet provided in week 14 of that course served as the basis for our complex testing set. It also directed us to the SUS testing procedure to speed up and efficiently test the usability metrics of the extension.

A screenshot of a computer

Description automatically generated

Figure 4: test procedure provided in CM2010.

### 2.1.3 Discussion Prompts

Lastly, our group completed the discussion prompts throughout the course during our weekly meetings. These discussion prompts were primarily discussed ad hoc at the start of our meetings to help get us into a proper mindset for that week’s work. For example, in discussion prompt 7.202, the importance of integration testing was discussed before we decided on a manner for testing StudySync. It helped guide us to relevant industry practices. It ultimately led us to choose the testing method provided in CM2010, focusing on ensuring the program met our functional requirements and verifying the validity of the MVP.

## 2.2 Planning and Iteration

In section 2.2.2 of the project proposal, we created a Gantt chart to guide us through the completion of this assignment. This Gantt chart made it extremely easy to compare our progress against the expectations we laid out for ourselves in the project proposal. We had to adjust it slightly after submitting the project proposal as we found that making weekly sprints dedicated to the features defined in section 4.3 of the project proposal was more precise and easier to manage rather than basing sprints on user stories. Every week, the three teams knew when they were expected to jump into the project and what they were expected to do. The technical team was given four days to implement a given system feature. The testing team was given one to two days to build a test set for that system feature and its completed functional requirements. Finally, a day was given for the team to meet live to reflect on the prior week and confirm the following plan.

This methodology worked exceptionally well for this project due to the work ethic of our team members. Hashem has industry experience and could produce efficient production-level code very quickly. While Sunidhi and Mason could promptly test and confirm the implementation, Hashem led to completion. Finally, Gage’s writing skills and project management oversight allowed the team to document and produce a quality final deliverable. In another setting, our plan would be too casual to lead to success. However, we feel the weekly meetings and effort put in by every teammate allowed us to follow a plan built for us that worked in our given circumstances. There were problems regarding prompt responses and confusion surrounding what was required at certain times. However, the simple nature of the system features allowed us to clear up confusion without limiting our end deliverable.

# 3 Prototyping and Iteration

## 3.1 Prototyping

Prototyping and evaluating user feedback was a critical aspect of our development process. Section 5 of our project proposal details the prototype and user feedback loop followed to produce the design and user interface. In brief, the team used market analysis to create low-fidelity prototypes that met our project objectives. The prototypes were then tested against direct user feedback to see the direction the users wanted us to follow. We then quickly iterated through prototypes to come to a final design, which largely stuck throughout the project's development.

During the iterative development cycle, we used SUS surveys to test our usability choices every other week. The SUS survey allowed us to test our design choices held up when functionality was implemented. We found that the users responded positively to the product as functionality was implemented. Therefore, little was needed on our end to change the product's design as our development process proceeded. The market analysis and initial user involvement paid dividends as the project progressed. This allowed our technical team to focus on implementing our project solutions rather than adjust the design on the fly.

## 3.2 Iteration

Most of this half of the semester was spent in iterative development cycles loosely related to an agile development process. Our technical team was given a system feature to implement at the start of every week. The system features are in section 4.3 of the project proposal.

The technical team was then given four days to implement this functionality in our codebase. Hashem was the technical team lead, and he effectively divvied up roles and managed the GIT repository in which the code base was stored. After the technical team implemented the functionality of the given week’s system feature, the testing team wrote a testing suite which directly mapped to the functional requirements covered by that particular system feature. The testing suite was then run to confirm all functionality was successfully implemented. The group then met every Saturday to discuss the past week and the overall success or failure of the sprint.

While this iterative process was ongoing, the report team documented all methods used by the team in a coherent final report that touched on the successes of the individual weeks and the project as a whole. For a picture of our work, our GIT logs and code base can be found in response 2 of this submission.

A screenshot of a black and white text

Description automatically generated

Figure 5: Defining roles after project proposal submission.

Our team worked in a tight iterative loop based on a weekly approach. Due to the simplicity of the system features and in-depth research completed in the project proposal, we could quickly iterate through functional development and deliver what we set out to build.

# 4 Design

## 4.1 User Interface

The design of our web extension was largely sorted within the project proposal. We followed a prototype loop in which user feedback was elicited to adjust the final design incrementally. The technical team built a high-fidelity prototype, the last major step in our midterm response. This prototype was built using JavaScript and HTML and would serve as the basis for the start of the iterative development phase. For a deep dive into the prototyping process and how the team developed the design, please refer to section 5 of the project proposal.

## 4.2 Software Requirements Specification

Within the project proposal in section 4, we outlined the SRS to be used to guide the development of the web extension. This section does not need to be re-detailed and is better viewed within the context of the project proposal. However, we want to reflect on how this section guided the development process. In the following section, the development process will be broken down every week, and the marker will see that the system features and functional requirements defined in section 4 of the project proposal were referenced and used regularly throughout the latter half of this course. The SRS was the most critical work we produced in the project proposal. There were some minor errors in the SRS, but overall, the quality of this section is why we completed this project successfully.

