Final Deliverable

StudySync Web Extension

Sunidhi Abhange (SN: 220302962)

Gage Fleming (SN: 220103602)

Hashem Ramadan (SN: 220179685)

Mason Waldapfel (SN: 220111098)

University of London

CM2020: Agile Software Projects

Team 68 (Tutor Group 6)

Table of Contents

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.

“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 vision statement can be broken down into the aims and objectives below.

- Aims
    - Limit web-based distractions to UofL students.
    - Provide a single base point for many study-based tools.
    - Increase student productivity by creating better study habits.

    - Provide actionable statistics to help guide study sessions.

- Objectives
    - Provide Courser-integrated productivity tools that enhance study sessions
    - Remove all website distractions from Coursera study sessions.
    - Effectively track 80% of study time for tasks related to courses accessed via Coursera.

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 and Market Analysis

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 an example, such as the extension Web Blocker, 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.

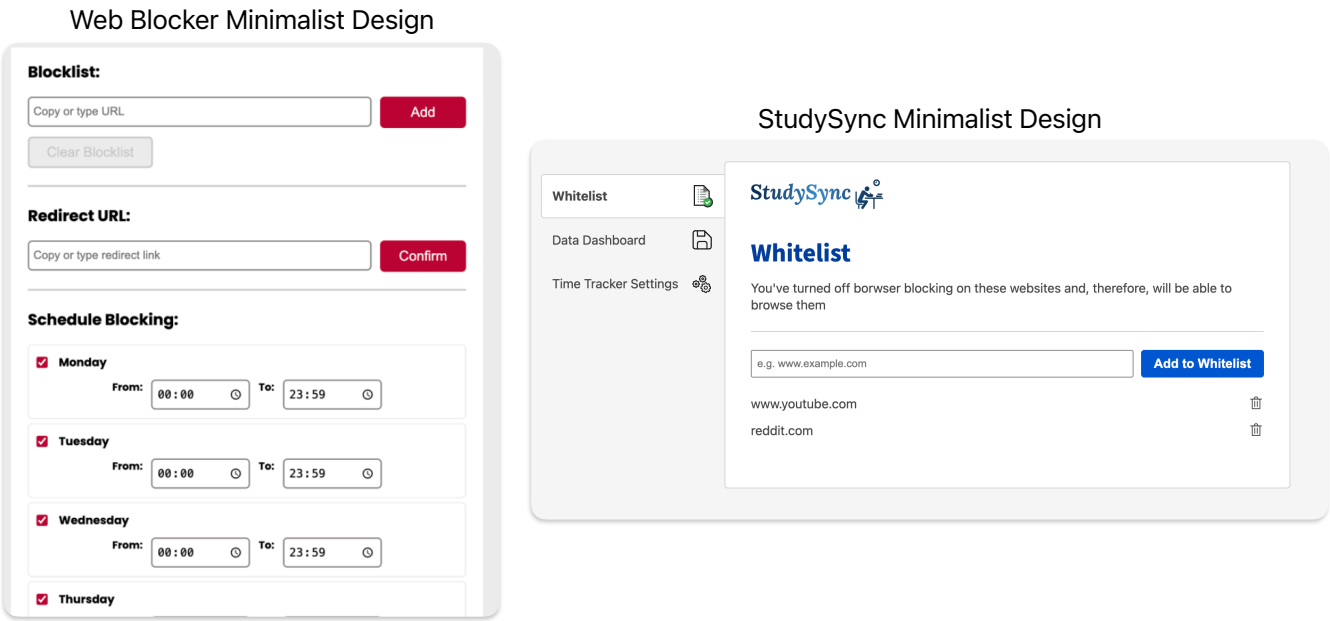


Figure 1: Comparison between Web Blocker and StudySync Designs.

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 section 5 of this report, one can see that SUS survey's 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 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.

Below is the updated Gantt chart, which accurately displays the scope of the development process. 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.

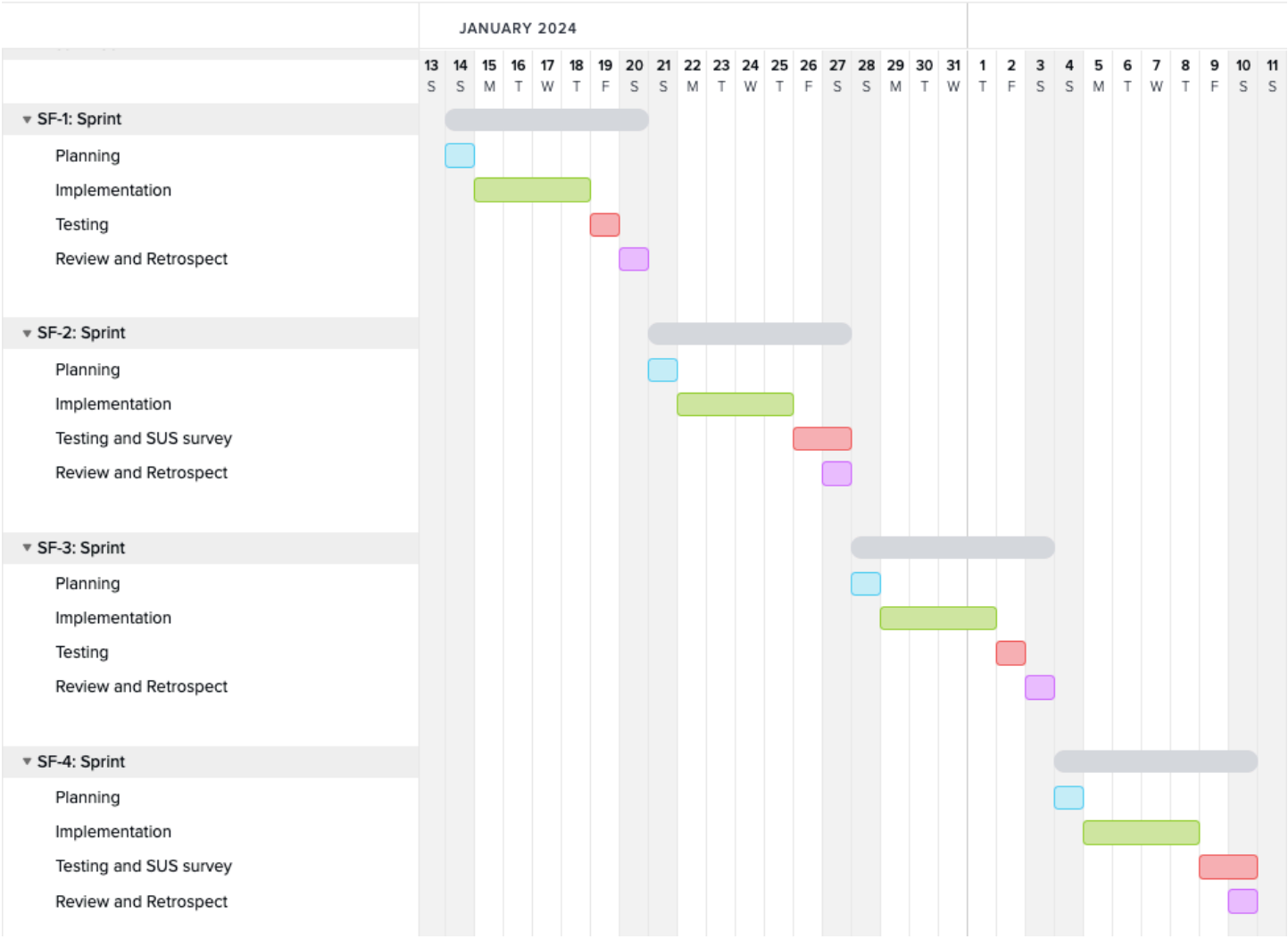


Figure 2: Gantt chart 1/2.

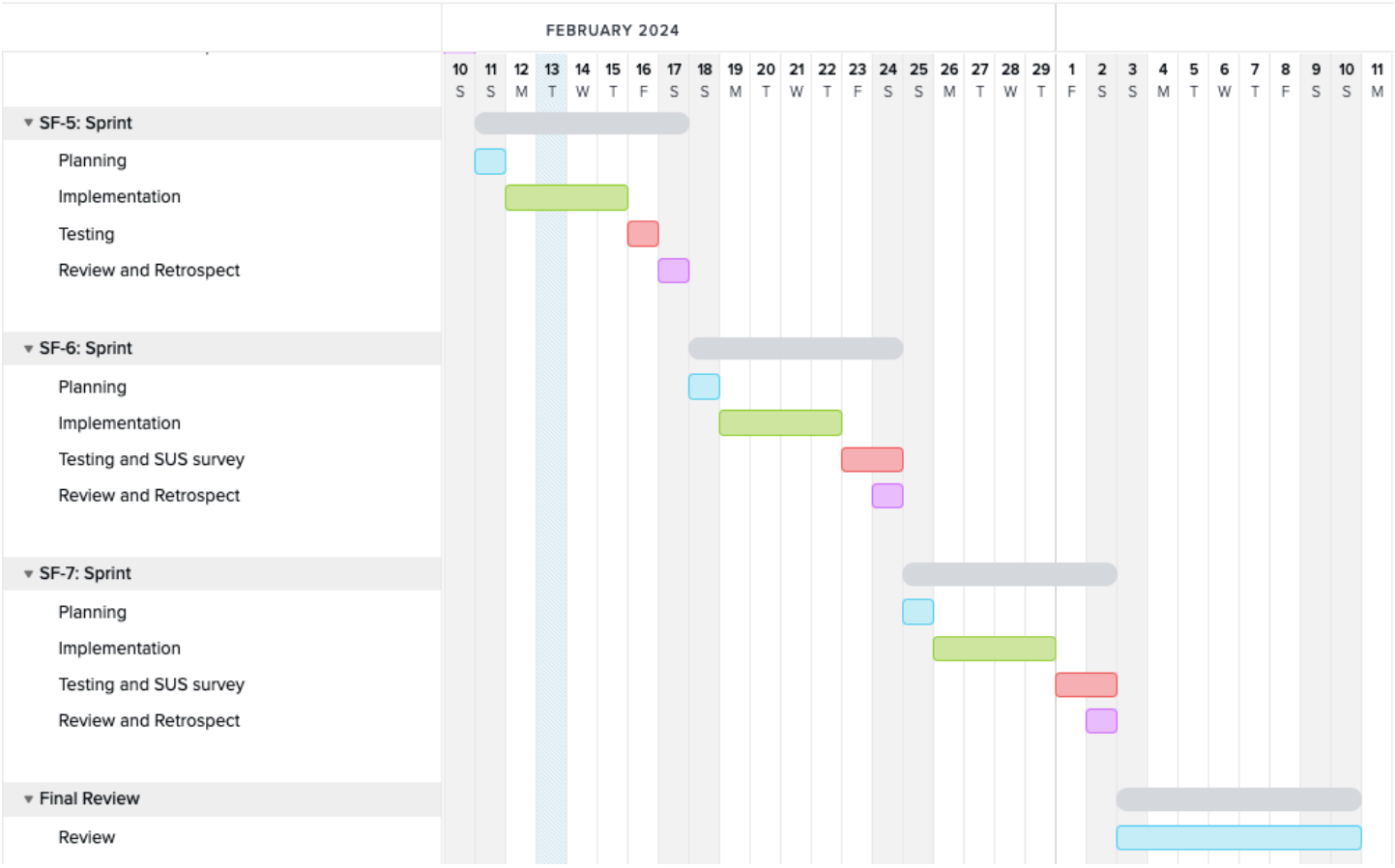


Figure 3: Gantt chart 2/2.

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.
- 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. This was done by staying in the loop with each team and documenting each process as the work progressed.

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 Google's standards, 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.

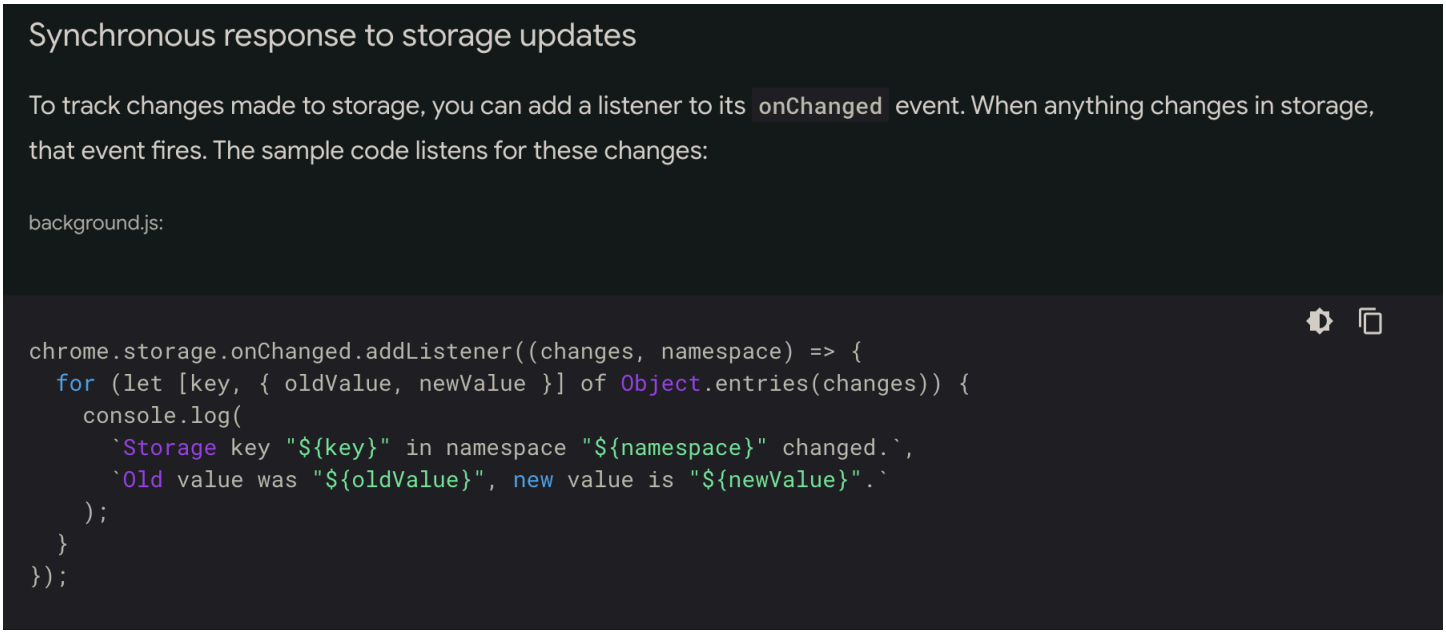


Figure 4: The documentation used to help guide storage implementation.

2.1.2 Software Design and Development

The software design and development course offered by the University of London was another great resource that helped the testing team navigate this 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 extension's usability metrics.

This is based on p106 of "ISO/IEC/IEEE International Standard - Software and systems engineering – Software testing –Part 3: Test documentation," in ISO/IEC/IEEE 29119-3:2013(E) , vol., no., pp.1-138, 1 Sept. 2013, doi: 10.1109/IEEESTD.2013.6588540.

https://ieeexplore.ieee.org/document/6588540

Test Procedure ID	Objective and Priority			Estimated Duration:
123	To test the behaviour of the password entry box under			10 minutes
Startup: Open browser, go to domain.com/login page				
Relationship to other procedures: none				
Test log				
Date	Initials	Test item		Ok/ not ok
Comments:				
Procedure				
Step and test case	Activities	Examination of result	Actual result	Test result
1.1	Type in 'test' to the password box	Check that four stars appear in the password box	Four stars appeared	Pass
Stop and wrap up. Clear browser data and close window				

Figure 5: 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 7.203, 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 CM2020, focusing on ensuring the program met our functional requirements and verifying the validity of the MVP.

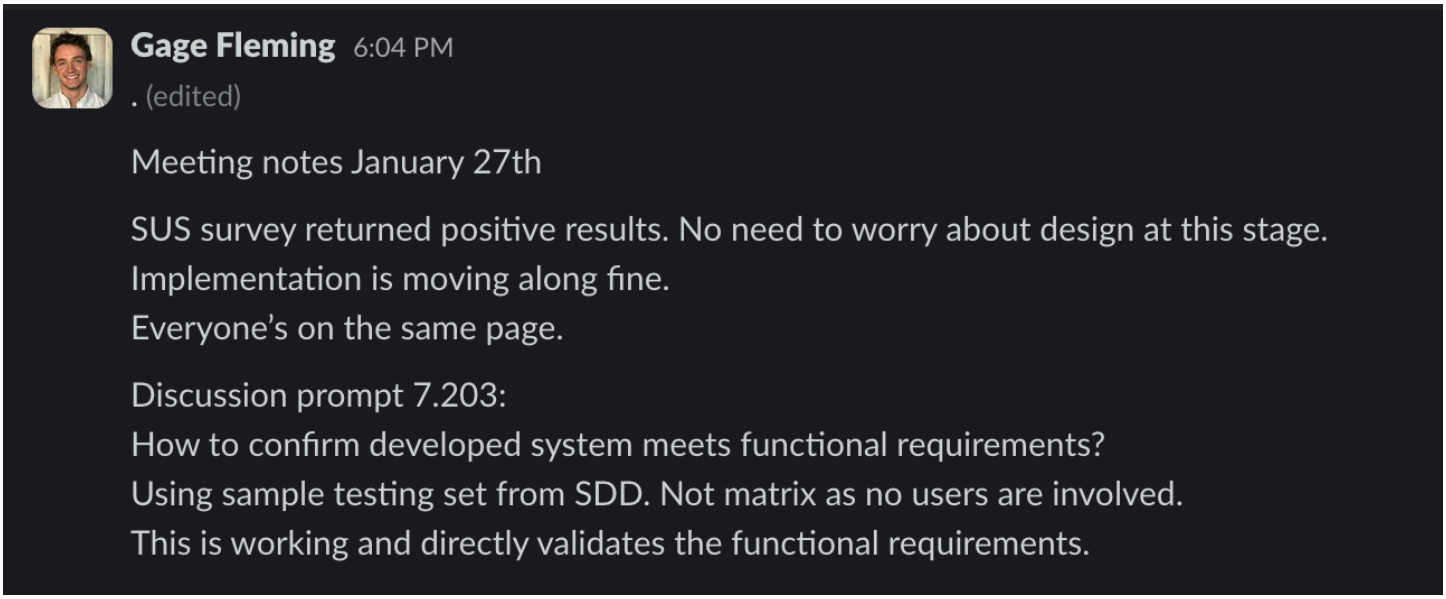


Figure 6: Brief meeting notes from the January 27th meeting.

2.2 Planning and Iteration

The revised Gantt chart above laid out the broad development plan for the project. 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. A SUS survey would also be completed every other week to verify that adding functionality to our current design yielded positive results.

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. At the same time, Sunidhi and Mason could promptly test and confirm the implementation. 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. For a deep dive into what the process looked like and the nuances, please refer to section 5.

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.

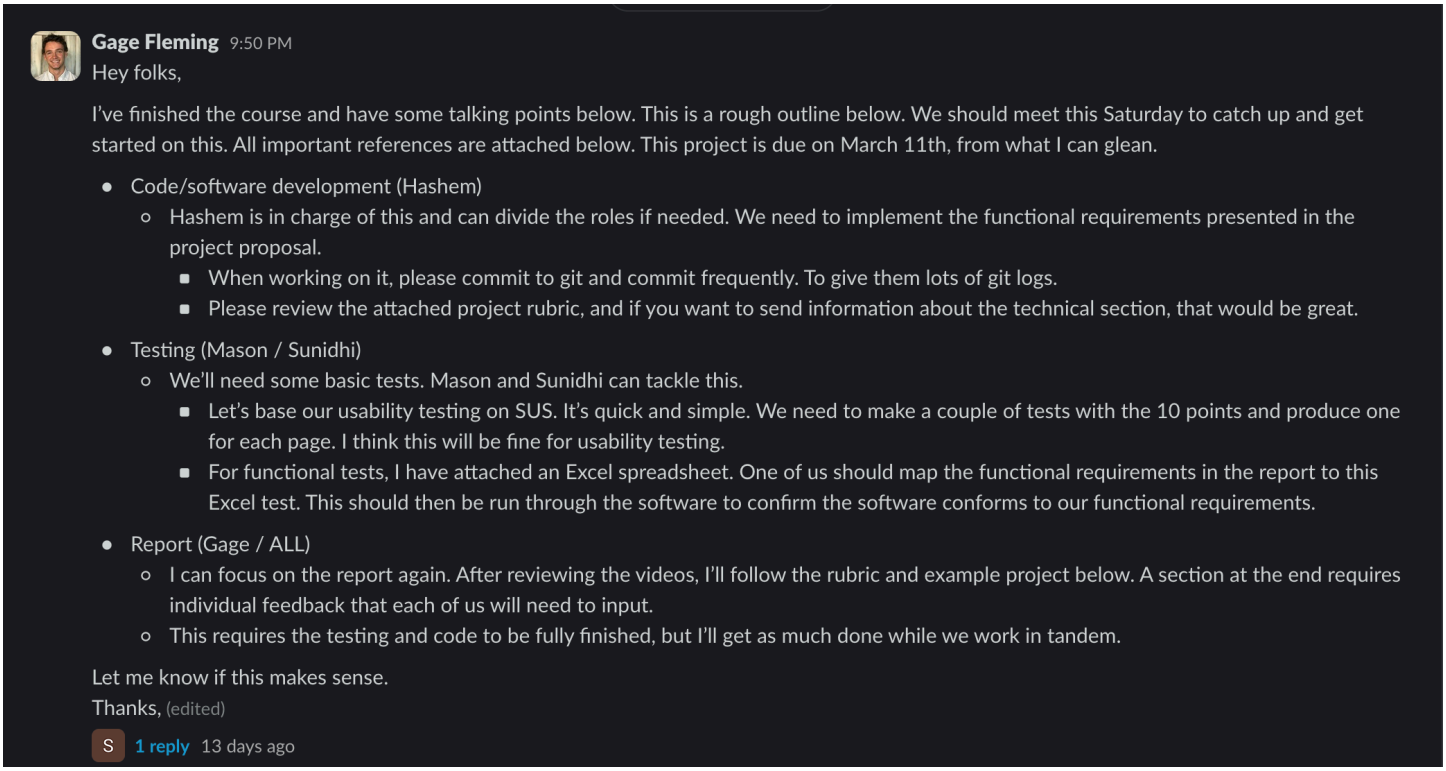
During the iterative development cycle, we used SUS surveys to test our usability choices every other week. The SUS survey allowed us to test our design choices held up when functionality was implemented. We found that the users responded positively to the product as functionality was implemented. Therefore, 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 functionality was added to our high-fidelity prototype. 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-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.



Gage Fleming 9:50 PM

Hey folks,

I've finished the course and have some talking points below. This is a rough outline below. We should meet this Saturday to catch up and get started on this. All important references are attached below. This project is due on March 11th, from what I can glean.

- Code/software development (Hashem)
  - Hashem is in charge of this and can divide the roles if needed. We need to implement the functional requirements presented in the project proposal.
    - When working on it, please commit to git and commit frequently. To give them lots of git logs.
    - Please review the attached project rubric, and if you want to send information about the technical section, that would be great.
- Testing (Mason / Sunidhi)
  - We'll need some basic tests. Mason and Sunidhi can tackle this.
    - Let's base our usability testing on SUS. It's quick and simple. We need to make a couple of tests with the 10 points and produce one for each page. I think this will be fine for usability testing.
    - For functional tests, I have attached an Excel spreadsheet. One of us should map the functional requirements in the report to this Excel test. This should then be run through the software to confirm the software conforms to our functional requirements.
- Report (Gage / ALL)
  - I can focus on the report again. After reviewing the videos, I'll follow the rubric and example project below. A section at the end requires individual feedback that each of us will need to input.
  - This requires the testing and code to be fully finished, but I'll get as much done while we work in tandem.

Let me know if this makes sense.

Thanks, (edited)

S 1 reply 13 days ago

Figure 7: 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. Section 5 provides a deeper dive into the iterative nature of this project.

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.

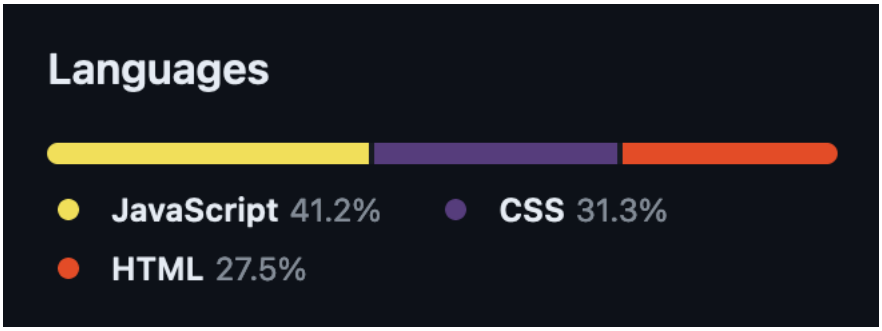


Figure 8: Languages used in extension.

In summary, the design of our web extension was primarily completed within the project proposal. However, some changes were made through the iterative development process, which is discussed in the following section.

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 system feature 4.3.1, “Interact with the Extension via the Dropdown Menu,” from the project proposal to start the week. 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.