We combined the SRS with a quasi-agile development process that fit our busy schedules. This process is broken down in detail in the following section.

Through the SRS, we could also pinpoint the browser interfaces and the tech stack needed to develop the product. This involved JavaScript, HTML, CSS, and some API usage provided by the Chrome browser. Seeing that the team comprised all level five students, we felt confident about this tech stack. Integrating the web extensions with the manifest standard and Chrome API proved a nice challenge in extending our skill set.

// TODO insert image of language percent on Git Hub.

In summary, the design of our web extension was primarily completed within the project proposal. Some changes were made through the iterative development process, which are discussed in the following section. The initial design work we did in the project proposal led to the project's success.

# 5 System Development

## 5.1 Development Process

This section will cover the development process on a week-by-week basis. We followed an ad hoc agile development process based on system features defined in the project proposal. The development process followed a seven-week plan. Each week will be broken down below.

### 5.1.1 Week One

The technical team received SF-1, “Interact with the Extension via the Dropdown Menu,” from the project proposal to start the week, which can be found in section 4.3.1 of the project proposal. This feature was meant to begin the development process by getting the dropdown menu to operate.

This week’s implementation was primarily focused on JavaScript event handlers based on buttons and displaying data via the dropdown menu. It took a little longer than expected, as many state variables and trackers had to be initialized to ensure the dropdown menu affected the overall state of the program. This being the first sprint, some functional requirements could not be fully completed due to the reliance on a fully functional time-tracker/whitelist being ready to receive the data. However, the technical team got it to a point where it would be easy to integrate the captured data once implemented during their respective sprint.

A screenshot of a web page

Description automatically generated

Figure 6: StudySync dropdown menu.

The technical team caught two issues with these functional requirements. The first is that the dropdown did not need an add-to-whitelist button. The system automatically blocks and prompts the user when navigating to a blocked website. Therefore, it would be redundant to add a button in the dropdown menu that would never be reached. Secondly, the users wanted a simple dropdown menu, and the technical team felt a single settings button would better meet their needs. Therefore, the functional requirements were refactored to the below.

|  |  |
| --- | --- |
| Dropdown: | **Dropdown general functionality** |
| .SettingsButton | StudySync shall direct the user to the settings pages of the extension. |

|  |  |
| --- | --- |
| Dropdown.Whitelist: | **Dropdown whitelist functionality** |
| .Toggle: | StudySync shall toggle the whitelist between on and off when the user clicks the toggle button. |
| .QuickDelete: | StudySync shall delete a whitelist item if it’s corresponding trashcan button is clicked on by the user. |

|  |  |
| --- | --- |
| Dropdown.TimeTracker: | **Dropdown time-tracker functionality** |
| .Display: | StudySync shall display the current course and task being tracked along with the total time spent on the current task. |
| .StatsButton: | StudySync shall redirect the user to the data dashboard HTML page when the user clicks this button. |

The below image captures some of the functionality of the dropdown menu that was completed at this stage. Chrome APIs were introduced to keep an eye on the tabs opened by the user, and the settings button was implemented to redirect the user to the extensions settings page. Lastly, the whitelist logic was implemented to allow for easy connection to the whitelist.

A screen shot of a computer program

Description automatically generated

Figure 7: JavaScript snippet of the dropdown menu.

Once the technical team felt they had implemented the functionality they thought was possible, the testing team created a testing suite which directly mapped to this system feature. It initially failed during the first testing due to the whitelist toggle and time-tracker not being able to integrate with a dropdown menu yet. We had about a 40% initial passing rate, which was not ideal for our first sprint. The complete test set can be found in the dependencies folder under the Excel sheet labelled 4.3.1\_test\_set.xlsx.

With this week essentially over, we reflected on the week and pointed out the positives and negatives. The most significant positive was returning to efficient work after a difficult midterm season. The most critical negative was a failing test set. We saw this as the growing pains of switching to system features instead of user stories. However, moving into the following week, we were on the correct path. This was also an excellent educational week, as we learned that sprints are hard to define in a localized manner. Adding to the system often requires understanding the addition's relationship with the preexisting modules. Our initial plan did not account for this.

### 5.1.2 Week Two

This week was initially meant to focus on the navbar between the settings pages. However, our technical team had already implemented this functionality for our high-fidelity prototype. Therefore, the implementation phase of this week was already complete. Consequently, we wanted to take advantage of this oversight, speed through this week's testing, and create a SUS survey for our users to review for this and the last system feature. This way, we would have time should future development issues crop up.