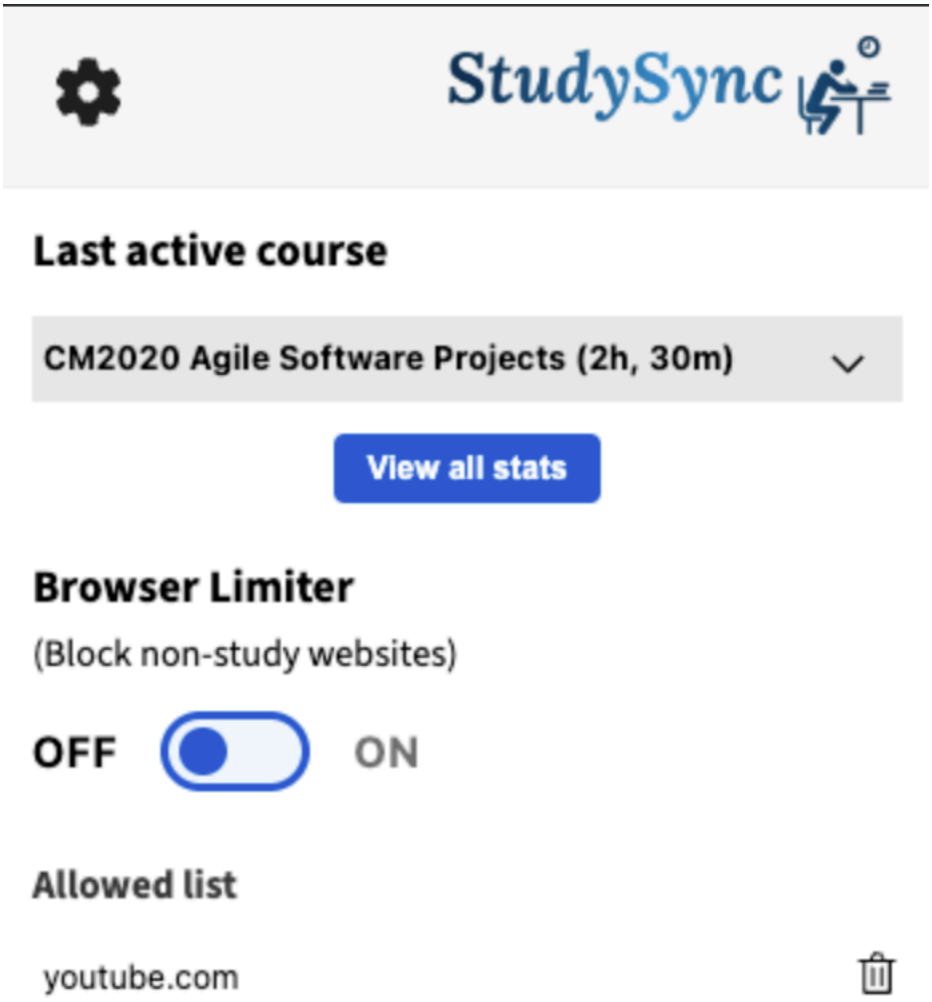


Figure 9: StudySync dropdown menu.

The technical team started by catching two issues with the 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.

```
const tabs = await chrome.tabs.query({ currentWindow: true });

// Sort tabs according to their index in the window.
tabs.sort((a, b) => a.index - b.index);

const optionsBtn = document.querySelectorAll('[data-action="go-to-options"]');
optionsBtn.forEach((btn) => {
  btn.addEventListener('click', function () {
    if (chrome.runtime.openOptionsPage) {
      chrome.runtime.openOptionsPage();
    } else {
      window.open(chrome.runtime.getURL('options.html'));
    }
  });
});

const allowList = await chrome.storage.sync.get('browserLimiterAllowList', (data) => {
  console.log('allowList: data', data);
  return data;
});
console.log('allowList', allowList);
```

Figure 10: 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.

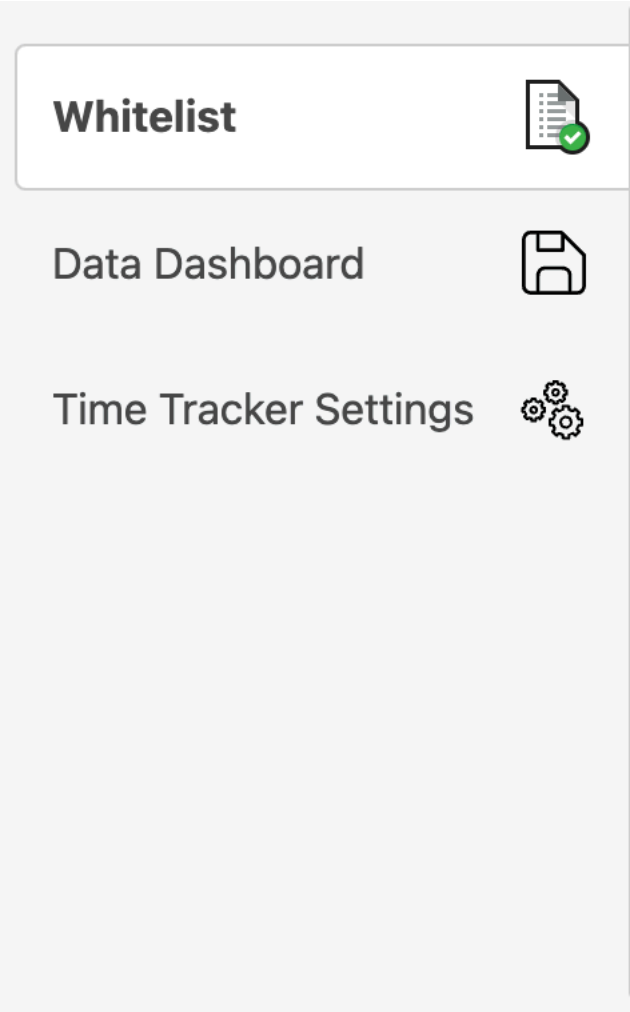


Figure 11: StudySync Navbar.

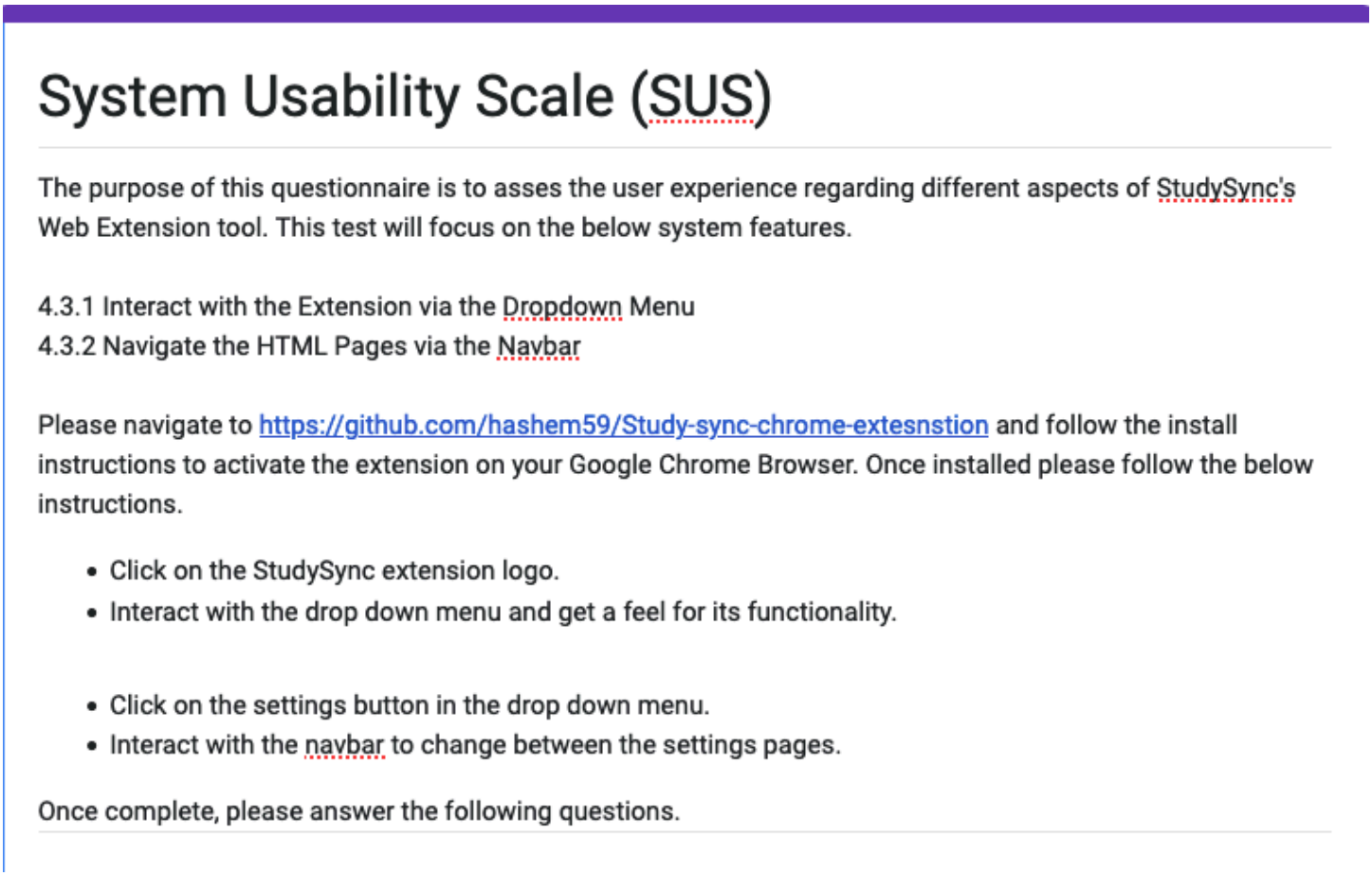
The code was already completed during our project proposal phase. It was done as the navbar we felt was part of the overall design, and we wanted to ensure the high-fidelity prototype had all the navigation features needed to get a rough look at the design. The code involved is primarily HTML and CSS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

```
58      <side-nav>
59        <nav>
60          <ul>
61            <li>
62              <a
63                id="tab-Whitelist"
64                href="#content-Whitelist"
65                role="tab"
66                aria-selected="true"
67                aria-controls="content-Whitelist"
68                >Whitelist</a
69              >
70            </li>
71            <li>
72              <a
73                id="tab-DataDashboard"
74                href="#content-DataDashboard"
75                role="tab"
76                aria-controls="content-DataDashboard"
77                >Data Dashboard</a
78              >
79            </li>
80            <li>
81              <a
82                id="tab-TimeTrackerSettings"
83                href="#content-TimeTrackerSettings"
84                role="tab"
85                aria-controls="content-TimeTrackerSettings"
86                >Time Tracker Settings</a
87              >
88            </li>
89          </ul>
```

Figure 12: 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. For complete results, please refer to Appendix Section 10.1.



System Usability Scale (SUS)

The purpose of this questionnaire is to asses the user experience regarding different aspects of StudySync's Web Extension tool. This test will focus on the below system features.

4.3.1 Interact with the Extension via the Dropdown Menu
4.3.2 Navigate the HTML Pages via the Navbar

Please navigate to https://github.com/hashem59/Study-sync-chrome-extesnstion and follow the install instructions to activate the extension on your Google Chrome Browser. Once installed please follow the below instructions.

- Click on the StudySync extension logo.
- Interact with the drop down menu and get a feel for its functionality.

- Click on the settings button in the drop down menu.
- Interact with the navbar to change between the settings pages.

Once complete, please answer the following questions.

Figure 13: SUS Survey One Description.

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.

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.

```
16  function getAllowList() {
17    return new Promise((resolve, reject) => {
18      chrome.storage.sync.get(['browserLimiterAllowList'], (data) => {
19        resolve(data.browserLimiterAllowList);
20      });
21    });
22  }
```

Figure 14: 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.

```
async function renderAllowList() {
  const allowList = (await getAllowList()) || [];
  const allowListContainer = document.querySelector('ul.allow-list');
  if (!allowListContainer) return;
  allowListContainer.innerHTML = '';
  if (allowList.length == 0) return;
  let html = '';
  allowList.forEach((domain) => {
    const li = `
      <li class="allow-list__item">
        <div>${domain}</div>
        <button class="button" data-url="${domain}" type="button" title="Remove">
          <svg
            aria-hidden="true"
            focusable="false"
            class="icon icon-remove"
            viewBox="0 0 18 17"
          >
            <use href="#icon-remove"></use>
          </svg>
        </button>
      </li>
    `;
    html += li;
  });
  allowListContainer.innerHTML = html;
  allowListContainer.querySelectorAll('button').forEach((btn) => {
    btn.addEventListener('click', async function (e) {
      const domain = e.currentTarget.getAttribute('data-url');
      const newAllowList = allowList.filter((item) => item !== domain);
      await chrome.storage.sync
        .set({ browserLimiterAllowList: newAllowList })
        .then(() => renderAllowList());
    });
  });
}
```

Figure 15: 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.

```
61  async function handleAddToAllowList() {
62    const form = document.querySelector('.add-to-whtelist-form');
63    if (!form) return;
64    form.addEventListener('submit', async function (e) {
65      e.preventDefault();
66      console.log('submit', e.currentTarget);
67      let domain = e.currentTarget.querySelector('input').value;
68      // if domain does not has https:// or http://, add it
69      if (!domain.includes('http')) domain = `https://${domain}`;
70
71      try {
72        // check if domain is a valid domain
73        const url = new URL(domain);
74        domain = url.hostname;
75        if (domain) {
76          const allowList = (await getAllowList()) || [];
77          allowList.push(domain);
78          await chrome.storage.sync.set({ browserLimiterAllowList: allowList });
79          renderAllowList();
80          e.target.querySelector('input').value = '';
81        } else {
82          showDomainError(form);
83        }
84      } catch (error) {
85        showDomainError(form);
86      }
87    });
88  }
```

Figure 16: 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 held the weekly Saturday meeting. It was clear our plan was currently leading to success. We were a week ahead of schedule to allow for potential oversights, and the users responded well to the technical implementation. 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.

```
chrome.tabs.onUpdated.addListener((tabId, info, tab) => {
  if (tab.url) {
    chrome.storage.sync.get
    chrome.storage.sync.get(['browserLimiterAllowList', 'browserLimiterState']).then(async (data) => {
      const browserLimiterState = data.browserLimiterState;
      try {
        const allowList = data.browserLimiterAllowList || [];
        const url = new URL(tab.url);
        const domain = url.hostname;
        if (tab.url && tab.url.includes('coursera.org')) {
          const queryParameters = tab.url.split('?')[1];
          const urlParameters = new URLSearchParams(queryParameters);
        } else if (!allowList.includes(domain) && browserLimiterState) {
          const response = await chrome.tabs.sendMessage(tabId, {
            type: 'StopTab',
            domain: domain,
          });
        }
      } catch (error) {
        console.warn('error', error);
      }
    });
  } else {
    console.log('tab.url is not defined');
  }
});
```

Figure 17: 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.

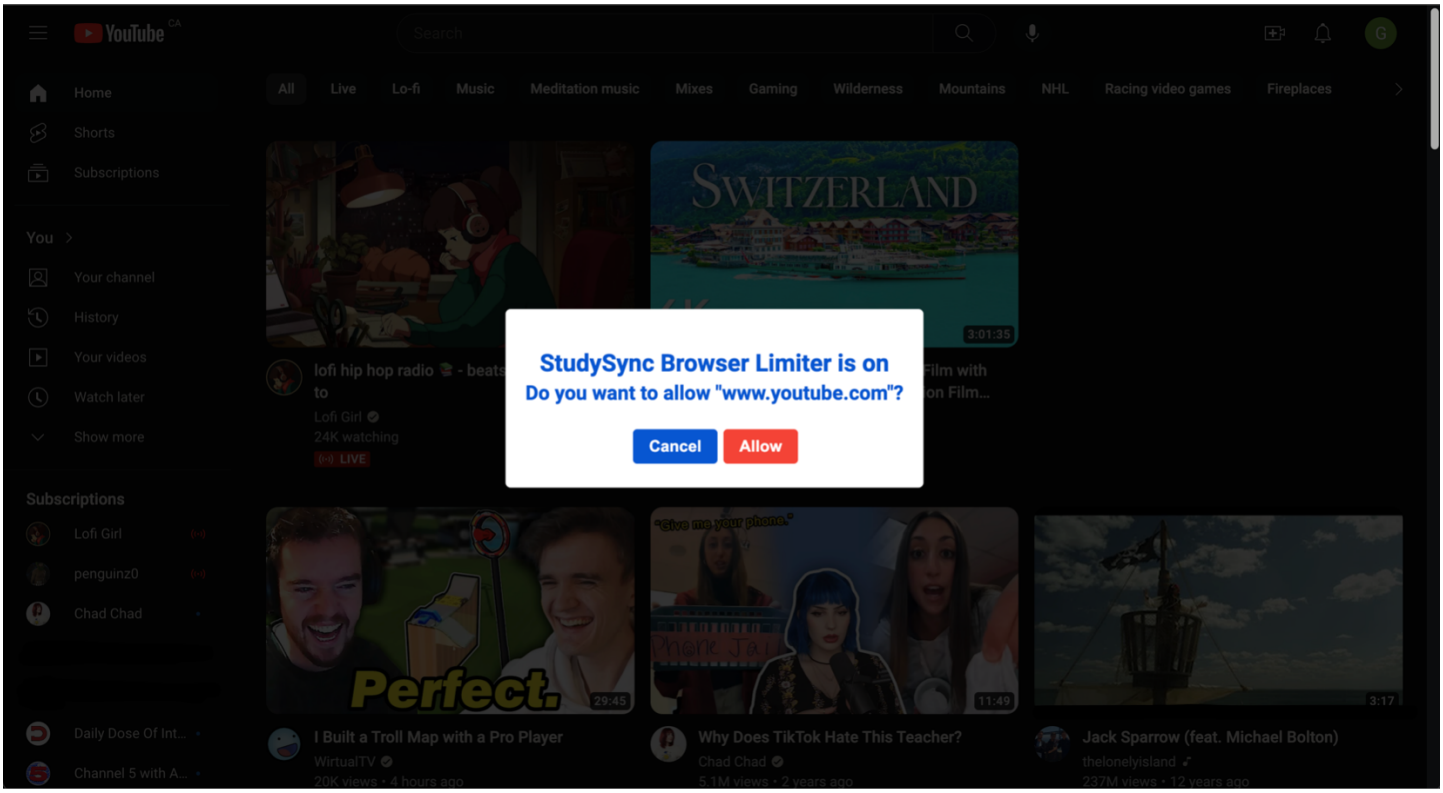


Figure 18: Blocking popup.

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.

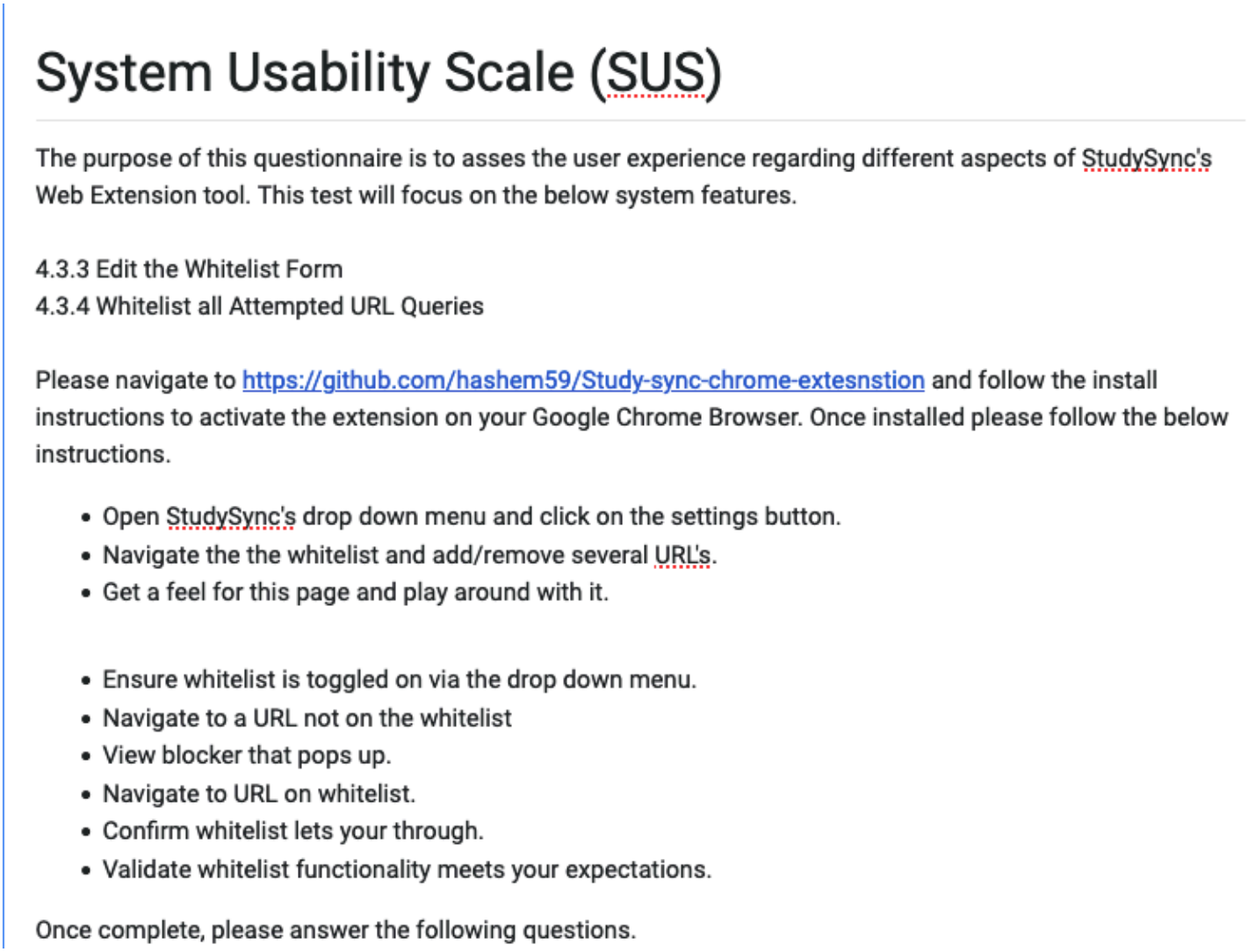
```
async function showBlocker(domain) {
  if (document.querySelector('.study-sync__blocker_overlay')) return;
  // create custom element  and append conetnt to it to protect it from surronding elements
  const overlay = document.createElement('div');
  document.body.appendChild(overlay);

  overlay.classList.add('study-sync__blocker_overlay');
  overlay.innerHTML = `
    <div class="">
      <div class="study-sync__blocker-overlay-content">
        <h2 class="study-sync__blocker-overlay-message">
          StudySync Browser Limiter is on
          <br>
          <small>Do you want to allow "${domain}"?</small>
        </h2>
        <button class="study-sync__blocker-overlay-cancel" type="button">
          Cancel
        </button>
        <button class="study-sync__blocker-overlay-allow" type="button">Allow</button>
      </div>
    </div>
  `;
  overlay.querySelector('.study-sync__blocker-overlay-allow').addEventListener('click', async () => {
    chrome.storage.sync.get(['browserLimiterAllowList']).then(async (data) => {
      const allowList = data.browserLimiterAllowList || [];
      allowList.push(domain);
      overlay.remove();
      await chrome.storage.sync.set({ browserLimiterAllowList: allowList });
    });
  });
  overlay.querySelector('.study-sync__blocker-overlay-cancel').addEventListener(
    'click',
    async () => {
      overlay.remove();
      await chrome.runtime.sendMessage({ type: 'CloseTab' });
    }
  );
}
```

Figure 19: 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 past two system features. The testing team created the SUS survey in Appendix 10.2, which confirmed the users were still happy with the project's direction. Therefore, no changes needed to be made to the extension's design.



System Usability Scale (SUS)

The purpose of this questionnaire is to asses the user experience regarding different aspects of StudySync's Web Extension tool. This test will focus on the below system features.

4.3.3 Edit the Whitelist Form
4.3.4 Whitelist all Attempted URL Queries

Please navigate to https://github.com/hashem59/Study-sync-chrome-extesnstion and follow the install instructions to activate the extension on your Google Chrome Browser. Once installed please follow the below instructions.

- Open StudySync's drop down menu and click on the settings button.
- Navigate the the whitelist and add/remove several URL's.
- Get a feel for this page and play around with it.

- Ensure whitelist is toggled on via the drop down menu.
- Navigate to a URL not on the whitelist
- View blocker that pops up.
- Navigate to URL on whitelist.
- Confirm whitelist lets your through.
- Validate whitelist functionality meets your expectations.

Once complete, please answer the following questions.

Figure 20: SUS Survey Two Description.

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 extension's time-tracker functionality.

Once again, to start this week, our technical team was provided with four days to complete the implementation of this system feature. Unfortunately, we did not account for the difficulty of implementing this system feature. This was the most challenging week to complete, and we’re lucky we had the foresight to gain some extra time in a prior week. Most of the code for this system's features can be found in coursera.js and content.js in the source code files.

The technical team started by implementing the .validate portion of this system requirement. The code of which can be found below. This portion was quite simple, as the team only needed to implement listeners which called helper functions when the user was on a Coursera-based URL. The two functions below would initialize the tracking of tasks on Coursera and, if necessary, create a new course to be tracked on the system.

```
5   if (window.location.href.includes('coursera.org/learn/')) {
6     trackActiveCourse();
7   }
8
9   function trackActiveCourse() {
10    const courseHandle = window.location.href.split('coursera.org/learn/')[1].split('/')[0];
11    addcourseToCoursesList(courseHandle);
12    addCourseToStorage(courseHandle);
13  }
```

Figure 21: .validate code for the Timetracker.