To quickly paint a picture of how our navigation bar came to be, Hashem referenced and took inspiration from similar web extensions to see how they managed their navbars. We also used the prototype survey feedback to create the navbar below. Which is vertical and has clear visual queues and clear titles for what each link leads to. The design is heavily documented in the project proposal.

A screenshot of a data dashboard

Description automatically generated

Figure 8: StudySync Navbar.

The code was already completed during our project proposal phase. It was done as the navbar we felt was part of the overall design, and we wanted to ensure the high-fidelity prototype had all the navigation features needed to get a rough look at the design. The code involved is primarily HTML and CSS for the navbar. The HTML is below, but please refer to the code dependencies for a deeper dive.

A screen shot of a computer program

Description automatically generated

Figure 9: HTML code for the navbar.

With the navbar completed heading into this week, the testing team was able to focus on creating a test set for this system feature. Due to the low functional requirements defined by this system feature, the test set resulted in a 100% passing rate. Please refer to the Excel file 4.3.2\_test\_set.xlsx in the dependencies for an overview of this test set.

The testing team could now focus on the SUS test, which is below. This was created to focus on the functionality developed in the dropdown menu from last week in conjunction with the navbar, which was the focus of this week. The SUS survey returned positive results when functionality was added to our initial design. Thus, we knew there wouldn’t be much point in further testing the design and functionality.

// TODO SUS SURVEY #1

Lastly, the team decided to skip a meeting for the work this week and move on to the work for week three immediately. We wanted to take advantage of this oversight in the project plan as there were likely oversights that would lead to negative consequences later. We pushed talking about the SUS results to the meeting next week.

### 5.1.3 Week Three

Moving on from last week, our team jumped into this week’s work on Tuesday; our team was to implement system feature 4.3.3, “Edit the whitelist form.” In summary, it was a week focused mainly on getting the logic of the whitelist implemented to a capacity that allowed the user to interact with it.

The technical team had a busy week as the groundwork they laid was to be the foundation for the whitelist as a whole. They needed to implement the whitelist storage first, which was to be implemented using the Chrome storage API. The project proposal did not account for this part of the whitelist implementation, as functional requirements for storage were not explicitly defined. Therefore, the report team created the below functional requirement to be completed in addition to the Whitelist.Form requirements.

|  |  |
| --- | --- |
| Whitelist.Storage | **Whitelist persistent storage** |
| .StoreViaAPI | StudySync shall store the user-defined whitelist in persistent storage with the Chrome storage API. |

With this requirement confirmed, the technical team began work on implementing persistent storage for the whitelist. They encountered many hurdles and repeatedly referred to the Google developer documentation. However, in the end, they created a well-separated and secure storage location for the whitelist. The code for the storage can be found in core.js. Many functions access the chrome storage API and refer to it. An example of the getAllowList function is below, which shows an access attempt to the storage location. The API straightforwardly abstracts away storage details into easy-to-digest functions. This allows the extension to easily store, update and retrieve data related to the extension and also blocks access to the data from outside sources, which meets the SEC-1 requirement defined in the project proposal.

A screen shot of a computer code

Description automatically generated

Figure 10: getAllowList function.

The code below is a snippet from the core.js file, which implements much of this system features functionality. The renderAllowList function displays the stored whitelist for updating to the user. It utilizes helper functions that retrieve the stored whitelist and populate the HTML page with the information. Finally, it creates a function attached to the “Add to whitelist” button. This function allows the user to manually add items to the whitelist from this page. It first retrieves the user input and stores it in the persistent storage.

A screen shot of a computer program

Description automatically generated

Figure 11: renderAllowList function.

The renderAllowList function allowed the user to view what was already on the whitelist. However, the handleAddToAllowList function enables users to add websites to the whitelist. The function mainly contains validation code to confirm that the URL the user has entered is valid. If so, it processes the input and ensures it’s in a standardized format. Otherwise, it displays an error and gracefully recovers from the wrong input.

A computer screen shot of a program code

Description automatically generated

Figure 12: handleAllowList function.

The technical team had a big week but completed it within our newly defined deadlines. It was now on the testing team to validate the functionality against the functional requirements. With the addition of the new persistent storage requirement, our testing set contained four unique tests. All of whom passed on the first try. Please refer to the Excel file 4.3.3\_test\_set.xlsx to look into our testing schema for this system feature.

Lastly, our team met on Saturday to discuss the prior two weeks. Most importantly, the results of the SUS survey. The results here confirmed that adding functionality to our design maintained an overall satisfaction level in terms of usability, and we felt confident this would be a continuous trend. Our planning in the midterm submission paid off as we started to complete the functional requirements for the system. We were also ahead of schedule, and overall morale was relatively high after finishing this week's work. We decided to end the meeting with a discussion of discussion prompt 7.203, in which we discussed testing practices and how ours fit in with our development plan. We felt confident that our testing scheme validated that the system's functionality met the functional requirements.

### 5.1.4 Week Four

This week, we were to implement system feature 4.3.4, “Whitelist all attempted URL queries.” The team was excited to start this week, which involved finishing up the whitelist portion of the extension. The technical team was given the functional requirements, and their goal was to finish implementing them within four days.

The technical team started by creating a listener attached to the tabs the user opened. This listener can be found below. It retrieves the extension data from the storage and performs checks to confirm the visited URL is allowed. It first checks to ensure the limiter is enabled, and if so, it tries to parse the URLs visited by all tabs. It runs through these URLs, confirming whether the user can access them. It allows the user to go through or send a message to the tab, forcing the showBlocker function within the content.js file to be called. This listener is the main logic behind verifying whether the URL can be visited.

A screen shot of a computer screen

Description automatically generated

Figure 13: Tab listener.

After this, the technical team implemented the showBlocker function, the extension's limiting functionality. The team needed to ensure some block was implemented, refocusing the user on their studies. It would first block the user from interacting with the site and then prompt the user to either add the site to the whitelist or cancel and close the tab. The blocking pop-up is below. If the user decides to allow the website, then the domain will be added to the whitelist, and the page will be refreshed to allow access.

A screenshot of a computer

Description automatically generated

Figure 14: Blocking popup.

The code for the showBlocker function is below. It implements the above functionality by interacting with the DOM and the storage API. The blocker is a div added to the Dom, which is adjusted via the innerHTML property to show the visual look depicted above. It then creates two event listeners, which handle the buttons by adding the URL to the whitelist or closing the current tab.

A screen shot of a computer program

Description automatically generated

Figure 15: showBlocker function.

Through their perseverance, the technical team navigated this week's difficulties and completed the implementation for this week’s system feature on Thursday; with exactly four days spent in development, we were still ahead of schedule. Our testing team then got the testing suite for this system feature built out and validated. The testing suite passed alongside all prior failing whitelist functionality (test set 4.3.1 was revisited), confirming the whitelist was officially complete. Please refer to the Excel file 4.3.4\_test\_set.xlsx for a full rundown of the testing scheme. Overall, the team was very excited about where we were and wanted to get user feedback based on an SUS survey for the whole whitelist functionality.

// TODO SUS SURVEY FOR WHITELIST

After the SUS survey, the team met and reflected on the prior week. We were ahead of schedule and were in a good place heading into implementing the time tracker functionality. Therefore, the meeting ran short as we wanted to give ourselves some downtime before developing the time tracker.

### 5.1.5 Week Five

This week, our team was to complete the implementation of system feature 4.3.5, “Track Coursera study time.” In summary, it was meant to be the implementation week for the time-tracker functionality of the extension.

Like last week, the development plan did not account for the difficulty of implementing the time tracker functionality. Unlike last week, we could not recover gracefully from this oversight. The technical team needed ten days to complete the implementation of the requirements for this system feature. Thus, we were dealing with a time crunch for the final two weeks. That time crunch will be discussed in detail in the affected weeks.

// TODO BREAK DOWN CODE FOR THIS WEEK

After the development team successfully implemented the functional requirements, it was Wednesday the following week.

# 6 Analysis

## 6.1 Functional Analysis

In the project proposal, we defined an SRS, and this describes the functional components the system needs to have to fit our problem space. These functional components guided the development of our software project. The team stayed within the scope defined in this section to ensure the project's successful development. After the development of each system feature, a test set was created to validate the implemented solution meets the functional requirements laid out in the SRS. The system was validated through this rigorous testing set to meet the stakeholders' functional requirements expected of the MVP.

## 6.2 Technical Analysis

While the project met the functional requirements, the technical implementation of these requirements fell under-successful. Our development process did not follow strict implementation guidelines. No test-driven development was implemented, and similar development practices were not used. We can’t be reasonably sure that our code is bug-free from an implementation standpoint as we don’t have a testing set against which to run it. This is an unfortunate oversight in the development and planning processes, and this would not be passable in a real-world setting.

Should the extension continue development, the team should first develop a unit test set and refactor the code to remove potential module cohesion and coupling oversights. For example, it’s pretty challenging to determine what is happening in the below function. HTML code is mixed in with JavaScript, and it’s clear that many different tasks are being completed. Thus, the relationship of tasks within this function is not as strongly related as we would have preferred.

A screen shot of a computer program

Description automatically generated

Figure 16: Poorly related function implementation.

Therefore, the technical implementation could use some work to validate that the technical implementation is stable and reliable for our product.

## 6.3 Security Analysis

We identified some simple security requirements within the project proposal, as seen in section 4.7.3. We defined that only the web browser user could view the data stored by the extension and that no other entity would be able to view it.

# 7 Evaluation

# 8 Conclusion

# 9 Individual Reflection (Name)

# 10 Appendix

# 11 References