After the time tracker could discern when the user was on a Coursera-based URL, we implemented a generic heartbeat class. This heartbeat class would be the methodology used to track the time spent on Coursera. A snippet of this class can be found below, while the whole class can be found in content.js.

```
class StartHeartbeat {
  constructor(tab = {}, courseHandle = null) {
    this.tab = tab;
    this.courseHandle = courseHandle || this.tab.url.split('/learn/')[1].split('/')[0];
    this.init();
    // Setup visibility change listener outside to avoid attaching multiple listeners
    document.addEventListener('visibilitychange', () => {
      if (!document.hidden) {
        this.init();
      } else {
        if (this.heartbeatInterval !== null) {
          clearInterval(this.heartbeatInterval);
          this.heartbeatInterval = null; // Clear the interval ID
        }
      }
    });
  }

  init() {
    this.heartbeatInterval = null;
    this.heartbeatFrequency = 5000; // 5 seconds
    // Send immediate heartbeat in case the page is already visible
    this.sendHeartbeat();
    // Then set the interval for subsequent heartbeats
    this.heartbeatInterval = setInterval(() => this.sendHeartbeat(), this.heartbeatFrequency);
  }
```

Figure 22: The constructor and init method from the heartbeat class.

The snippet above depicts the heartbeat class's constructor and init methods. The former uses the init function to set up the class's functionality. It creates a five-second interval timer that beats every five seconds while a user is on a valid Coursera course. This class then sends a heartbeat object via a JSON object, which is depicted below and will be captured by the extension. The extension then processes the data within the heartbeat object. This is done through event listeners and helper functions such as sendHeartbeat.

```
146     sendHeartbeat() {
147       const activityType = this.determineActivityType();
148       console.log('Sending heartbeat, activityType: ', activityType);
149       if (activityType) {
150         chrome.runtime.sendMessage({
151           type: 'heartbeat',
152           activityType: activityType,
153           courseHandle: this.courseHandle,
154           tab: this.tab,
155           timeToAdd: this.heartbeatFrequency
156         });
```

Figure 23: The sendHeartbeat method.

The above method is the helper function that sends the heartbeat data in JSON format. This is done via the sendMessage Chrome API, which allows simple message communication within the extension. This JSON data enables the extension to track the user's time per course. One can see the associated metadata, ranging from the activity type to the course name.

Once the technical team got the extension to track time spent on Coursera in general, It was time to add more specific functionality to track the course and task the user was focusing on.

```
124     determineActivityType() {
125       // Placeholder logic for determining activity type based on current URL or page content
126       if (document.querySelector('#main-container video')) {
127         return 'Watching videos';
128       } else if (document.location.pathname.includes('/quiz/') || document.location.pathname.includes('/exam/')) {
129         return 'Assignments';
130       } else if (
131         document.location.pathname.includes('supplement/') ||
132         document.location.pathname.includes('ungradedLab/') ||
133         window.location.href.includes('.pdf')
134       ) {
135         return 'Reading articles';
136       } else if (
137         document.location.pathname.includes('resources/') &&
138         (document.querySelector('video') || document.querySelector('audio'))
139       ) {
140         return 'Watching Webinars';
141       } else {
142         return 'Other activities';
143       }
144     }
```

Figure 24: determineActivityType method from Startheartbeat class.

The StartHeartbeat class uses the method above to confirm what type of task the user is working on. This functionality is hardcoded for the MVP. However, there is potential in the future to allow the user to customize their tasks and get more responsiveness from this data tracking. It uses the URL path to decide what type of task the user is working on.

Lastly, the code for deciding which course the user is working on can be found in the class's constructor. All this information is then combined into a heartbeat object, which is used to store the data.

The technical team took seven days to implement the above functionality, which was behind schedule. Luckily, the oversight in week two of this project development cycle provided the extra week. Therefore, the team as a whole was not too worried about the running time of this system feature. We also had a light two weeks ahead of us and could cut into that time if needed. Due to this delay, the team decided not to meet on February 10th.

Moving forward to the week of February 11th, the technical team started work on storing the time tracker data. Much like the whitelist information, the Chrome storage API was extremely useful in storing the time tracker data. The code below depicts the function onMessage, which is a function defined within background.js. This function handles and stores the heartbeat message information within the Chrome API storage functionality. It does access some data, which will be included below.

```
// listen for CloseTab message, and close the tab
function onMessage() {
  chrome.runtime.onMessage.addListener(async (obj, sender, response) => {
    if (obj.type === 'CloseTab') {
      chrome.tabs.remove(sender.tab.id);
    } else if (obj.type === 'heartbeat') {
      const { courseHandle, activityType, timeToAdd } = obj;
      console.log('heartbeat', courseHandle, activityType, timeToAdd);
      const course = await chrome.storage.sync.get([courseHandle]);
      console.log('course', course); // {courseHandle: {name: 'course name', data: {Assignments: 0, Reading articles: 0, Watching
      Webinars: 0, Watching videos: 0}}}
      const courseData = { ...course[courseHandle], lastActiveAt: Date.now() };
      if (courseData.data[activityType]) {
        courseData.data[activityType] += timeToAdd; // in meliseconds
      } else {
        courseData.data[activityType] = timeToAdd; // in meliseconds
      }

      await chrome.storage.sync.set({ [courseHandle]: courseData });
      await chrome.storage.sync.set({ lastActiveCourse: courseHandle });
    }
  });
}
```

Figure 25: onMessage function.

For the above function to work, data to update needs to be stored, which is where the following function, addCourseToStorage, comes into place. This function handles the creation of a new course storage object, which is used to store course-related data permanently within the storage API. If the course is not yet stored, it creates the object with empty data values and passes the information to the storage API.

This object then acts as the storage block for the course. It updates the tracked time the user has sent within the course. The technical team had issues getting the JSON data into a standardized format. However, with this function's development, the time tracker's main storage aspects were complete. The extension can now effectively track the time and store it within the system.

```
function addCourseToStorage(courseHandle, title, url) {
  // promise based code
  return new Promise((resolve, reject) => {
    chrome.storage.sync.get(courseHandle, function (data) {
      if (data[courseHandle]) {
        console.log('Course already tracked', data[courseHandle]);
        resolve();
      } else {
        // add course to chrome storage
        const course = {
          name: courseHandle.replace(/-/g, ' ').replace(/\b\w/g, (l) => l.toUpperCase()),
          lastVisitedUrl: url,
          firstAccessedAt: Date.now(),
          data: {
            'Watching videos': 0, // in meliseconds
            'Watching Webinars': 0,
            'Reading articles': 0,
            Assignments: 0,
            'Other activities': 0
          }
        };
        chrome.storage.sync.set(
          {
            [courseHandle]: course
          },
          function (result) {
            console.log('Course added to tracked list', result);
            resolve();
          }
        );
      }
    });
  });
}
```

Figure 26: addCourseToStorage function.

The testing team was then tasked with creating a test set to validate the implementation of this system feature. Please refer to the Excel file 4.3.5_test_set.xlsx for a full rundown of the testing scheme. When running this new system feature against the test set, it passed with a 100% success rate, and thus, the implementation was validated.

This week ran 14 days, most of which was spent on implementation. We would have been in a tough spot if we stuck to the original development plan. However, due to the extra time, we were back on schedule and not ahead of it. The team held a meeting on February 17th in which the technical team was applauded for their effort. We were happy with the system's progress and noticed the end of the semester was looming upon us. Thus, we were ready to move on to the sixth system feature.

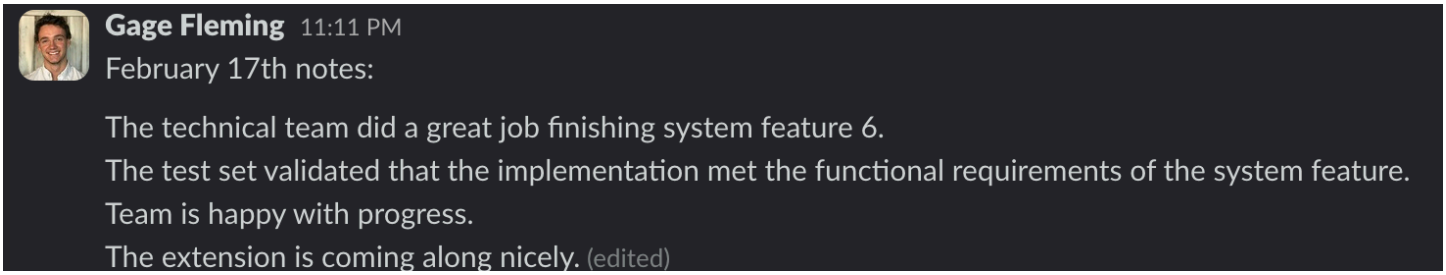


Figure 27: Week 6 meeting notes.

5.1.6 Week Six

This week, we were to implement the system feature 4.3.6 and View the time-tracker stats on the data dashboard. To start, the technical team pointed out some limitations regarding the storage. We would only be able to have a single semester stored. Thus, the semester functional requirement would be removed from the system. We also made a group decision to remove the visualization aspect of this system feature. The users could export the data via an upcoming requirement, and with their skill set, it would be better to let them decide how to visualize the data to get the most out of it. Thus, the functional requirements were refactored to the below.

TimeTracker.DataDashboard:	**Display the data collected by the time tracker.**
.Group: .Course: .SelectCourse	StudySync shall organize the data into logical separations. StudySync shall display data visualizations based on the currently selected course. StudySync shall allow the user to select the current course to be analyzed on the data dashboard.
.Export:	StudySync shall allow the user to export the time-tracker data to CSV format.

The technical team was then sent off to implement this system feature. They started by linking the data to the dropdown menu. This was to catch up on a functional requirement from week one, which required the time tracker data to be stable.

```
try {
  const { lastActiveCourse: lastActiveCourseHandle } = await chrome.storage.sync.get('lastActiveCourse');
  console.log('lastActiveCourseHandle', await chrome.storage.sync.get(lastActiveCourseHandle));
  console.log('lastActiveCourseHandle', await chrome.storage.sync.get([lastActiveCourseHandle]));
  const { [lastActiveCourseHandle]: lastActiveCourse } = await chrome.storage.sync.get(lastActiveCourseHandle);
  let totalTime = 0;

  Object.keys(lastActiveCourse.data).forEach((activity, index) => {
    const time = lastActiveCourse.data[activity];
    totalTime += time;
    console.log('ourse__tracking-info-table', document.querySelector(`.course__tracking-info-table`));
    document.querySelector(`.course__tracking-info-table`).innerHTML += `
    <tr>
      <td>${activity}</td>
      <td>${MsToHM(time)}</td>
    </tr>
  `;
  });

  document.querySelector('.course__summary-title').textContent = lastActiveCourse['Name'] + ` (${MsToHM(totalTime)})`;
  document.querySelector('#last-active-course').style.display = '';
} catch (error) {
  console.log('error', error);
}
```

Figure 28: Code that populates dropdown menu with currently tracked course.

The above code dynamically populates the dropdown menu with the currently tracked course data and responds to the user's actions. It simply queries the local storage within a try-catch block, which is in place in case a URL-related issue pops up. This code would be the final requirement to pass the first week's system feature. Thus, that completed test set can be found in the test dependencies. A fully functioning pop-up is depicted below. One can see how it blends seamlessly with the Coursera UI and responds to user actions.

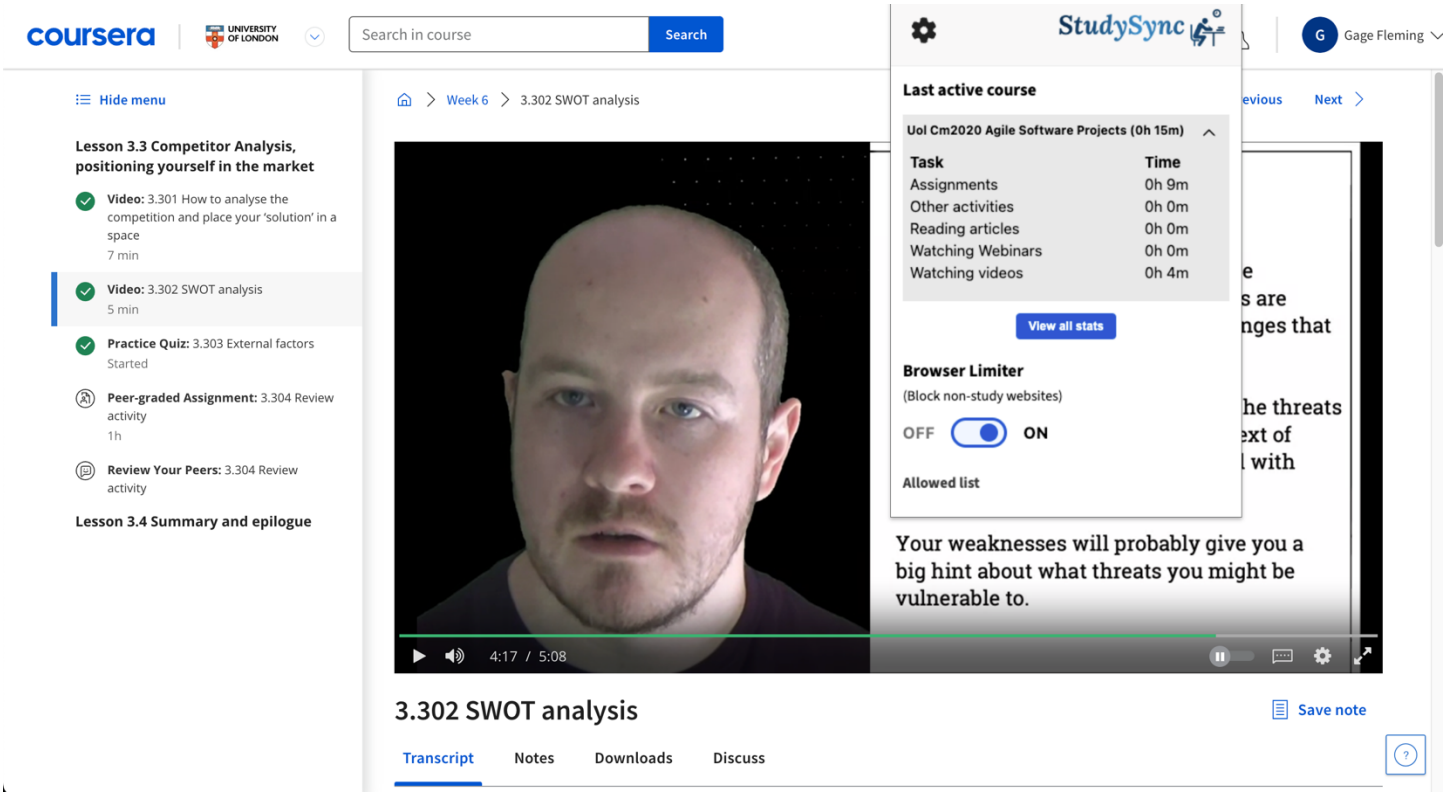


Figure 29: Pop-up reference photo.

The technical team then implemented the data dashboard page. This would group the courses by their respective name and allow the user to access the dropdown list, which depicts the finite data that makes up the whole group. Lastly, the user could export the data via a CSV file. The fully functioning data dashboard page can be found below. It depicts the complete functionality of this system feature.

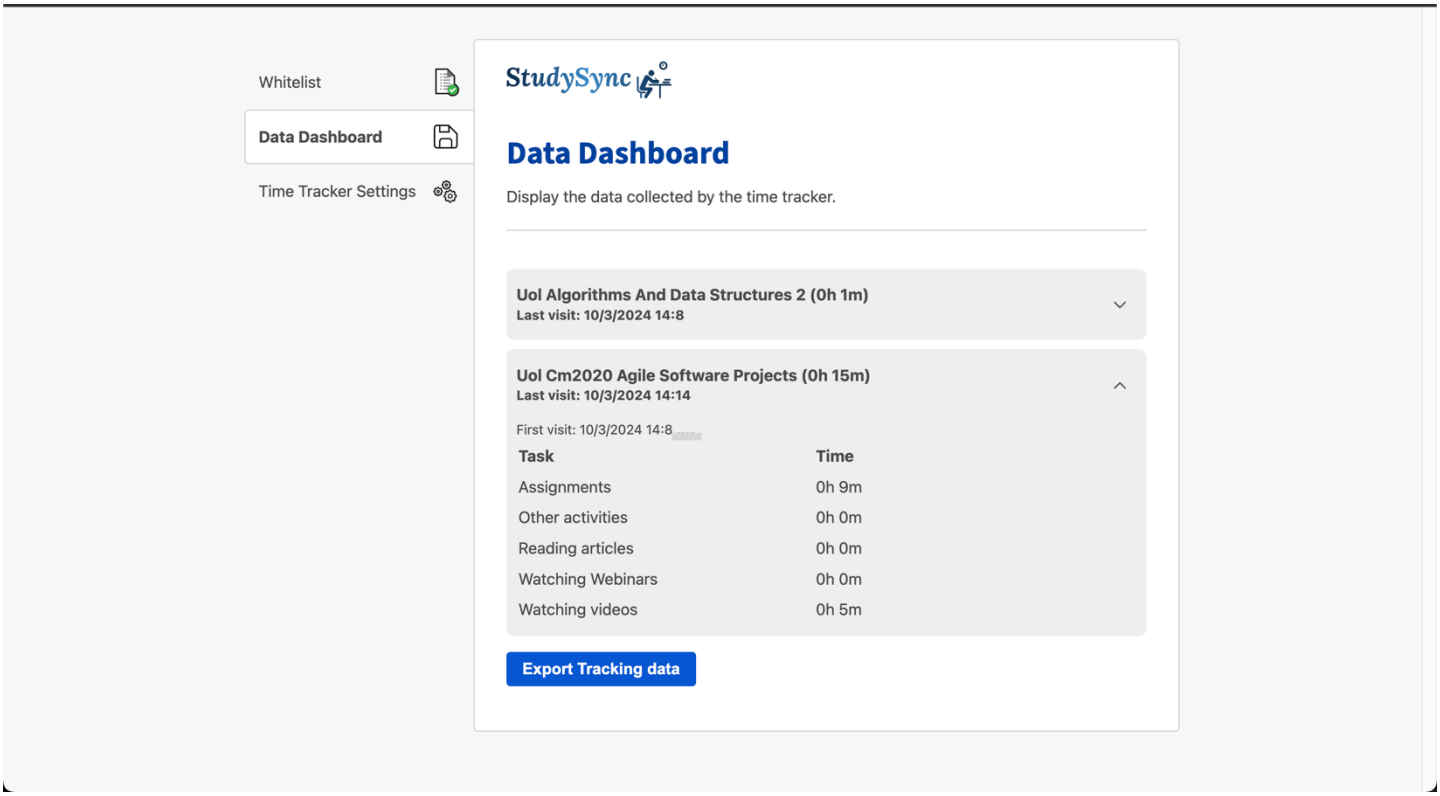


Figure 30: The data dashboard page.

The code for this can be found in the options.html file and options.js. The renderCoursesList function within options.js handles the data dashboard data population. This function queries the local storage and creates course divs that contain and display the data to the user.

```
async function renderCoursesList() {
  // get courses from storage

  const { courses: handles } = await chrome.storage.sync.get('courses');
  let content = '';
  for (const handle of handles) {
    const { [handle]: course } = await chrome.storage.sync.get(handle);
    const totalTime = Object.values(course.data).reduce((acc, time) => acc + time, 0) || 0;
    content += `
    <li class="course">
      <details class="accordion">
        <summary class="course__summary" role="button">
          <strong class="course__summary-title">
            ${course['Name']} (${MsToHM(totalTime)})
            <br />
            <small>Last visit: ${formateDate(course['Last accessed at'])}</small>
          </strong>
          <svg aria-hidden="true" focusable="false" class="icon icon-caret" viewBox="0 0 10 6">
            <path fill-rule="evenodd" clip-rule="evenodd" d="M9.354.646a.5.5 0 00-.708 0L5 4.293 1.354.646a.5.5 0 00-.708.708l4 4a.5.5 0
            00.708 0l4-4a.5.5 0 000-.708z" fill="currentColor"></path>
          </svg>
        </summary>
        <div class="course__tracking-info">
          <small>First visit: ${formateDate(course['First accessed at'])}</small>
          <table class="course__tracking-info-table">
            <tr>
              <th>Task</th>
              <th>Time</th>
            </tr>
            ${Object.keys(course.data).map((activity) => {
              return `<tr><td>${activity}</td><td>${MsToHM(course.data[activity])}</td></tr>`;
            })}
          </table>
        </div>
      </details>
    `;
  }
  const coursesContainer = document.querySelector('.courses-list');
  if (coursesContainer) coursesContainer.innerHTML = content;
}
```

Figure 31: renderCoursesList function.

Lastly, the team implemented the export function to allow users to analyze the data further using their unique skill sets. The function export data below is the primary handler of this task. Hashem's background in web development was critical to getting this function implemented. An example CSV file is depicted below the code for reference.

```
async function exportData() {
  const exportBtn = document.querySelector('[data-action="export-data"]');
  if (!exportBtn) return;
  exportBtn.addEventListener('click', async () => {
    const { courses: handles } = await chrome.storage.sync.get('courses');
    const data = [];
    for (const handle of handles) {
      const { [handle]: course } = await chrome.storage.sync.get(handle);
      data.push(course);
    }
    // create a .csv file that listes all the courses and their activities, time spent on each activity
    const csvString = arrayToCSVString(data);
    var blob = new Blob([csvString], { type: 'text/csv;charset=utf-8;' });
    var link = document.createElement('a');
    var url = URL.createObjectURL(blob);
    link.setAttribute('href', url);
    link.setAttribute('download', 'browser-limiter-data.csv');
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
  });
}
```

Figure 32: exportData function.

Name	First accessed at	Last accessed at	Assignments	Other activities	Reading articles	Watching Webinars	Watching videos
Uol Algorithms And Data Structures 2	10/3/2024 14:8	10/3/2024 14:28	0h 0m	0h 0m	0h 0m	0h 0m	0h 1m
Uol Cm2020 Agile Software Projects	10/3/2024 14:8	10/3/2024 14:14	0h 9m	0h 0m	0h 0m	0h 0m	0h 5m

Figure 33: Example CSV file.

With the functional requirements for this system feature implemented, the testing team implemented a test set to confirm the validity of the implementation. Once again, this system feature passes, as shown in Excel file 4.3.6_test_set.xlsx within the dependencies. They then produced a SUS survey and asked our user demographic to validate the newly implemented features, which returned positive results. The SUS survey description can be found below, and its results are in the appendix. Our team then had a brief meeting on February 24th. We also had our finals looming over us, so we decided to expedite the completion of the remaining system feature.

System Usability Scale (SUS)

The purpose of this questionnaire is to asses the user experience regarding different aspects of StudySync's Web Extension tool. This test will focus on the below system features.

4.3.5 Track Coursera Study Time
4.3.6 View Time-Tracker Stats on Data Dashboard

Please navigate to https://github.com/hashem59/Study-sync-chrome-extesnstion and follow the install instructions to activate the extension on your Google Chrome Browser. Once installed please follow the below instructions.

- Navigate to the Coursera platform.
- Go into a University of London provided course.
- View the drop down menu and the time tracker.

- Navigate to the data dashboard settings page.
- Interact with the stats displayed on screen.

Once complete, please answer the following questions.

Figure 34: SUS Survey Three Description.

5.1.7 Week Seven

This week, we needed to implement system feature 4.3.7, Adjust Time-Tracker Settings. Due to the limitations of the prior week, in which semesters could only be displayed singularly, this system feature was refactored to include a button on the settings page that would erase all stored data and start fresh, creating a new semester. This simple functionality is completed via the below function. It securely deletes the data and sets the extension up for tracking a new semester. It then alerts the user to the update in stored data. The time tracker page is also depicted below the code for reference.

TimeTracker.Settings:	**Adjust the time-tracker settings.**
.Delete:	When the user clicks the delete button, the system shall delete all stored data related to the time-tracker.

```
async function resetData() {
  const resetBtn = document.querySelector('[data-action="reset-data"]');
  if (!resetBtn) return;
  resetBtn.addEventListener('click', async () => {
    // confimr with user
    if (!confirm('Are you sure you want to reset all data?')) return;
    const { courses: handles } = await chrome.storage.sync.get('courses');
    if (!handles) return showmsg();
    for (const handle of handles) {
      await chrome.storage.sync.remove(handle);
    }
    await chrome.storage.sync.remove('courses');
    showmsg();
  });

  const showmsg = () => {
    const msg = document.createElement('p');
    msg.textContent = 'All data has been reset';
    // make it color green
    msg.style.color = 'green';
    resetBtn.parentElement.appendChild(msg);
    setTimeout(() => {
      msg.remove();
    }, 3000);
  };
}
```

Figure 35: resetData function.

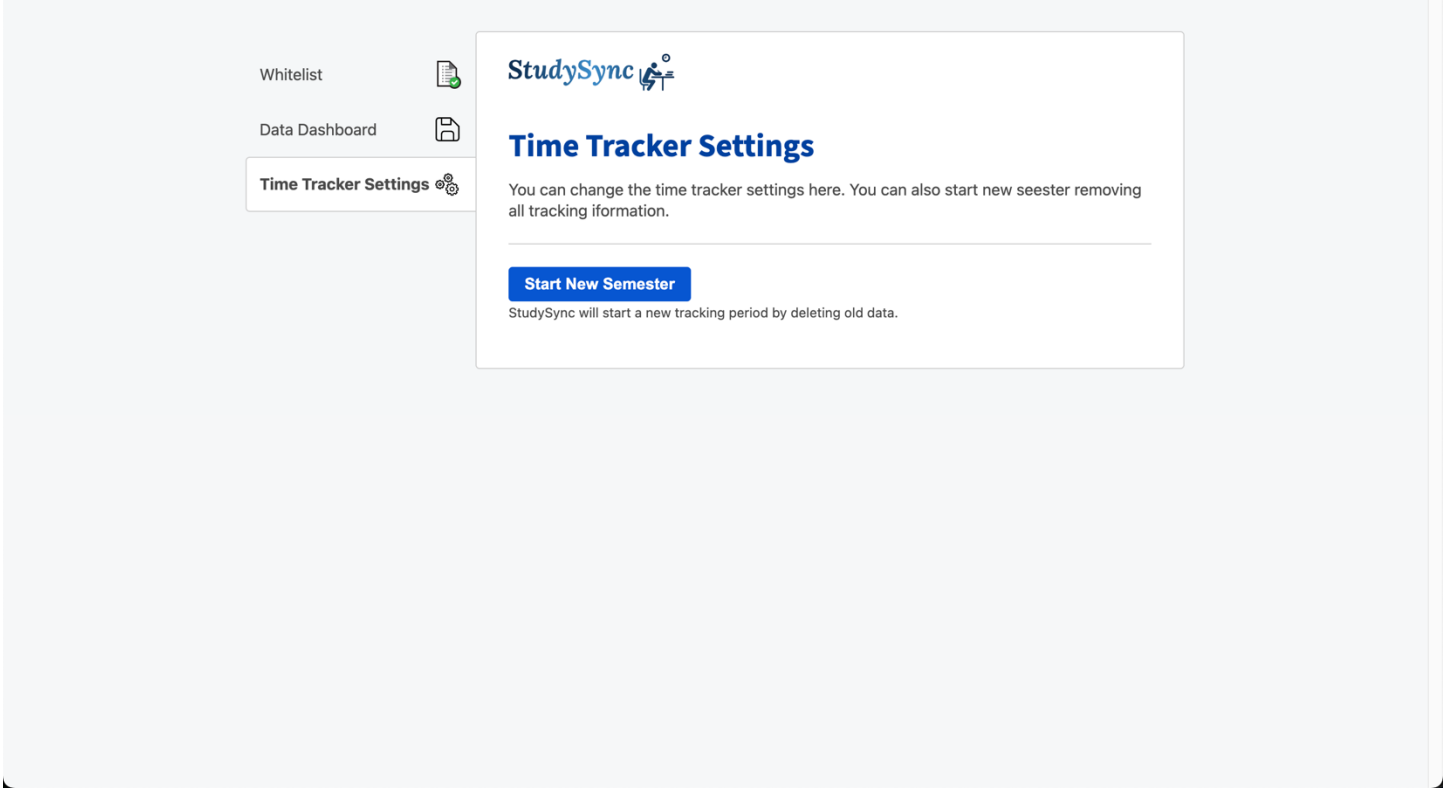


Figure 36: Time tracker settings page.

Finally, to end the development weeks, the team ran all test sets and validated that all system features were implemented correctly. We got a 100% success rate. Thus, we were satisfied that the MVP was now completed. A final SUS survey was then sent out to validate the system, which received positive feedback. The description can be found below, and the whole survey is in the appendix. This effectively ended the development process with the successful implementation of the MVP.

System Usability Scale (SUS)

The purpose of this questionnaire is to asses the user experience regarding different aspects of StudySync's Web Extension tool. This test will focus on the extension as a whole.

Please navigate to https://github.com/hashem59/Study-sync-chrome-extesnstion and follow the install instructions to activate the extension on your Google Chrome Browser. Once installed please interact with the extension and get a feel for it.

Once complete, please answer the following questions.

Figure 37: SUS Survey Four Description.

6 Analysis

In this section, a comprehensive analysis will be complete, which dives into finite parts of the system and validates the success or failure of portions of our implementation.

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. Therefore, the team has successfully implemented the functional components of this project.

6.2 Technical Analysis

While the project met the functional requirements, the technical implementation of these requirements fell a little short. 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, which 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 function below. 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 strong as we would have preferred.

```
async function renderAllowList() {
  const allowList = (await getAllowList()) || [];
  const allowListContainer = document.querySelector('ul.allow-list');
  if (!allowListContainer) return;
  allowListContainer.innerHTML = '';
  if (allowList.length == 0) return;
  let html = '';
  allowList.forEach((domain) => {
    const li = `
      <li class="allow-list__item">
        <div>${domain}</div>
        <button class="button" data-url="${domain}" type="button" title="Remove">
          <svg
            aria-hidden="true"
            focusable="false"
            class="icon icon-remove"
            viewBox="0 0 18 17"
          >
            <use href="#icon-remove"></use>
          </svg>
        </button>
      </li>
    `;
    html += li;
  });
  allowListContainer.innerHTML = html;
  allowListContainer.querySelectorAll('button').forEach((btn) => {
    btn.addEventListener('click', async function (e) {
      const domain = e.currentTarget.getAttribute('data-url');
      const newAllowList = allowList.filter((item) => item !== domain);
      await chrome.storage.sync
        .set({ browserLimiterAllowList: newAllowList })
        .then(() => renderAllowList());
    });
  });
}
```

Figure 38: Poorly related function implementation.

Thus, the technical implementation would be considered subpar, and some work could be done to ensure that it 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 could view it. This requirement was met through the use of the Chrome local storage API. Through online research, the team has found that the local storage appears reasonably secure for non-sensitive data. Reference 11.1 indicates that cross-site scripting is the main threat to local storage, and we don't see that threat as an issue for our target demographic.

We are confident that the data stored by our extension is not sensitive should it be leaked to a malicious entity. The only security concern we did not tackle is a malicious entity gaining access to the web browser. We don't see this as a valid security concern for this extension, as the ignorance of the extension and the attractiveness of other targets would keep the data out of sight. Therefore, we don't feel a password is needed for the extension.

The threat level is low, and we have followed standard security for this project. Thus, the extension has a reasonable security level for a product of this kind.

6.4 Comparative Analysis

In the project proposal, we performed a comprehensive market analysis to see the existing software that could solve our problem space. We found that while there were many high-quality extensions, nothing

met the needs required to solve all three of our project objectives defined in the proposal in section 2.1.3. This validated the development of StudySync. This section will confirm that StudySync has solved the project objectives. Thereby making it the best solution for our target demographic.

6.4.1 PO-1

Our tool successfully reacts and directly integrates with the Coursera website. The whitelist is hard coded to allow all Coursera links by default, and the time tracker effectively integrates with Coursera pages to keep track of user tasks within the Coursera domain. Moving forward, a thriving base has been built to build more functionality that integrates well with the Coursera web platform.

```
128     determineActivityType() {
129       // Placeholder logic for determining activity type based on current URL or page content
130       if (document.querySelector('#main-container video')) {
131         return 'Watching videos';
132       } else if (document.location.pathname.includes('/quiz/') || document.location.pathname.includes('/exam/')) {
133         return 'Assignments';
134       } else if (
135         document.location.pathname.includes('supplement/') ||
136         document.location.pathname.includes('ungradedLab/') ||
137         window.location.href.includes('.pdf')
138       ) {
139         return 'Reading articles';
140       } else if (
141         document.location.pathname.includes('resources/') &&
142         (document.querySelector('video') || document.querySelector('audio'))
143       ) {
144         return 'Watching Webinars';
145       } else {
146         return 'Other activities';
147       }
148     }
```

Figure 39: Example of URL parsing code.

6.4.2 PO-2

StudySync has successfully removed all website directions from study sessions. Any pages not on the whitelist will be blocked by a pop-up that will redirect users to their study sessions. The tool does this in a non-obtrusive way, allowing the user to control their study sessions without annoying them with complicated whitelisting loops. Users can easily list the site via the allow button and access the content if they need to visit it. The extension will then enable them to easily remove the site after the fact from the whitelist settings page.

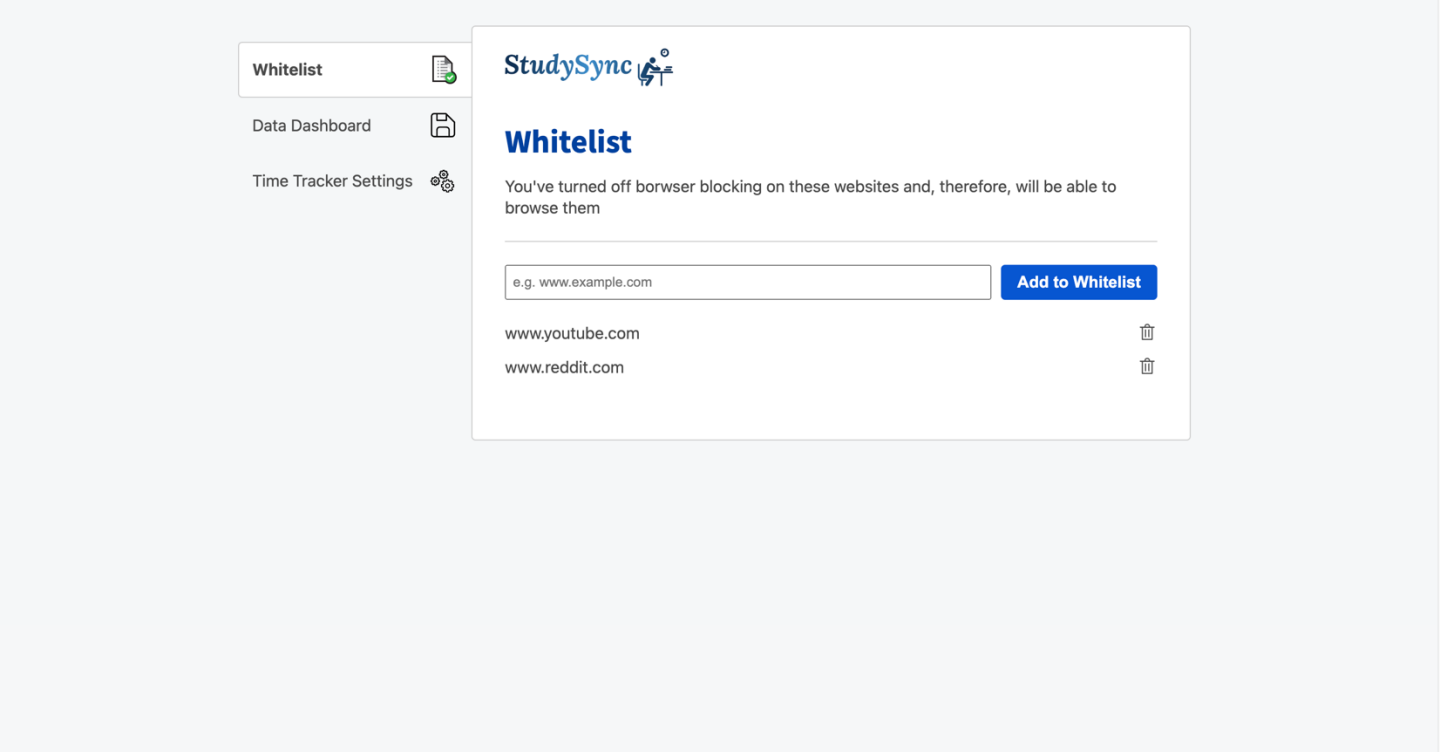


Figure 40: The Whitelist settings page.

6.4.3 PO-3

Lastly, StudySync effectively tracks most student study time on the Coursera platform. We were overzealous in adding the 80% metric to the original project proposal. However, given the nature of the extension, the tracking is successful.

6.5 Usability Analysis

Throughout the project, we performed intermittent SUS surveys to obtain user feedback and validate that adding functionality to our preexisting design led to a positive user reaction. These SUS surveys always returned positive results, leading to the team deeming the extension's usability exemplary.

In this section, we want to validate the user feedback by comparing our MVP to Nielson's ten usability principles. This will ensure that the MVP meets our stakeholders' expectations.

6.5.1 Visibility of System Status

The system keeps users informed via a pop-up that blocks pages, and a timer within the dropdown menu indicates how long a specific task has been tracked. The only addition the team can make is some indication the extension is on via the logo in the browser. Unfortunately, the extension provides a wide range of functionality that cannot be easily seen through the symbol. Therefore, something like the examples below is not possible.

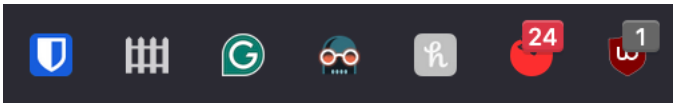


Figure 41: Examples of extension logos

6.5.2 Match Between the System and the Real World

Our system matches what the users expect from an extension integrated with the Coursera platform. Similar colour schemes and clear buttons allow the user to easily adjust the tool to the needs of their Coursera study sessions.

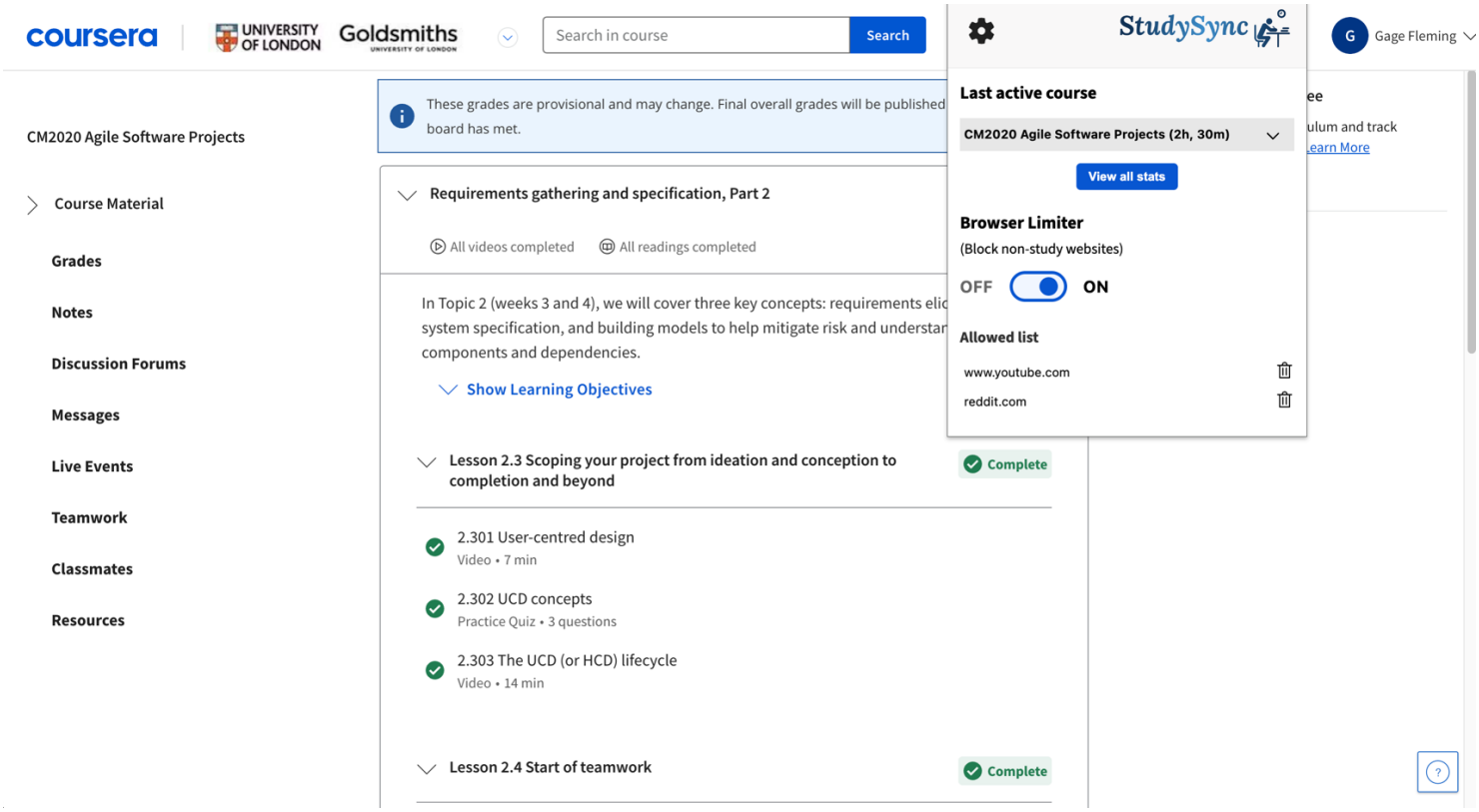


Figure 42: StudySync dropdown over the Coursera course page.

6.5.3 User Control and Freedom

Our simple system allows users to recover from accidental clicks or actions quickly. For example, if a user accidentally adds a URL to the whitelist, they can easily navigate to the settings page and remove the URL from it. No actions are irreversible. The only system that may be less forgiving is the time tracking. Unfortunately, we don't have a way to ensure the user is actively studying on Coursera. If the user were distracted on their phone, there would be no way for the extension to know this. Thus, it would still track the distracted time as study time without a way to reverse this erroneous time. It may be worth exploring allowing the user to remove recent study sessions from the total time for future iterations.

6.5.4 Consistency and Standards

The web extension's UI has a consistent look and feel. The user never has to guess what an action does or decipher what a symbol means.

6.5.5 Error Prevention

Through our system testing and evaluation, the team could not produce any meaningful errors that were unaccounted for. With that being said, our development process could have introduced some errors which would sneak in with user use of time. Our system could falter in this area for an initial launch.

6.5.6 Recognition Rather than Recall

The user is never more than one click away from accessing the functionality they desire. When they reach this functionality, the minimalist UI and simple functionality allow users to recognize the item they wish to look into via clear visual separation.

6.5.7 Flexibility and Efficiency of Use

The system lacks power user functionality. We wanted to focus on improving the basic functionality to an adequate level for the MVP to ensure a good base for potential future development. The current iteration, therefore, does not cater to power users and fails to meet this principle.

6.5.8 Aesthetic and Minimalist Design

The extension has a simple minimalist design that matches Coursera's aesthetic, which is the webpage it was designed for.

6.5.9 Help Users Recognize, Diagnose and Recover from Errors

Our extension does not alert the user to error messages. With simple functionality within the extension, we see a simple web browser reset as sufficient to get past any potential error. This could lead to usability issues should an error come to light that we are unaware of.

6.5.10 Help and Documentation

The README on GitHub for the project has simple instructions for loading the extension onto Google Chrome. However, there currently is no documentation or help for the extension. Therefore, it fails this principle.

The extension passed a brief comparison with Nielson’s usability principles. Combining this with the frequent user feedback confirmed that they, too, found the system usable. We feel confident in saying that the system is usable.

7 Evaluation

7.1 Development Process

The development process we followed for this project followed loose agile principles. We allotted the weekly plan four days a week for implementation, two for testing and user research, and a final one for a group meeting. This process was followed mainly with ad hoc changes as needed. This process, combined with our project plan, led to the successful implementation of an MVP that met the requirements in the project proposal. It also met the expectations of our stakeholders and, most importantly, our end users.

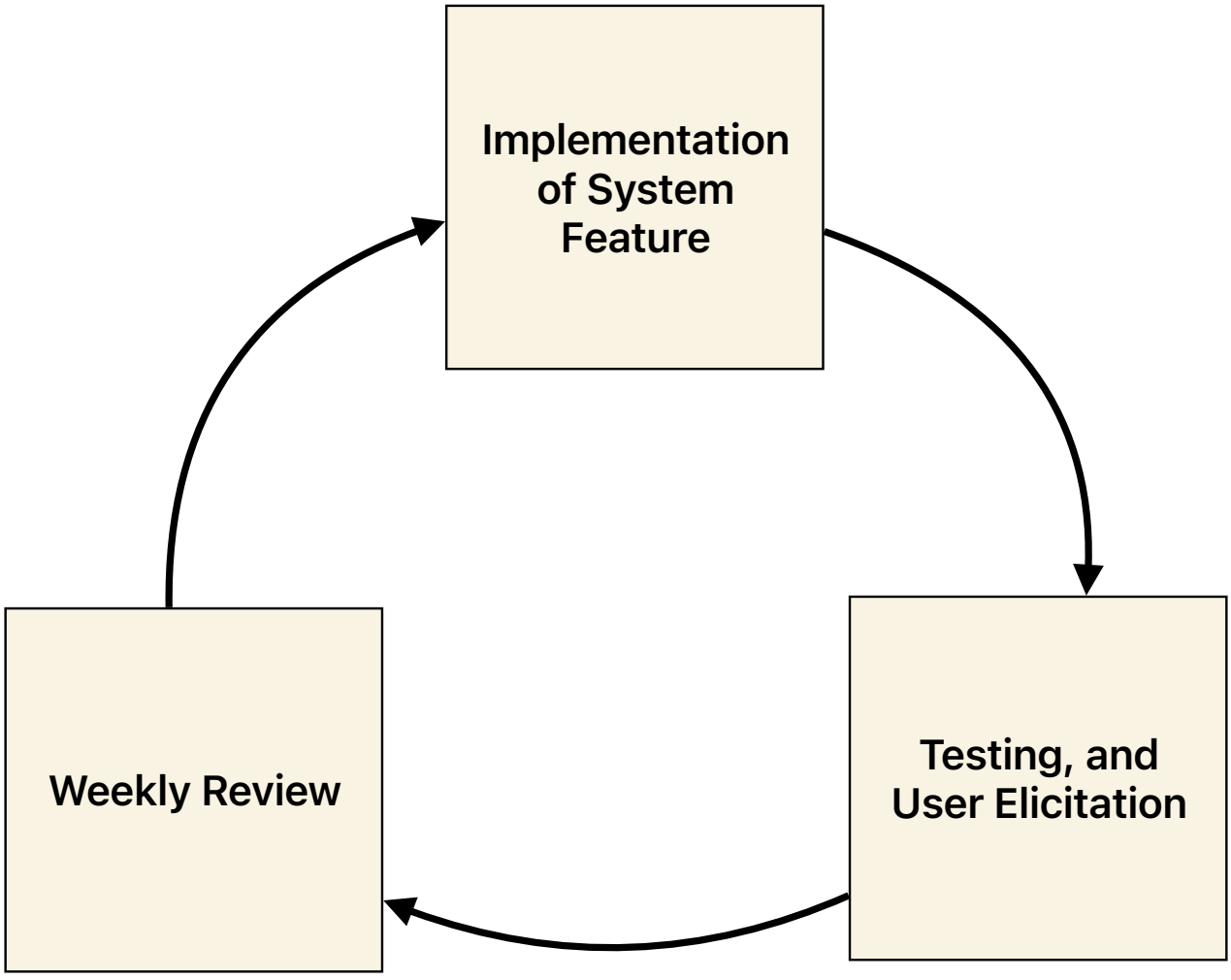


Figure 43: Weekly development process.

The Gantt chart initially proposed in our project proposal was also refactored to change the weeks to focus on system features instead of user stories. This was a positive adjustment as it gave the technical team more concrete goals for implementing the weekly goals.

Unfortunately, as mentioned in the technical analysis, the robustness and reliability of the codebase suffered due to our development approach. We did not dedicate any time to developing unit or code tests in any capacity. Therefore, the development approach would be invalid in the industry. Test suites must be built appropriately for future projects to ensure a robust and reliable code base. If StudySync were to fail due to a bug, the worst-case scenario would be the user's data being lost and their current web session being deleted. However, this approach would not be okay if we were working on something with more consequential meaning. Where more critical data is being managed.

The development approach was good but not well designed. The skill set of our team led to the successful development, where this plan served as a loose guideline. Thus, this plan may not have worked for a different team, and we don't believe that can be said of a great development plan.

In summary, the development process did lead to success. However, its loose nature would not be welcome within the industry, leading to a significant shortcoming involving the lack of a test set.

6.2 Team Evaluation

The team successfully worked together to manage this class and other responsibilities congruently. The team came from various backgrounds and naturally picked up roles where needed. The technical team led by Hashem developed the MVP and responded well to the feedback from the testing team led by Sunidhi and

Mason. Finally, the report was flushed out by the report team led by Gage. This all cumulated into the final deliverable, which we are proud of.

There were some hurdles, such as time zone difficulties and skillset mismatches. Gage was the odd one out as he lived in Canada while the rest of the team resided in London, UK. Thus, a significant time difference was always in play, and it was hard to have live communication in a manner that worked with everyone. Hashem also has multiple years of industry experience, and the team experienced a mismatch in skill set when implementing the code. Luckily, Hashem was very helpful in guiding the rest of the team in creating production-level code.

Overall, the team experienced issues, as with any project, from time zones to planning and development issues. However, we have always overcome hurdles and delivered on the proposed project. Effective teamwork was a significant factor in the success of this project.

6.3 Future of StudySync

Should this project continue, refactoring the codebase and developing a proper unit test suite should be the first steps. Afterwards, a solid base is available to implement further study tools to help our target demographic in their studies. With the vast array of users on the Coursera platform, there is plenty of opportunity to generalize the tool for use with more courses and programs. This could involve tools such as adding the Pomodoro clock or adding a revision tool that quizzes the user based on AI understanding of the content. Ultimately, the users would guide the future of this product, and in a real-world setting, we would want to ensure that a successful uptake is seen with the tool. Otherwise, pursuing further work could result in diminished returns.

If the product continues to succeed, a potential pitch to Coursera would be in store for StudySync. Acquisitions are a significant part of the tech space, and if our tool were deemed valuable enough, Coursera might want to invest in it. After all, exit plans are part of almost every start-up's business plan. We can look at the buzz around mergers and acquisitions within the tech space to confirm this; Tech news sites like Tech Crunch have pages dedicated to this kind of news.

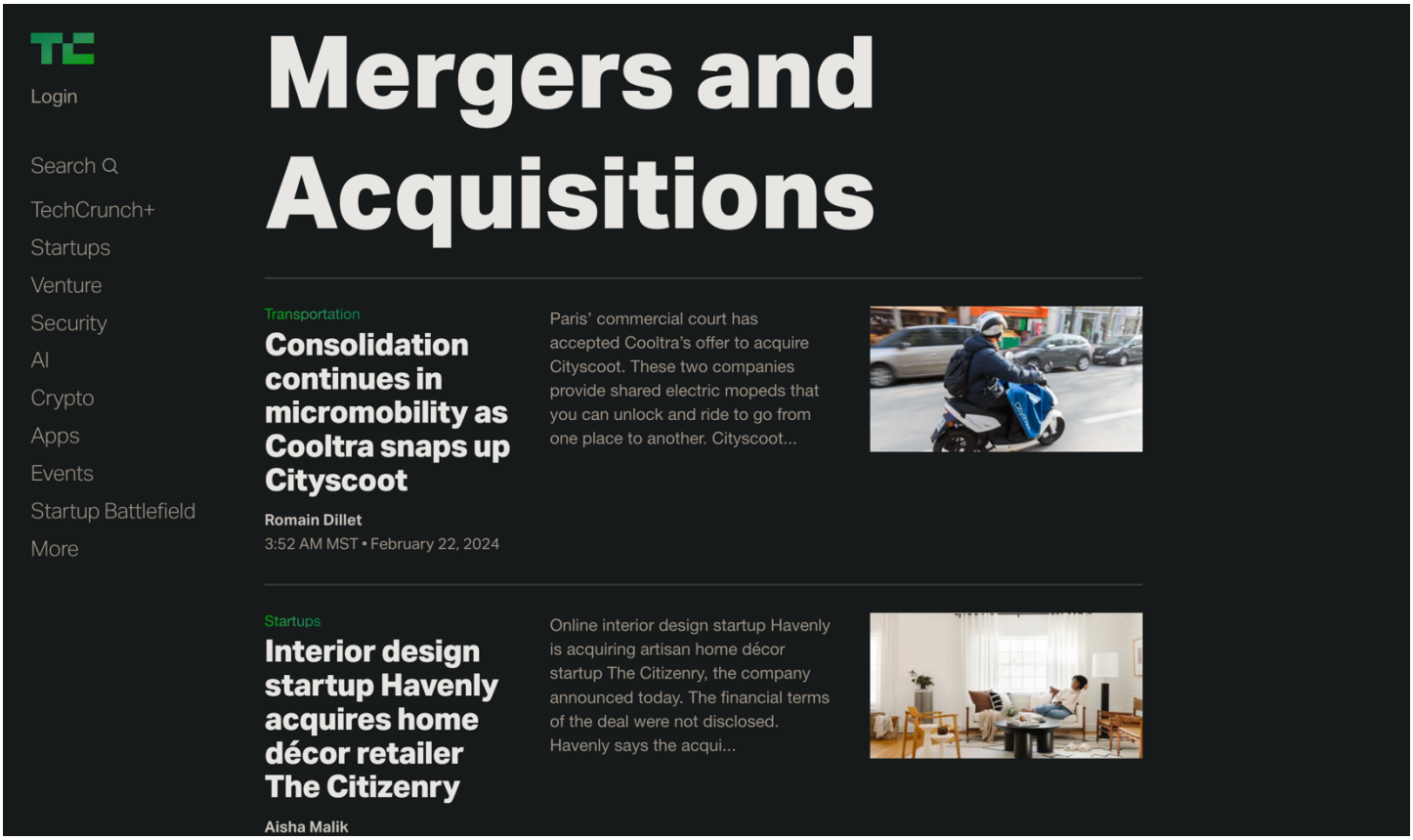


Figure 44: Tech Crunch's mergers and acquisitions page. https://techcrunch.com/tag/mergers-and-acquisitions/

8 Conclusion

Twenty weeks ago, our team started to propose/develop a web extension that would meet the vision statement declared in our introduction. Through effective planning, communication, and hard work, the team built a web extension that met this vision statement. This has been verified through extensive user involvement, stakeholder elicitation and analysis through industry standard measurements. The project successfully solves the problems and meets the objectives in section 2.1.3 of the proposal. This confirms that the tool we have built meets the needs of our target demographic better than any other tool currently on the market.

As users of this extension, we look forward to implementing it into our study toolset to complete this degree. We have also seen positive uptake from our target peers with this extension. While it's too early to tell if the extension will affect more than just our team, we look forward to seeing where the project leads. We believe a tool has been successfully made that can positively impact UofL students' study habits.

The individual teams also feel immense pride in how they contributed to this project. The technical team successfully navigated the implementation of the web extension. The testing team confirmed that the implementation provided met the project proposal standards. Finally, the report team put a nice bow on everything and successfully communicated this journey via a report.

The end-users are happy, and we hope the implementation of the feedback from the midterm and final deliverable is seen positively by the graders of this project. This has not been an easy journey, but it has been one that the team will remember for a long time. All the best – StudySync.

9 Individual Reflection (Gage Fleming)

This project has been a very difficult endeavour. It has challenged me in ways that have pushed my project management skills to the limit. I believe I was instrumental in leading this team and project to the final deliverable. I alone wrote the 8,000 words for the project proposal and the 10,000 words for the final report. Without that work, the team would not have finished the project.

The report was written in consistent language, which helps guide the reader through the project's development and the journey the team took to reach this point. It was meticulously crafted through my research and documentation of the development process as a whole. I documented as much as possible within my current schedule and am proud of my commitment to this project.

I also effectively led the team as a whole. Hashem completed the functional components because of my detailed system requirements and specifications, and the testing team's workflow was guided by my guidance.

My shortcomings were inclusion and patience. One member decided not to show up, and the work ethic of two other members was not near mine. It’s hard to adjust my expectations with group members, especially when the project is worth your grade. My project planning was also the penultimate guide toward completing the project. However, I could have let people in more to help them understand the why of certain decisions.

This journey has revealed my shortcomings and highlighted my strengths. I hope to apply these lessons to the rest of my degree journey. With gratitude, Gage Fleming.

10 Appendix

10.1 SUS Survey for System Features 4.3.1 and 4.3.2

System Usability Scale (SUS)

The purpose of this questionnaire is to asses the user experience regarding different aspects of StudySync's Web Extension tool. This test will focus on the below system features.

4.3.1 Interact with the Extension via the Dropdown Menu
4.3.2 Navigate the HTML Pages via the Navbar

Please navigate to https://github.com/hashem59/Study-sync-chrome-extesnstion and follow the install instructions to activate the extension on your Google Chrome Browser. Once installed please follow the below instructions.

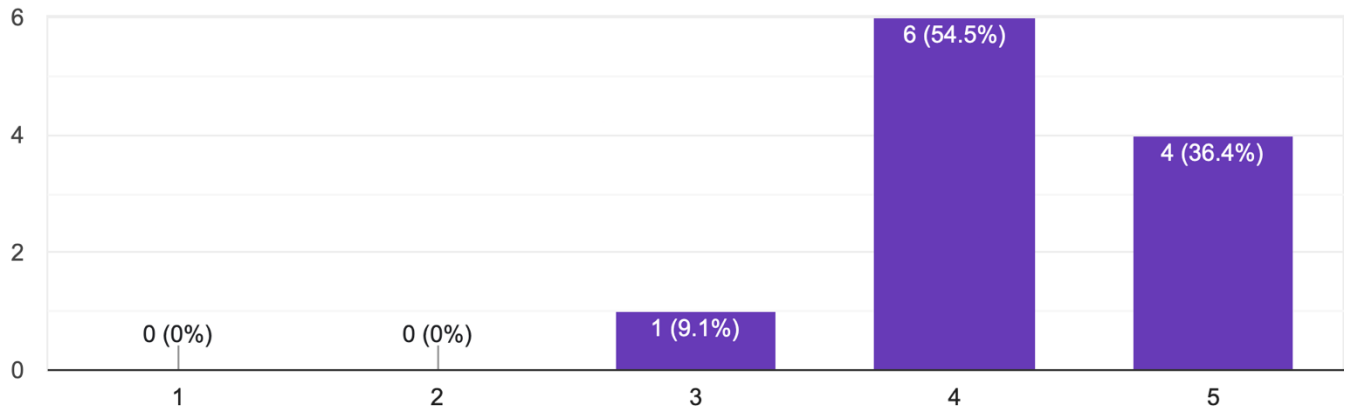
- Click on the StudySync extension logo.
- Interact with the drop down menu and get a feel for its functionality.

- Click on the settings button in the drop down menu.
- Interact with the navbar to change between the settings pages.

Once complete, please answer the following questions.

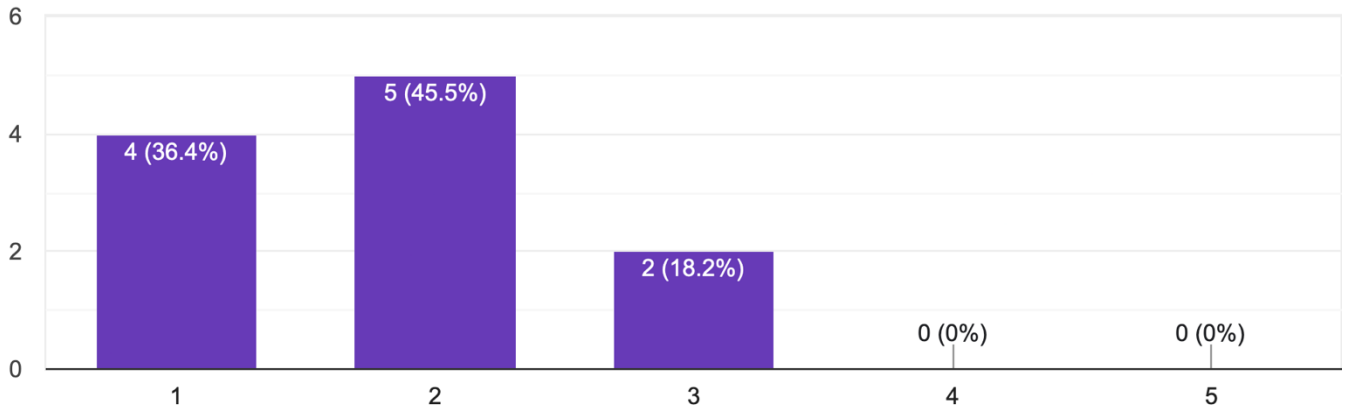
1. I think that I would like to use this system frequently

11 responses



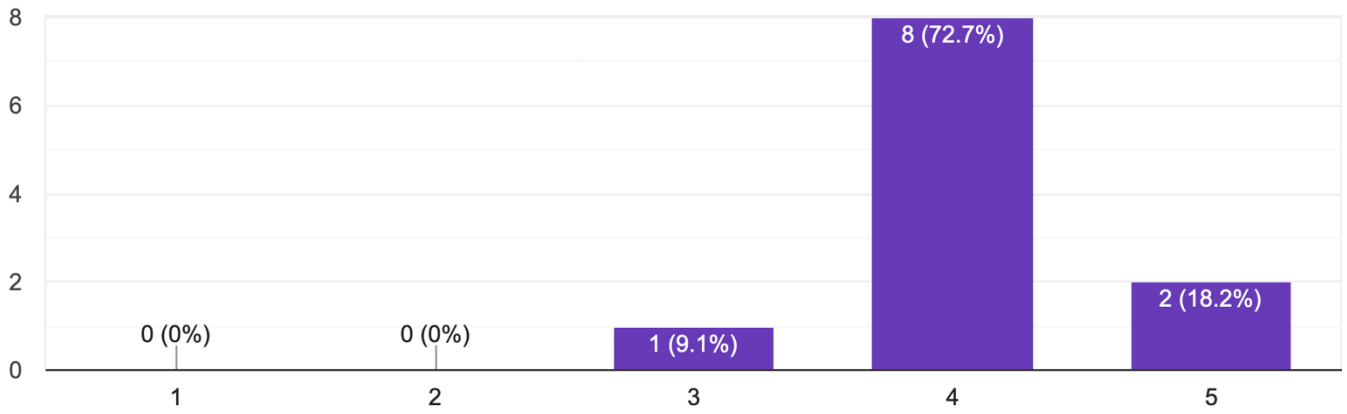
2. I found the system unnecessarily complex

11 responses



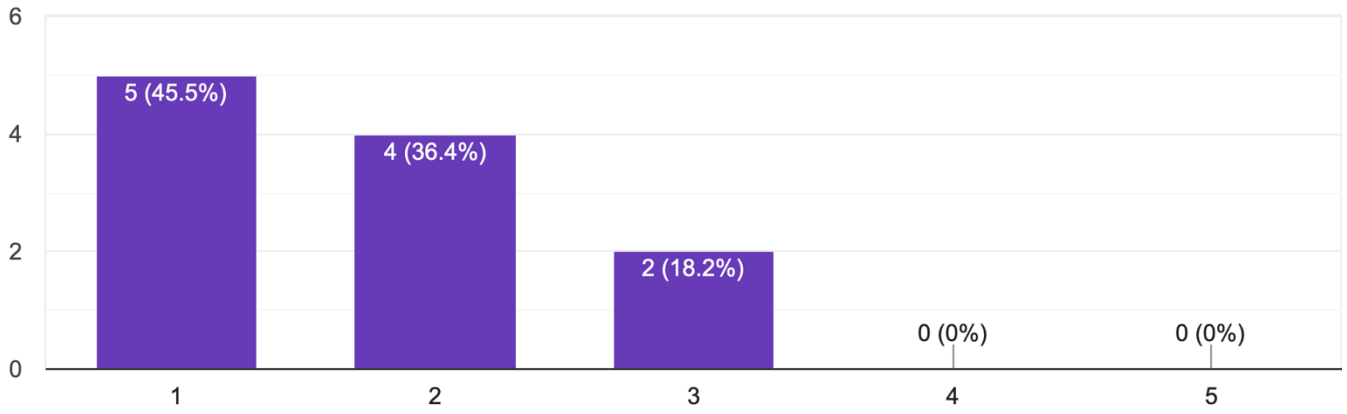
3. I thought the system was easy to use

11 responses



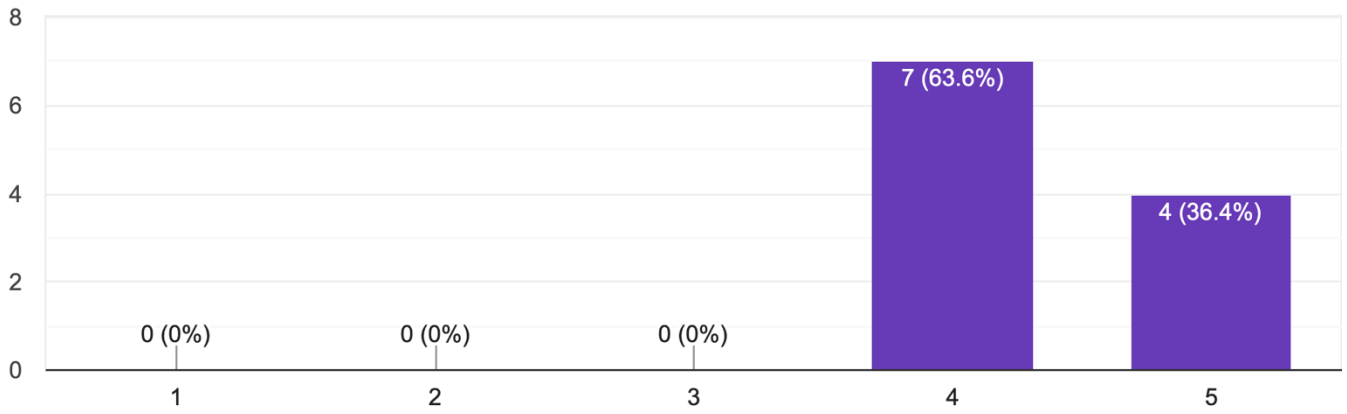
4. I think that I would need the support of a technical person to be able to use this system.

11 responses



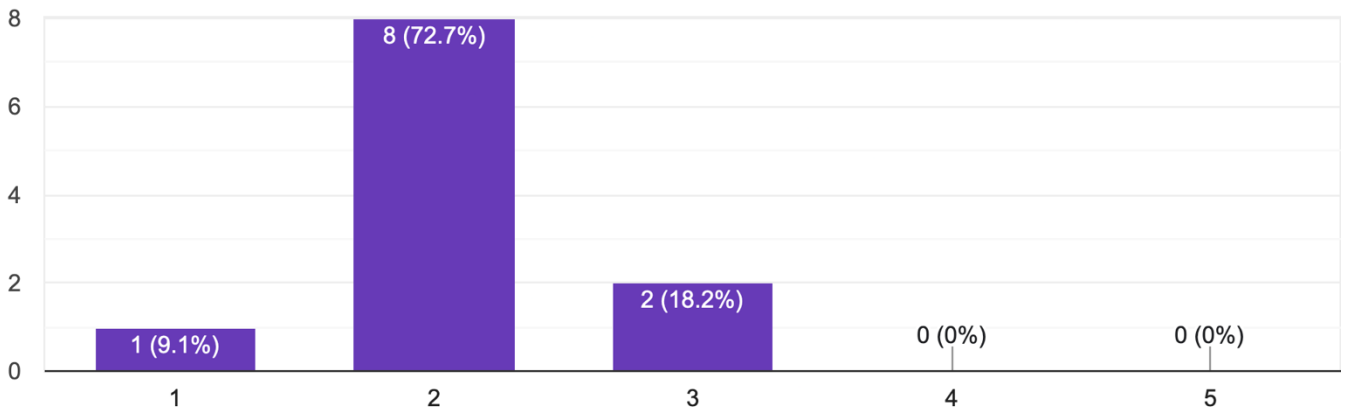
5. I found the various functions in this system were well integrated.

11 responses



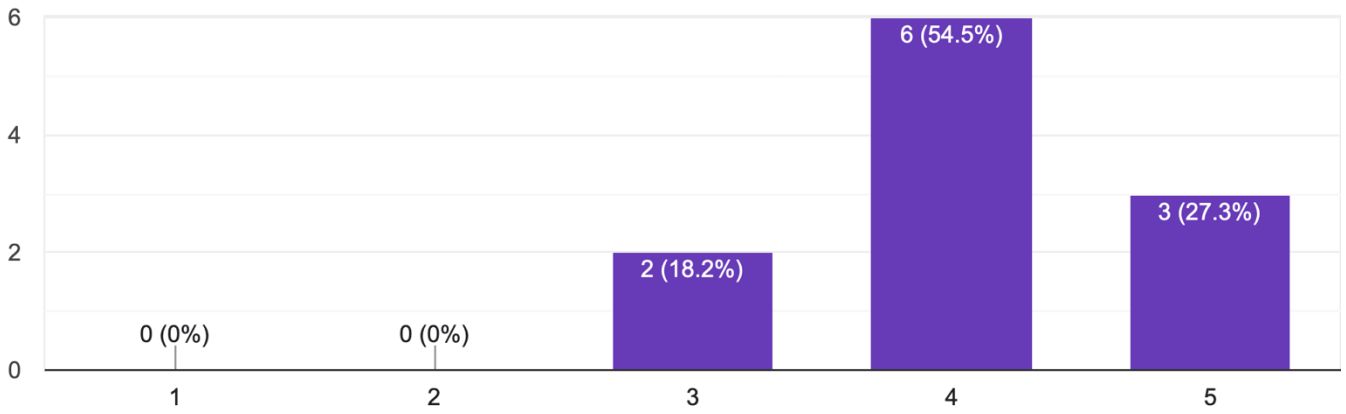
6. I thought there was too much inconsistency in this system.

11 responses



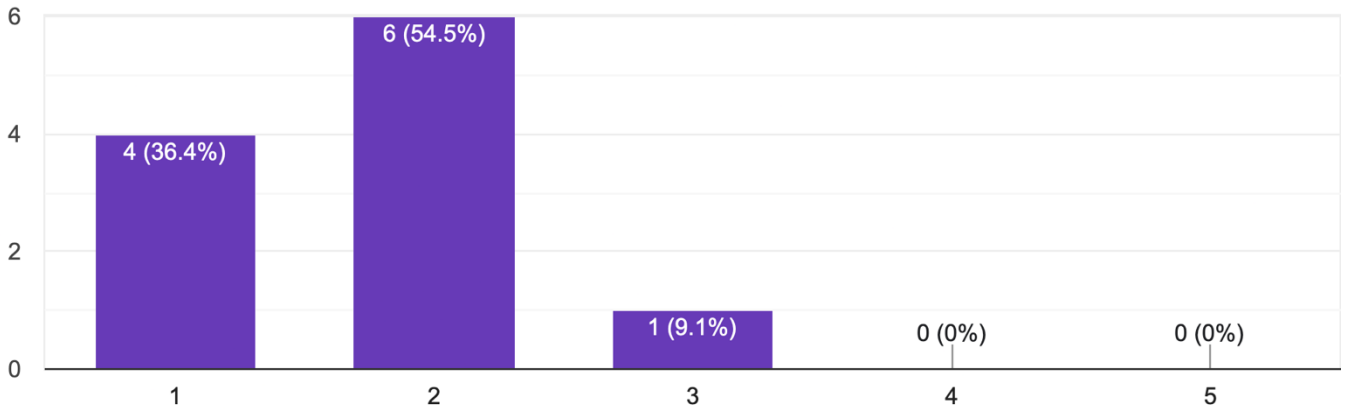
7. I would imagine that most people would learn to use this system very quickly

11 responses



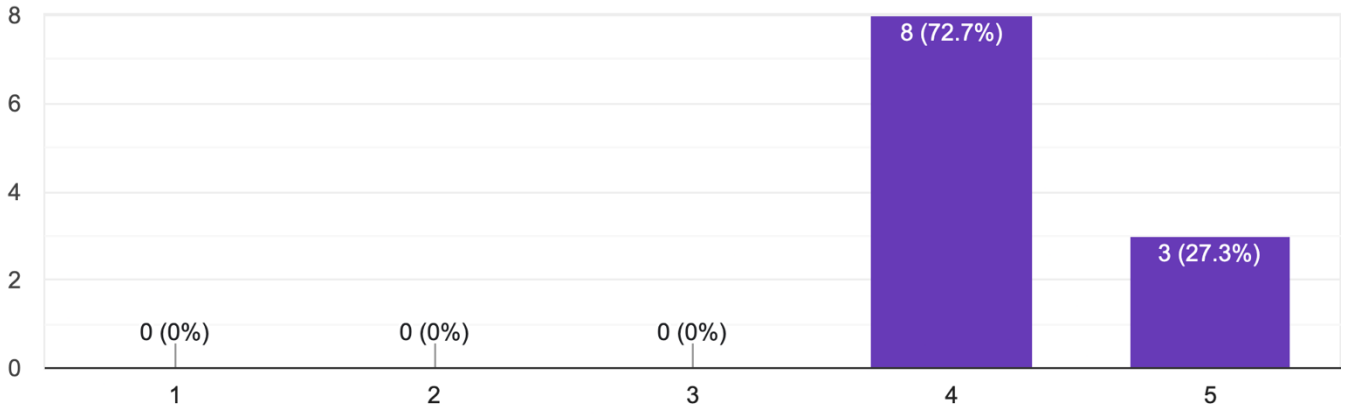
8. I found the system very cumbersome to use

11 responses



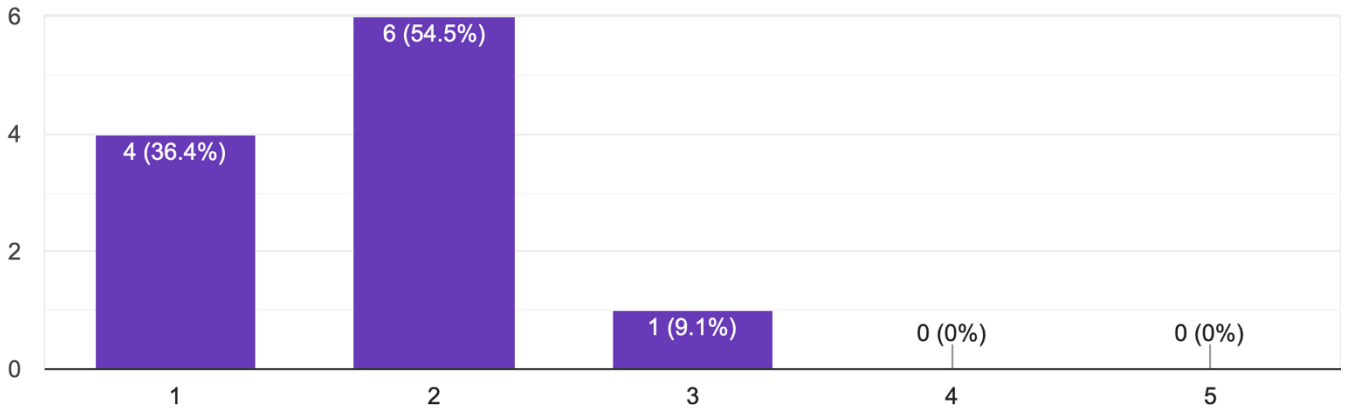
9. I felt very confident using the system

11 responses



10. I needed to learn a lot of things before I could get going with this system

11 responses



10.2 SUS Survey for System Features 4.3.3 and 4.3.4

System Usability Scale (SUS)

The purpose of this questionnaire is to asses the user experience regarding different aspects of StudySync's Web Extension tool. This test will focus on the below system features.

4.3.3 Edit the Whitelist Form
4.3.4 Whitelist all Attempted URL Queries

Please navigate to https://github.com/hashem59/Study-sync-chrome-extesnstion and follow the install instructions to activate the extension on your Google Chrome Browser. Once installed please follow the below instructions.

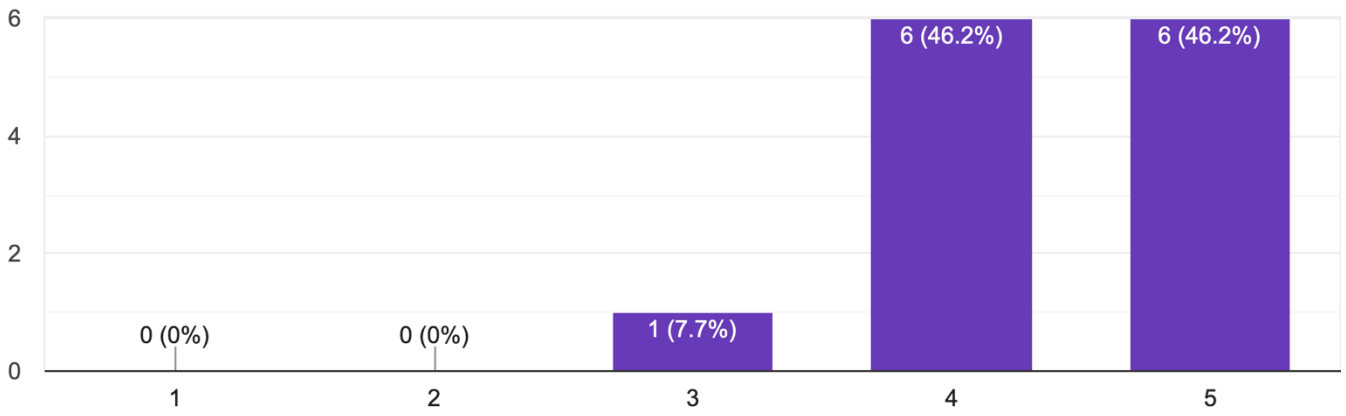
- Open StudySync's drop down menu and click on the settings button.
- Navigate the the whitelist and add/remove several URL's.
- Get a feel for this page and play around with it.

- Ensure whitelist is toggled on via the drop down menu.
- Navigate to a URL not on the whitelist
- View blocker that pops up.
- Navigate to URL on whitelist.
- Confirm whitelist lets your through.
- Validate whitelist functionality meets your expectations.

Once complete, please answer the following questions.

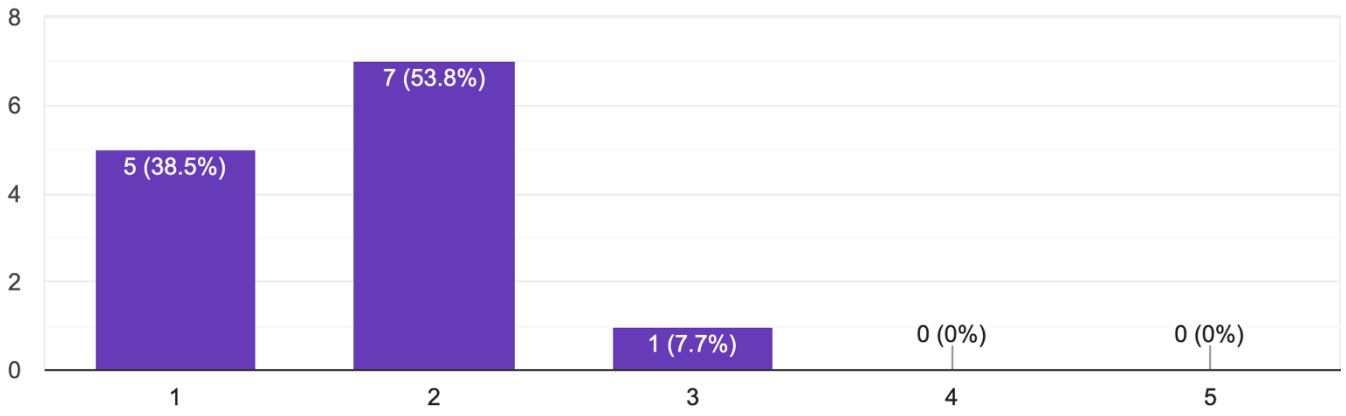
1. I think that I would like to use this system frequently

13 responses



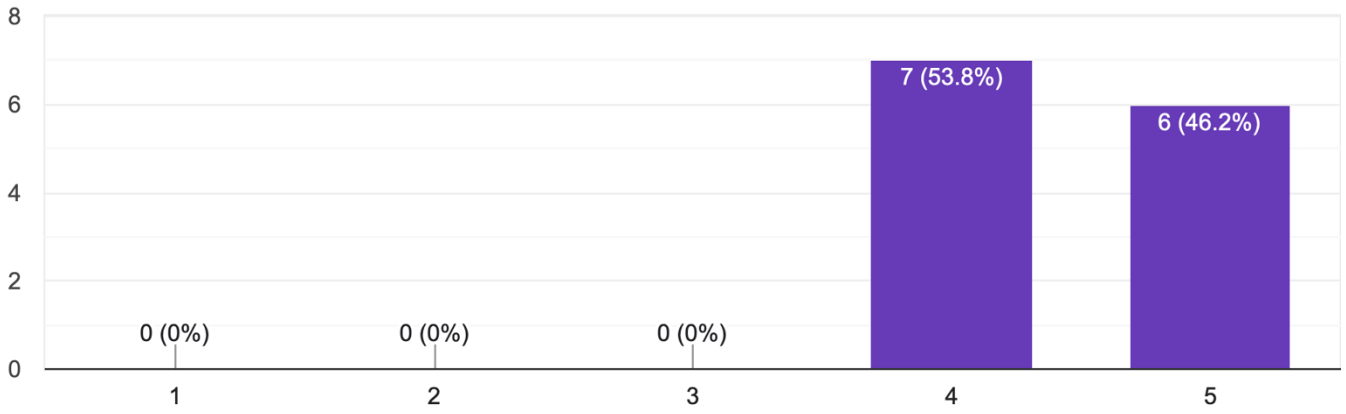
2. I found the system unnecessarily complex

13 responses



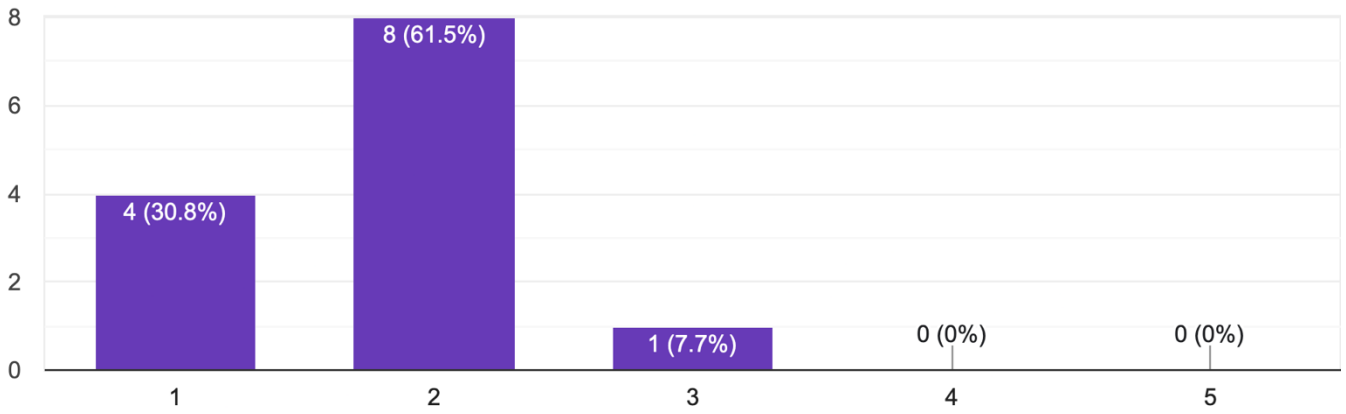
3. I thought the system was easy to use

13 responses



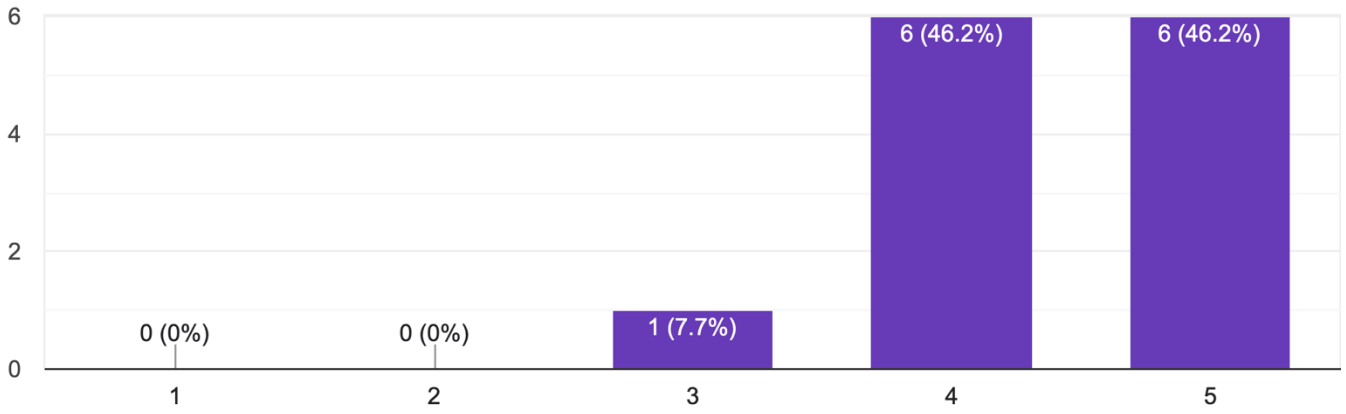
4. I think that I would need the support of a technical person to be able to use this system.

13 responses



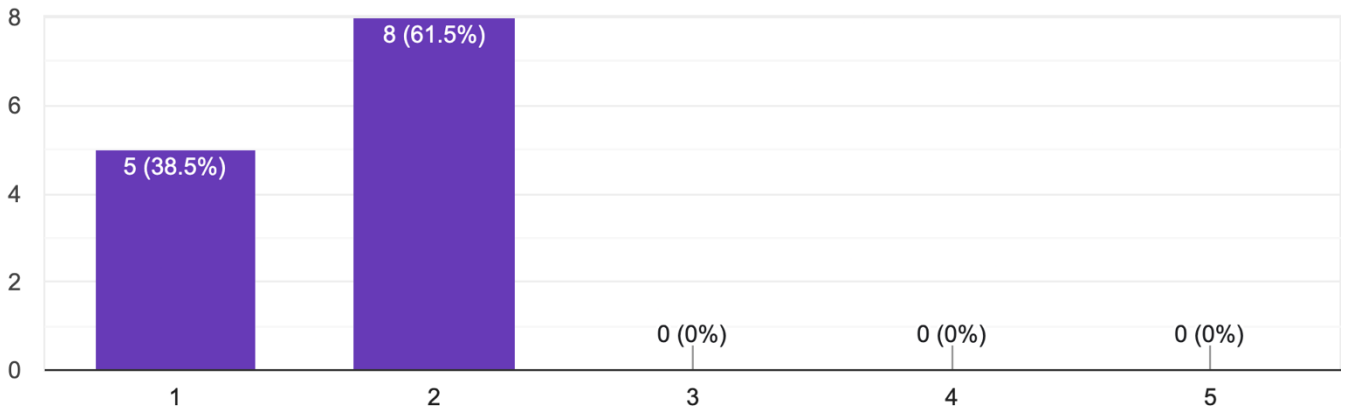
5. I found the various functions in this system were well integrated.

13 responses



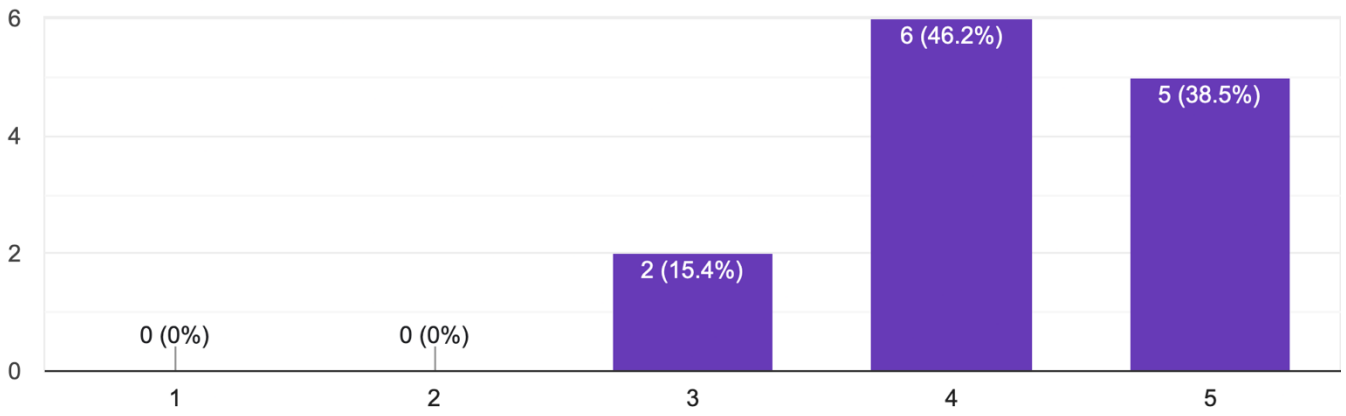
6. I thought there was too much inconsistency in this system.

13 responses



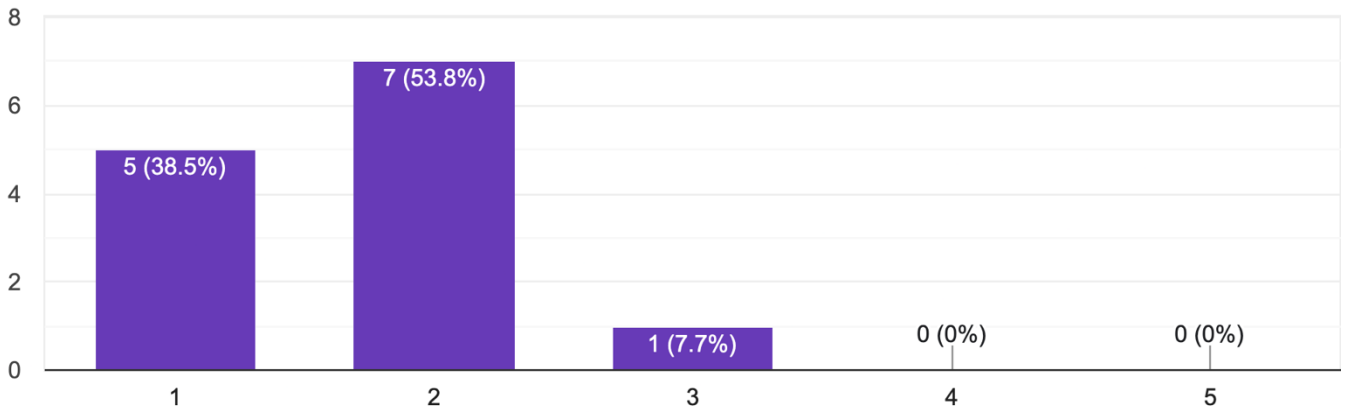
7. I would imagine that most people would learn to use this system very quickly

13 responses



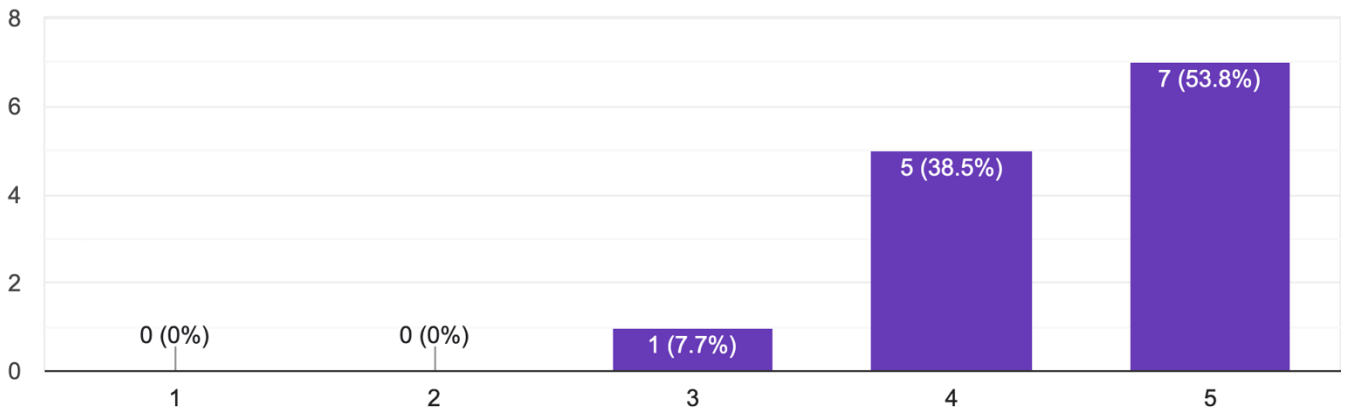
8. I found the system very cumbersome to use

13 responses



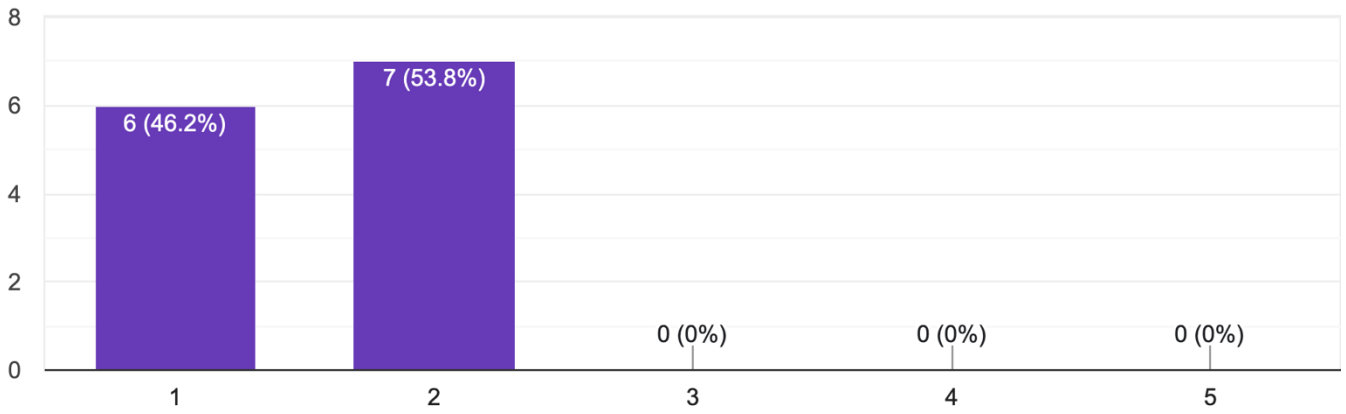
9. I felt very confident using the system

13 responses



10. I needed to learn a lot of things before I could get going with this system

13 responses



10.3 SUS Survey for System Features 4.3.5 and 4.3.6

System Usability Scale (SUS)

The purpose of this questionnaire is to asses the user experience regarding different aspects of StudySync's Web Extension tool. This test will focus on the below system features.

4.3.5 Track Coursera Study Time
4.3.6 View Time-Tracker Stats on Data Dashboard

Please navigate to https://github.com/hashem59/Study-sync-chrome-extesnstion and follow the install instructions to activate the extension on your Google Chrome Browser. Once installed please follow the below instructions.

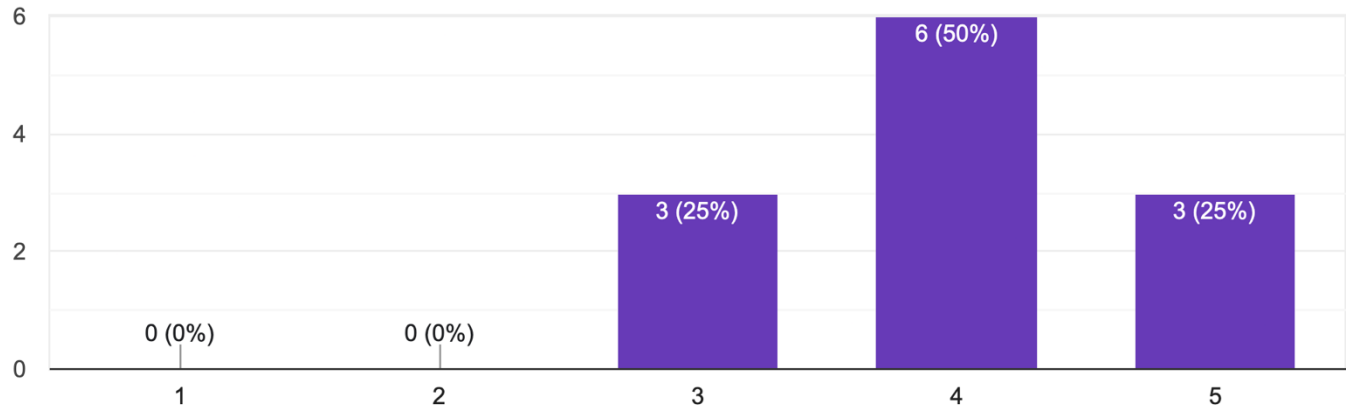
- Navigate to the Coursera platform.
- Go into a University of London provided course.
- View the drop down menu and the time tracker.

- Navigate to the data dashboard settings page.
- Interact with the stats displayed on screen.

Once complete, please answer the following questions.

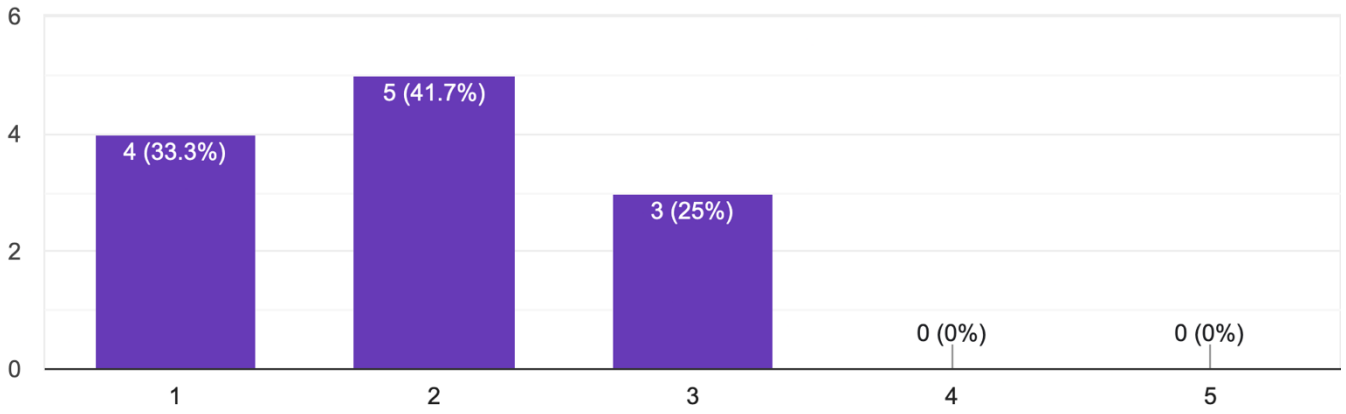
1. I think that I would like to use this system frequently

12 responses



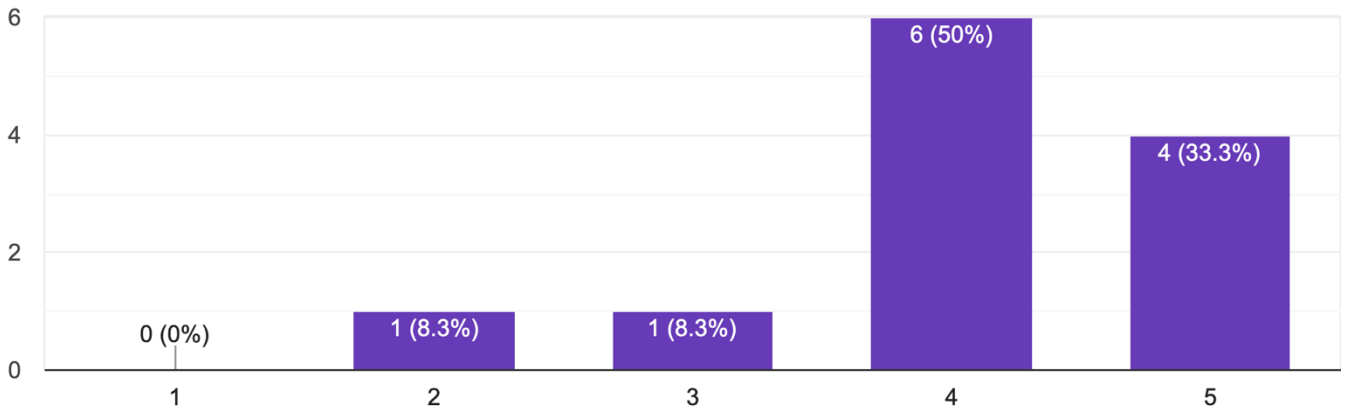
2. I found the system unnecessarily complex

12 responses



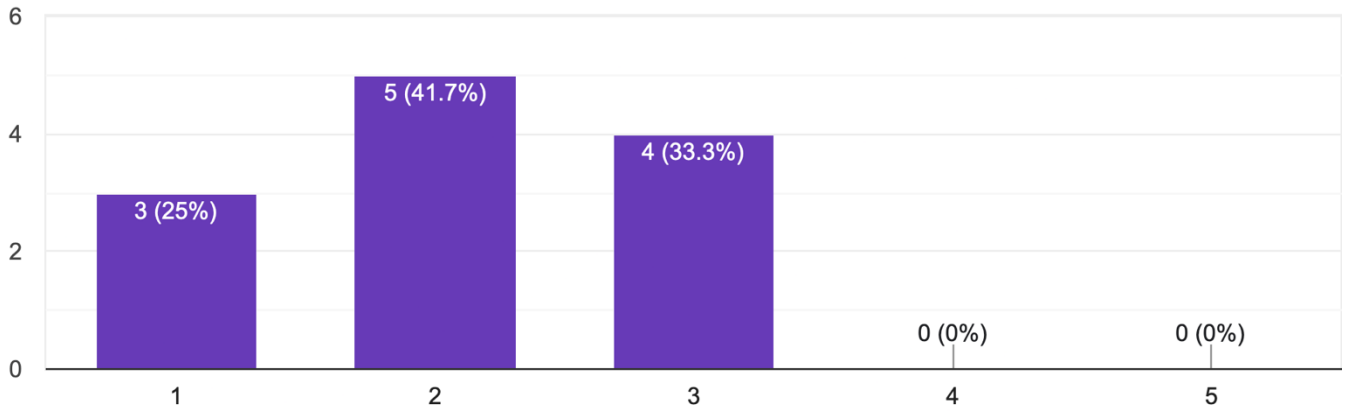
3. I thought the system was easy to use

12 responses



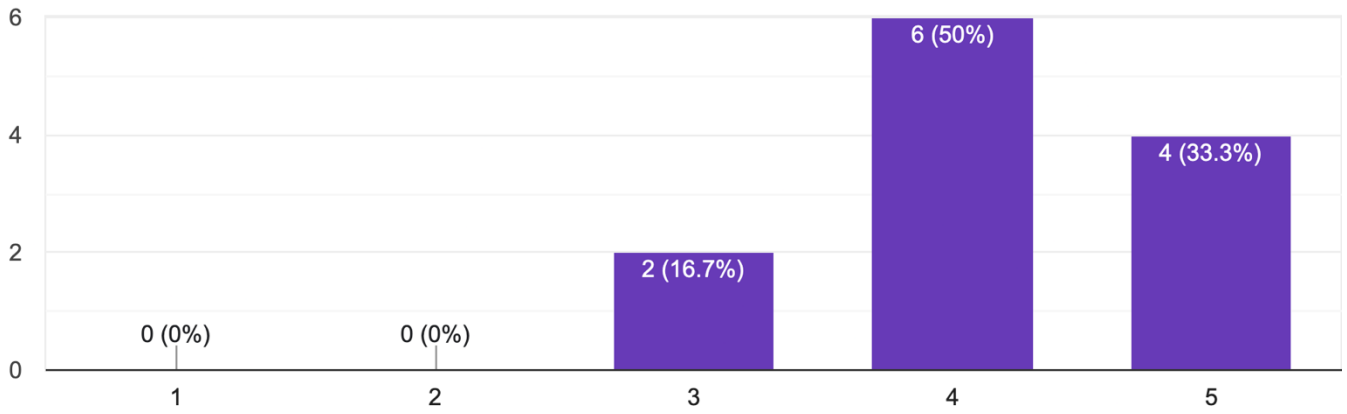
4. I think that I would need the support of a technical person to be able to use this system.

12 responses



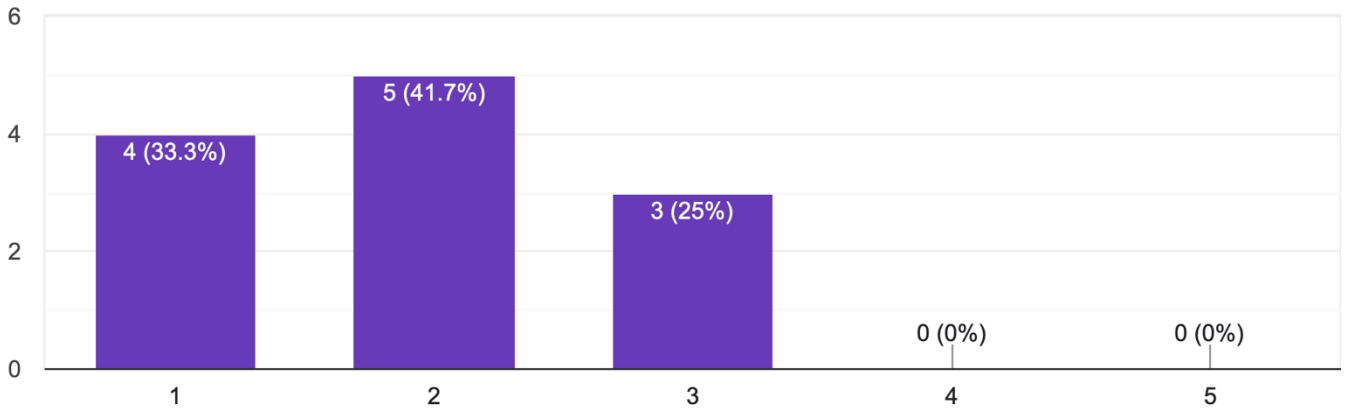
5. I found the various functions in this system were well integrated.

12 responses



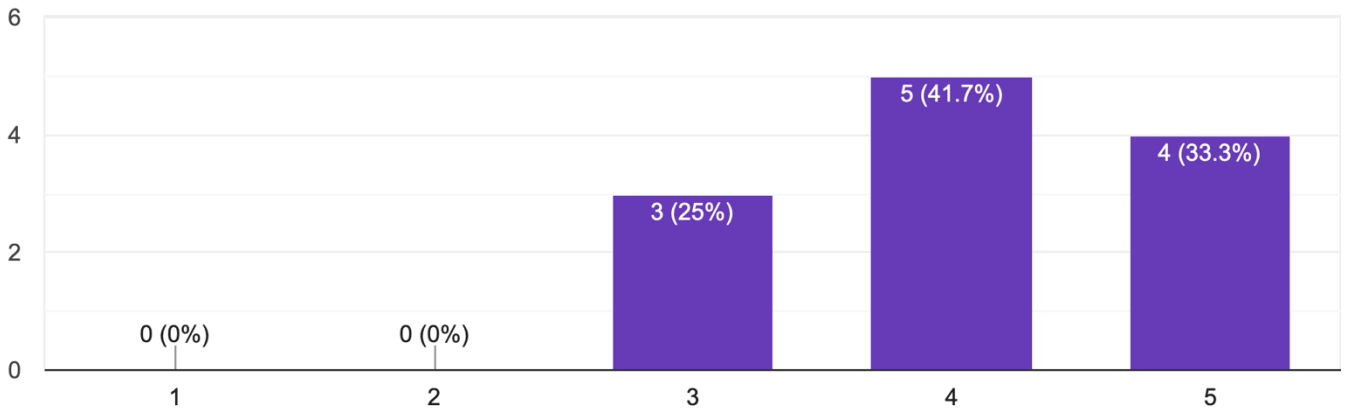
6. I thought there was too much inconsistency in this system.

12 responses



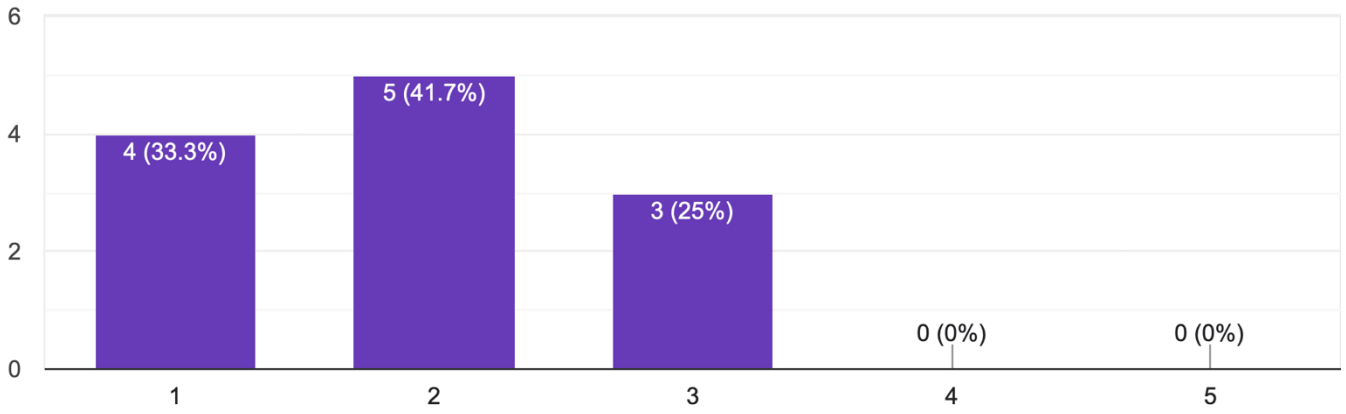
7. I would imagine that most people would learn to use this system very quickly

12 responses



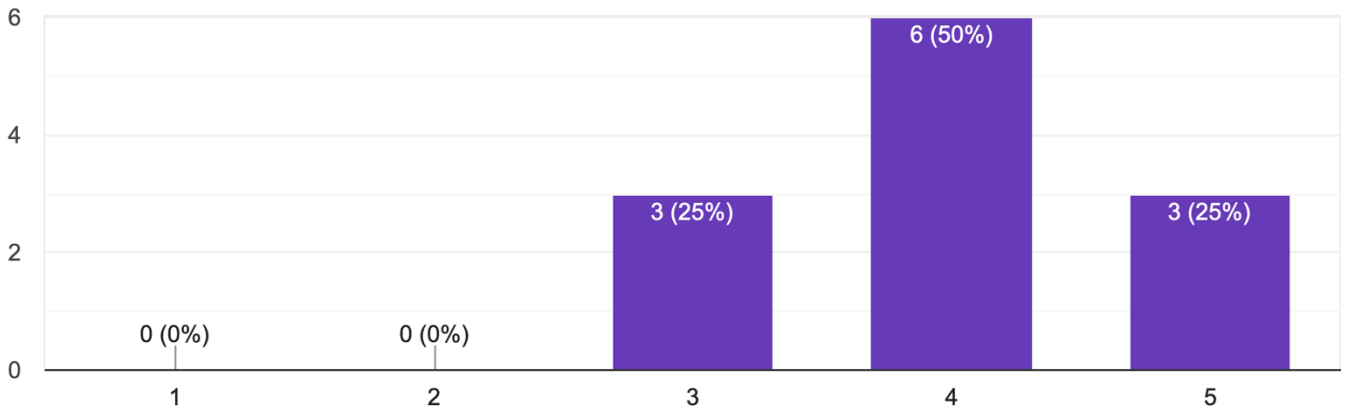
8. I found the system very cumbersome to use

12 responses



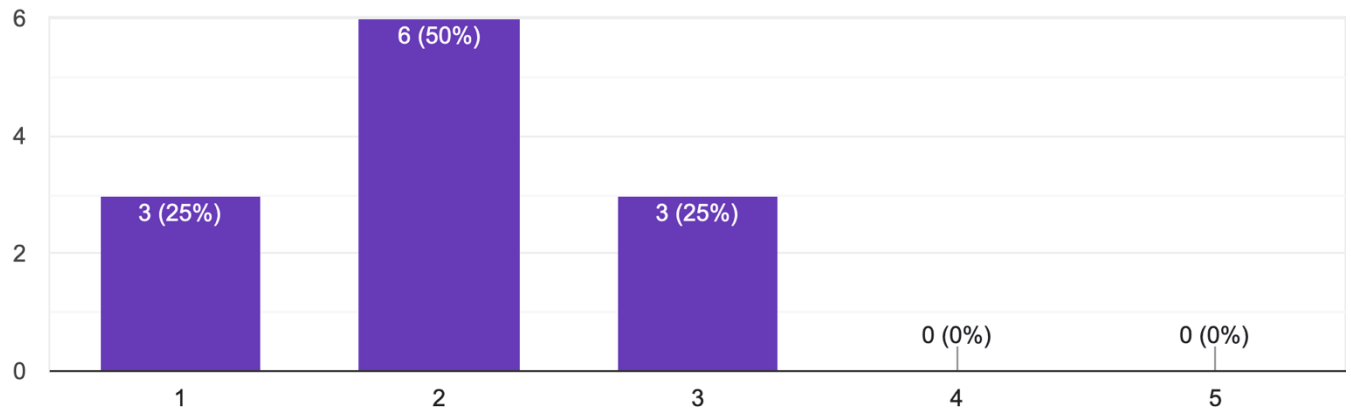
9. I felt very confident using the system

12 responses



10. I needed to learn a lot of things before I could get going with this system

12 responses



10.4 SUS Survey for Complete Extension

System Usability Scale (SUS)

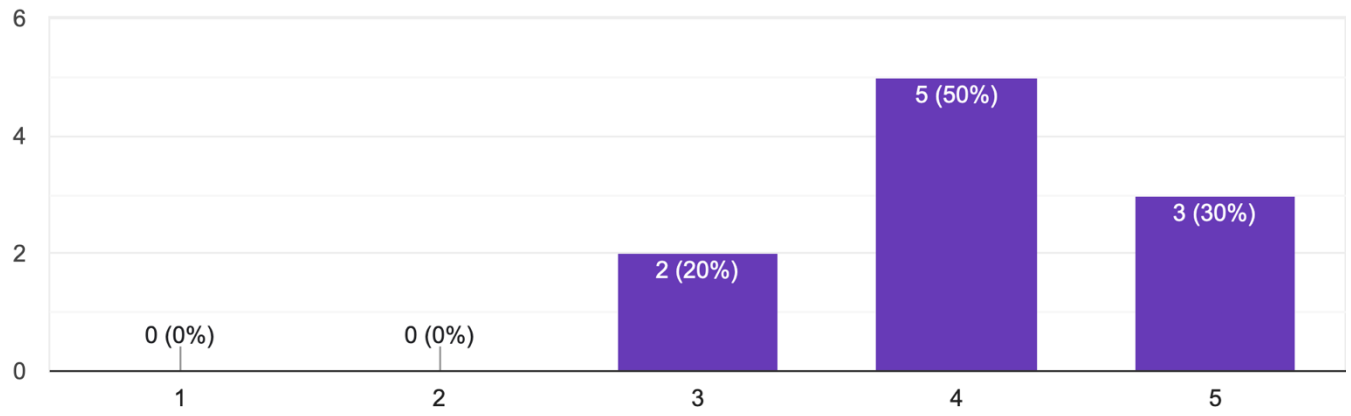
The purpose of this questionnaire is to asses the user experience regarding different aspects of StudySync's Web Extension tool. This test will focus on the extension as a whole.

Please navigate to https://github.com/hashem59/Study-sync-chrome-extesnstion and follow the install instructions to activate the extension on your Google Chrome Browser. Once installed please interact with the extension and get a feel for it.

Once complete, please answer the following questions.

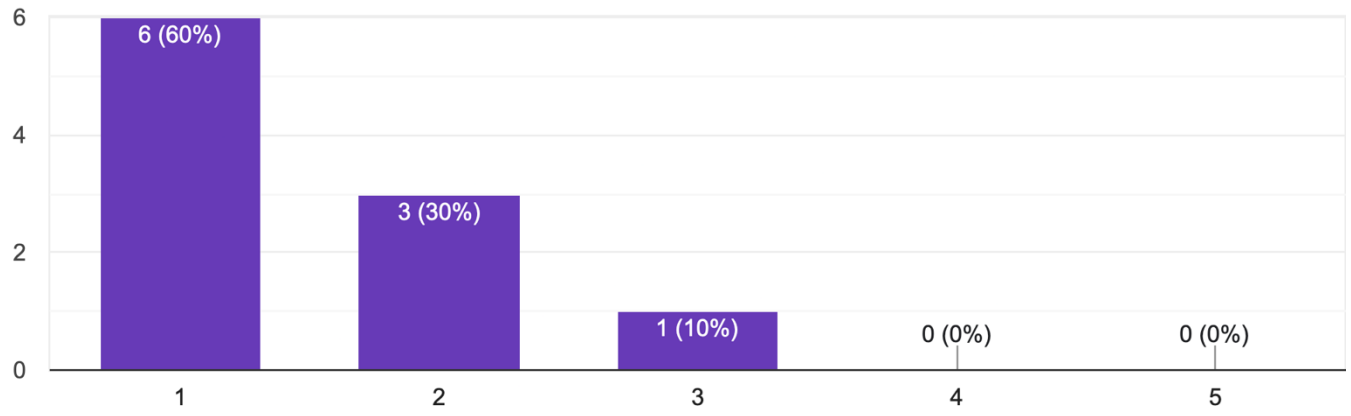
1. I think that I would like to use this system frequently

10 responses



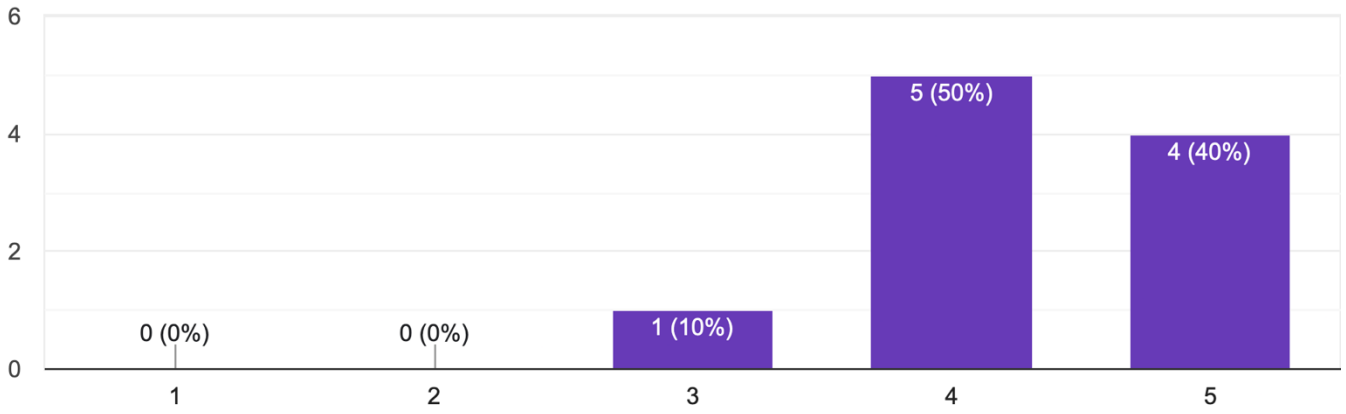
2. I found the system unnecessarily complex

10 responses



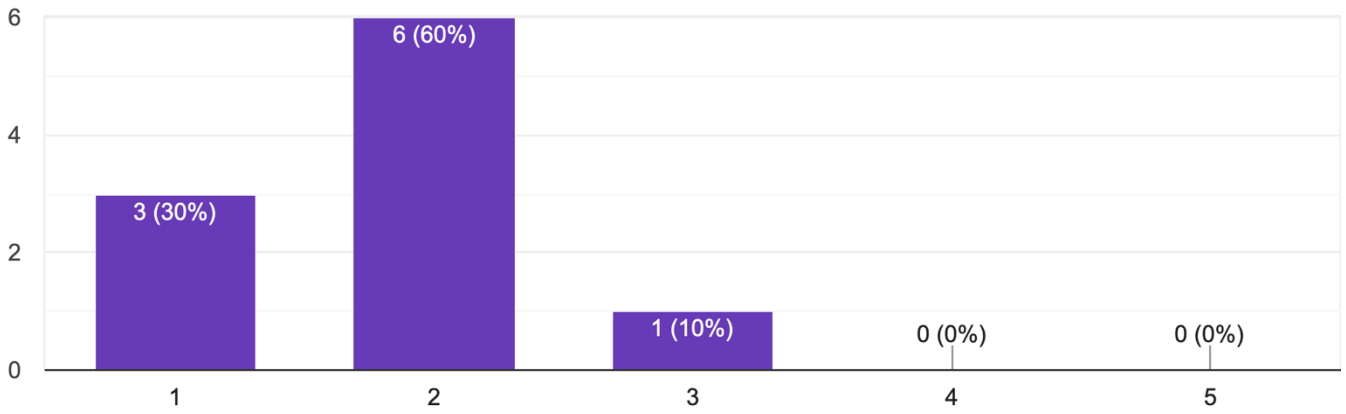
3. I thought the system was easy to use

10 responses



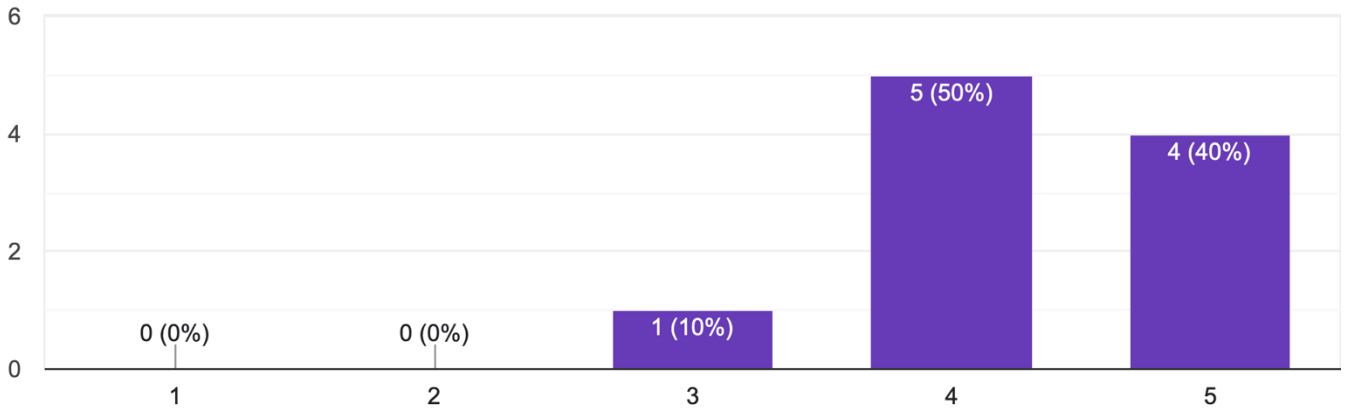
4. I think that I would need the support of a technical person to be able to use this system.

10 responses



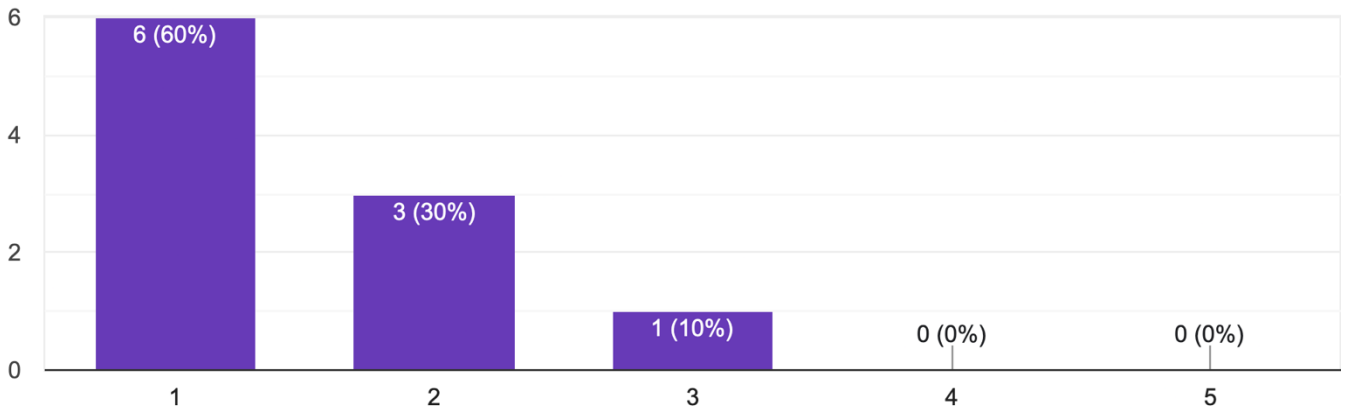
5. I found the various functions in this system were well integrated.

10 responses



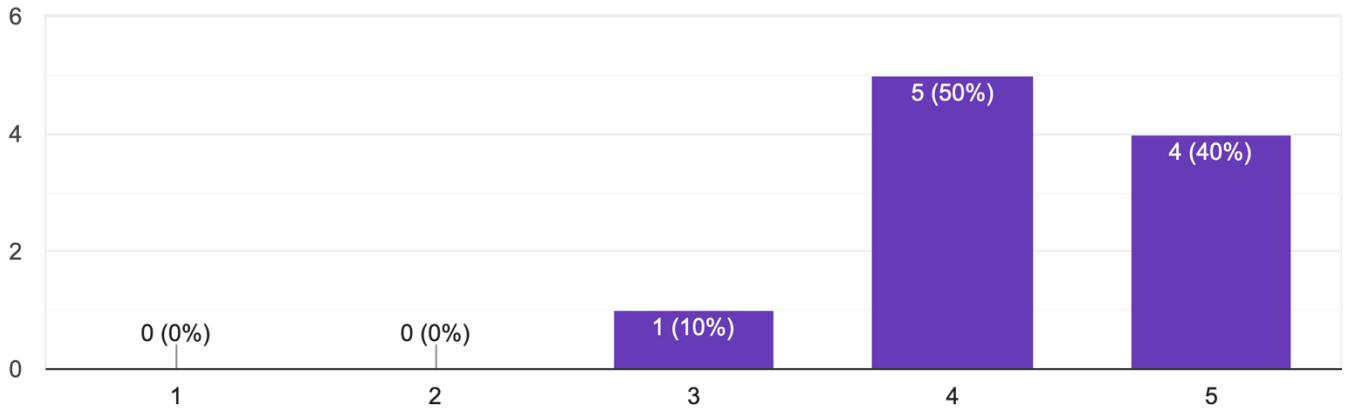
6. I thought there was too much inconsistency in this system.

10 responses



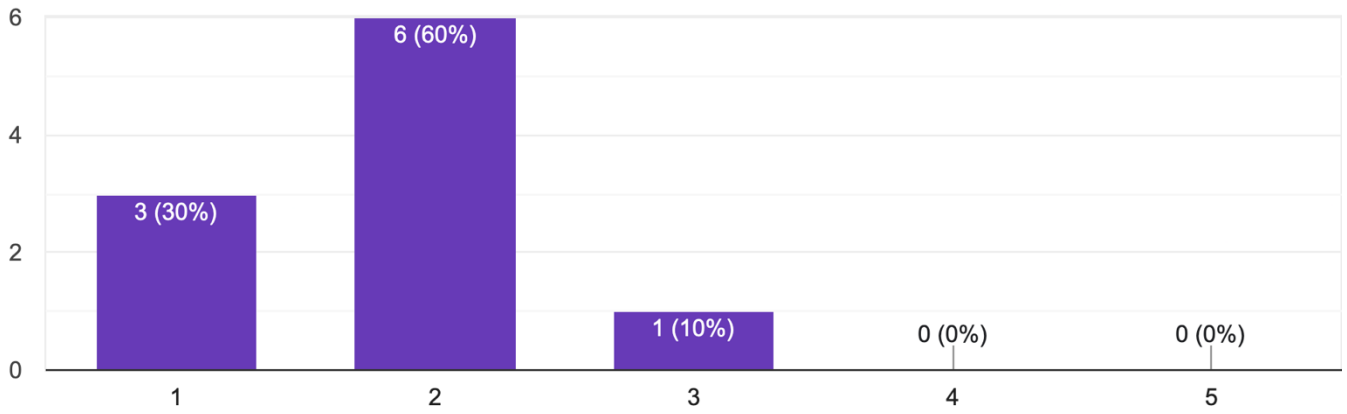
7. I would imagine that most people would learn to use this system very quickly

10 responses



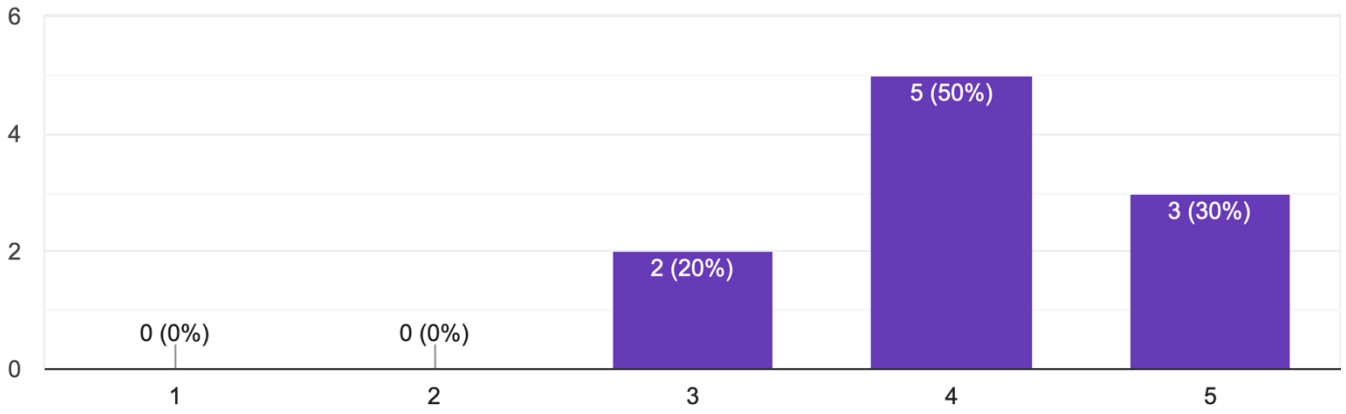
8. I found the system very cumbersome to use

10 responses



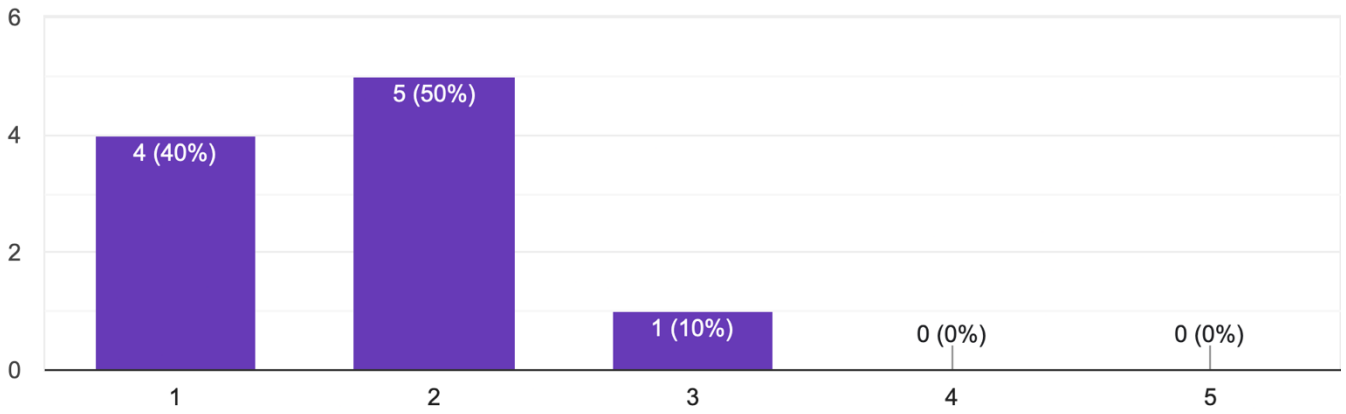
9. I felt very confident using the system

10 responses

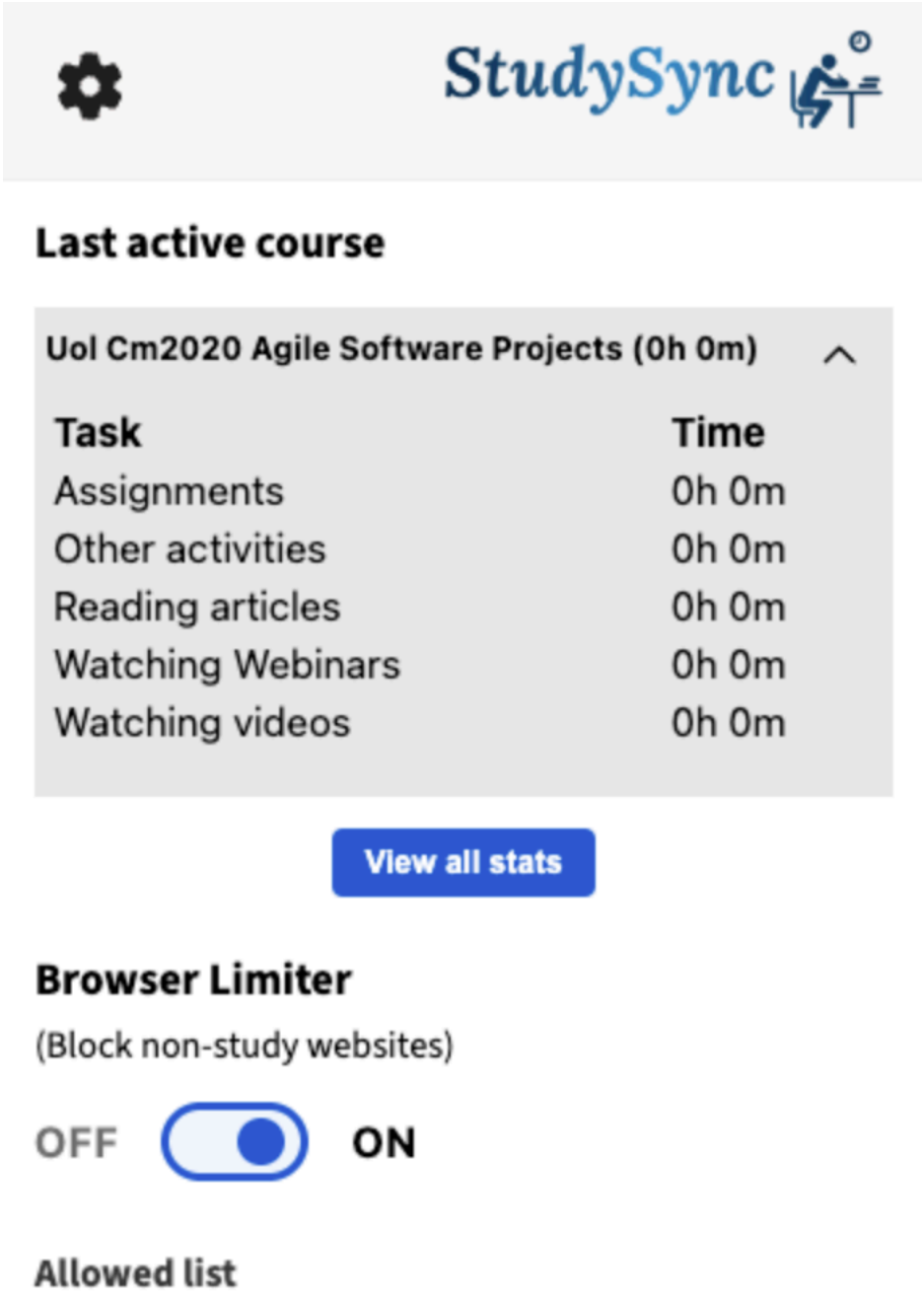


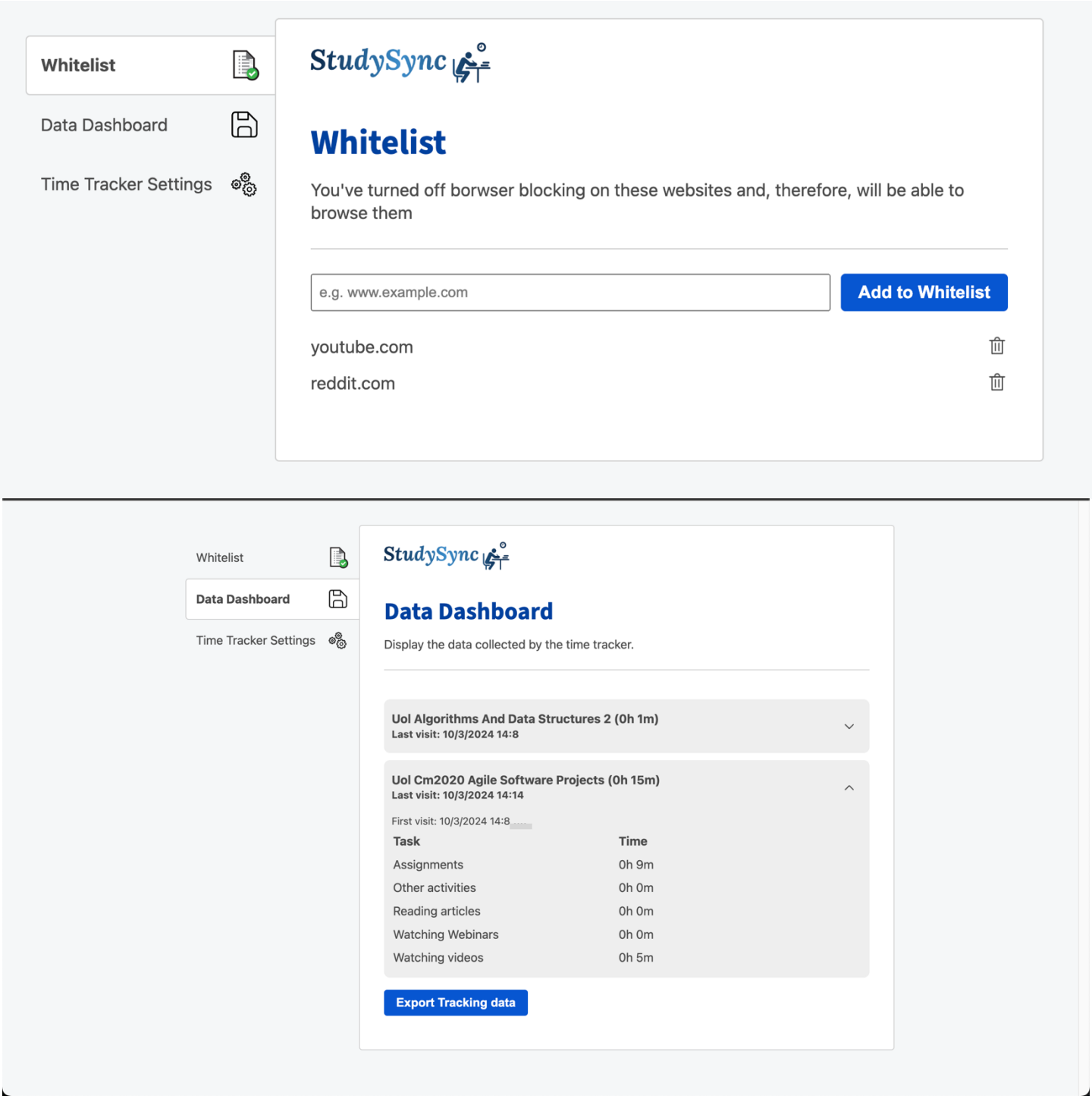
10. I needed to learn a lot of things before I could get going with this system

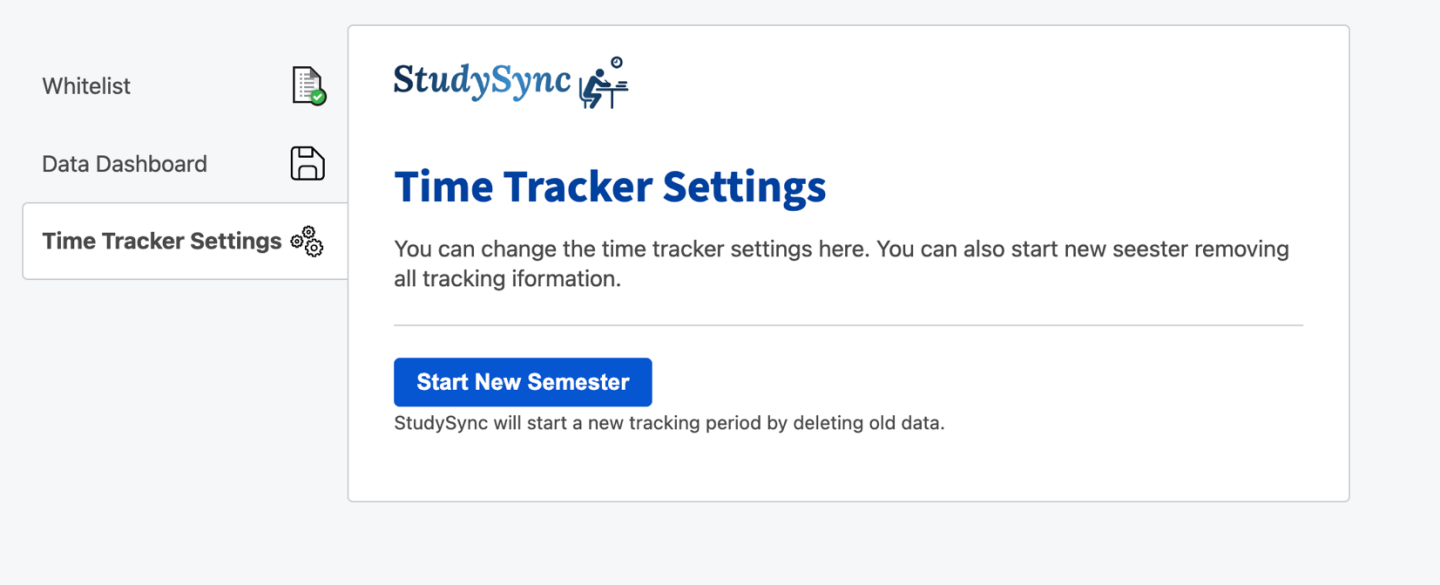
10 responses

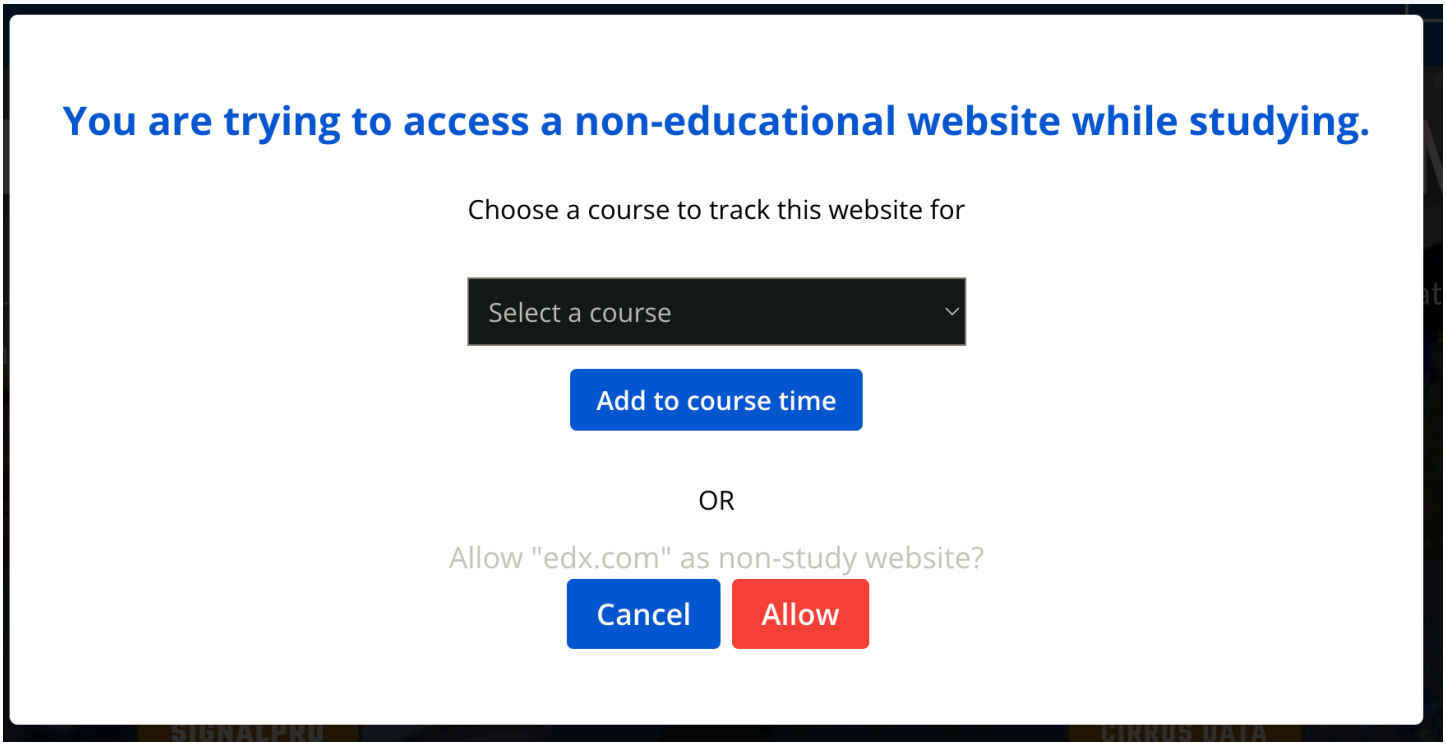


10.5 StudySync UI




Whitelist
Data Dashboard
Time Tracker Settings
StudySync
Whitelist
You've turned off borwser blocking on these websites and, therefore, will be able to browse them
e.g. www.example.com
Add to Whitelist
youtube.com
reddit.com
Whitelist
Data Dashboard
Time Tracker Settings
StudySync
Data Dashboard
Display the data collected by the time tracker.
Uol Algorithms And Data Structures 2 (0h 1m)
Last visit: 10/3/2024 14:8
Uol Cm2020 Agile Software Projects (0h 15m)
Last visit: 10/3/2024 14:14
First visit: 10/3/2024 14:8
Task
Time
Assignments
0h 9m
Other activities
0h 0m
Reading articles
0h 0m
Watching Webinars
0h 0m
Watching videos
0h 5m
Export Tracking data


Whitelist
Data Dashboard
Time Tracker Settings
StudySync
Time Tracker Settings
You can change the time tracker settings here. You can also start new seester removing all tracking iformation.
Start New Semester
StudySync will start a new tracking period by deleting old data.


You are trying to access a non-educational website while studying.
Choose a course to track this website for
Select a course
Add to course time
OR
Allow "edx.com" as non-study website?
Cancel
Allow

11 References

11.1 Security, P. P. (2022, April 22). Local storage versus cookies: Which to use to securely store session tokens. *Pivot Point Security*. https://www.pivotpointsecurity.com/local-storage-versus-cookies-which-to-use-to-securely-store-session-tokens/

11.2 Experience, W. L. in R.-B. U. (n.d.). *10 usability heuristics for user interface design*. Nielsen Norman Group. Retrieved 26 February 2024, from https://www.nngroup.com/articles/ten-usability-heuristics/

11.3 *Coursera | online courses & credentials from top educators. Join for free*. (n.d.). Coursera. Retrieved 3 March 2024, from https://www.coursera.org/learn/uol-cm2010-software-design-and-development/supplement/CDKIu/14-0208-test-procedure-template-files

11.4 *Sus: A retrospective - jux*. (2013, February 7). JUX - The Journal of User Experience. https://uxpajournal.org/sus-a-retrospective/

11.5 Chrome Extension: Getting started 'Hello World' Extension. Chrome for Developers. https://developer.chrome.com/docs/extensions/get-started/tutorial/hello-world

11.6 Chrome Extension: Manifest File Format. Chrome for Developers. https://developer.chrome.com/docs/extensions/reference/manifest

11.7 Chrome Storage: API' Chrome for Developers, https://developer.chrome.com/docs/extensions/reference/api/storage

11.8 Build a Chrome Extension – Course for Beginners, https://www.youtube.com/watch?v=0n809nd4Zu4

11.9 Design the user interface | Extensions. (n.d.). Chrome for Developers. Retrieved 6 January 2024, from https://developer.chrome.com/docs/extensions/develop/ui

11.10 Cipd | swot analysis | factsheets. (n.d.). CIPD. Retrieved 6 January 2024, from https://www.cipd.org/uk/knowledge/factsheets/swot-analysis-factsheet/

11.11 Freedom.to. (n.d.). Freedom: Internet, app and website blocker. Freedom. Retrieved 6 January 2024, from https://freedom.to/

11.12 Web blocker | block unlimited site for free. (n.d.). Retrieved 6 January 2024, from https://www.webblockerextension.com/

11.13 Time tracking software for improved productivity | webwork. (n.d.). WebWork Tracker. Retrieved 6 January 2024, from https://www.webwork-tracker.com

11.14 Chrome extension for effective time tracking. (n.d.). Web Activity Time Tracker. Retrieved 6 January 2024, from https://webtracker.online

11.15 Block site: Site blocker & focus mode. (n.d.). Retrieved 6 January 2024, from https://chromewebstore.google.com/detail/block-site-site-blocker-f/dpfofggmkhdbfcciajfdphofclabnogo

11.16 Toggl: Time tracking software, project planning & hiring tools. (n.d.). Retrieved 6 January 2024, from https://toggl.com/

11.17 Chrome web store—Program policies. (n.d.). Chrome for Developers. Retrieved 6 January 2024, from https://developer.chrome.com/docs/webstore/program-policies

11.18 Terms of use. (n.d.). Coursera. Retrieved 6 January 2024, from https://www.coursera.org/about/terms