Bilkent University
Department of Computer Engineering

Senior Design Project
T2321
Content Guard

Final Report

21902115, Gülin Çetinus

21902896, Zeynep Derin Dedeler

21802215, Bengisu Buket Kardoğan

21901841, Burak Öztürk

22001769, İlayda Zehra Yılmaz

Ayşegül Dündar

Atakan Erdem, Mert Bıçakçı

Cem Çimenbiçer

13.05.2024

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents:

1. Introduction

In an era dominated by the digital realm, social media platforms have become integral agents for information exchange. Among these platforms, Twitter stands out as a dynamic space characterized by its nonstop updates and diverse content. As individuals dive into the world of Twitter, they face the following challenges: navigating the expansive array of content effectively and managing the time devoted to engaging with Twitter. The widespread use of social media, especially on platforms such as Twitter, has given rise to a prevalent issue: the unintentional dissipation of time on topics that may not align with users' primary interests or goals. As individuals traverse the endless stream of tweets, the risk of diverting attention to unrelated or unproductive content becomes increasingly apparent. Examples of this challenge include a student attempting to stay updated on industry trends and being sidetracked by trending memes or unrelated discussions, leading to a loss of valuable study time. Similarly, professionals seeking to leverage Twitter for networking and professional development may encounter challenges. Drawing from research insights, studies such as the one conducted by Gomez-Rodriguez et al. (2014) emphasize the impact of information overload on social media users, highlighting the need for tools and strategies to enhance content relevance and user

productivity [1]. Furthermore, surveys show that two thirds of Twitter users have felt that they receive too many posts, and over half of Twitter users have felt the need for a tool to filter out the irrelevant posts [2] [3].

1.1. Purpose of the System

According to the previous section, we decided to foster a mutually beneficial relationship between users and their Twitter experience. Our project's name is "Content Guard," which comes from its fundamental mission to act as a cautious protector, safeguarding users from the pile of irrelevant or overwhelming content on Twitter. In our project, the aim is to develop an application and browser extension that empowers users to efficiently manage and tailor their Twitter experience by providing functionalities such as content filtering, timer constraints, and keyword-based filtering to enhance user control over their information consumption and optimize the relevance of their feed. In this application, the content consumed by a user will be investigated with the help of artificial intelligence, and the category to which it belongs will be found. According to these findings, users will be able to filter specific categories, track the time spent on consuming different categories, and set several timer constraints. In this way, we will come up with a solution to navigate Twitter content efficiently and ensure a control over social media interaction [3].

1.2. Design Goals

1.2.1. Usability

The user interface should allow easy navigation, intuitive controls, and a seamless user experience in configuring and utilizing the application and browser extension. It should not be too complicated because Content Guard aims to make content consumption easier for users of all ages [3].

1.2.2. Performance

The application's runtime should have optimized content filtering and processing efficiency to minimize latency and provide a responsive experience for users [3].

1.2.3. Reliability

The project to be done should correctly categorize content consumed. The assigned category to tweets should have a high proper accuracy rate.

Frequent or prolonged downtime of ContentGuard's servers should be avoided, as it will become part of users' daily lives.

If an error occurs in the program or if the desired operation cannot be executed, it is essential to provide the user with sufficient and appropriate information [3].

1.2.4. Extendibility

The application and extension should have a modular architecture enabling future updates, enhancements, and the addition of new features to adapt to evolving user needs and platform changes [3].

1.2.5. Privacy and Security

The project should prioritize user privacy by implementing necessary security measures to protect user data and ensure secure communication between the extension and the application server [3].

1.2.6. Scalability

ContentGuard will initially be crafted as a proof-of-concept application; however, our subsequent goal is to broaden its accessibility to a diverse audience. Hence, scalability is crucial to ensure ContentGuard can cater to a large user base [3].

1.3 Definitions, Acronyms, and Abbreviations

ML: Machine learning (ML) is a domain within artificial intelligence (AI) and computer science dedicated to leveraging data and algorithms to enable AI systems to emulate the learning processes observed in humans, thereby enhancing their accuracy over time [4].

NLP, Natural Language Processing, lies at the intersection of linguistics, computer science, and artificial intelligence. It focuses on enabling computers to understand

and analyze human language, effectively processing large volumes of textual data [5].

Chrome Extension: Chrome extensions are compact software applications designed to enhance and tailor users' interactions or productivity while navigating through their Google Chrome web pages. They are crafted using web technologies like HTML, JavaScript, and CSS.

UI: User interface refers to how users interact with an application or a website [6].

KVKK: The Turkish abbreviation for "Kişisel Verileri Koruma Kurumu" is the "KVKK." Established under Law No. 6698, the Personal Data Protection Authority is a public legal entity with administrative and financial autonomy tasked with carrying out duties pertaining to data protection [7].

TC: Test Case refers to the test unit and the number of the test [3].

1.4 Overview

In this report and subsequent sections, we will provide comprehensive insights into our existing and evolving software architecture system, subsystem, and different fields related to our project. This report will give you more detailed information about the ContentGuard project, offering a more comprehensive understanding of our system and workload [3].

2. What We Have Done

We have started our Content Guard project’s creative process by identifying the project’s goal as to improve Twitter’s content relevance to the user’s liking and time management while on Twitter. We have started our search for the existing projects and tools for the subject of a Twitter tool that helps user categorize and filter its content by keyword as well as managing the time while doing this. Not having found any one of our liking we have decided to create a project of our own. We have decided that making our project to be an extension was the most useful and best approach in this regard. Through the extension user would reach the keyword

blockers, category blockers and the times for these blockers using the extension UI. Then we have decided to create a Home page for our extension and add a button that led to our Home page in our extension UI by examining other similar extensions. The purpose of our extension is to manage and tailor their Twitter experience. Then we have names the project “ContentGuard” by voting among the team members because of its role as guarding the content of Twitter users to their liking and their time choices. Defining the functionalities more clearly we have started our research. Then we have created our extension’s UI. After this, we have genuinely implemented our extension with backend and frontend.

We have implemented a Chrome API-based extension that was compatible with browsers with the Chromium engine. This API is used for local storage elements like user settings and extension activities. We have decided and implemented a server that runs on a member's computer, uses JavaScript for communication with the clients, SQLite for databases that are necessary to keep the data, and GPT 3.5 Turbo from OpenAI for NLP tasks that are necessary for blocking. With this we have decided our technologies.

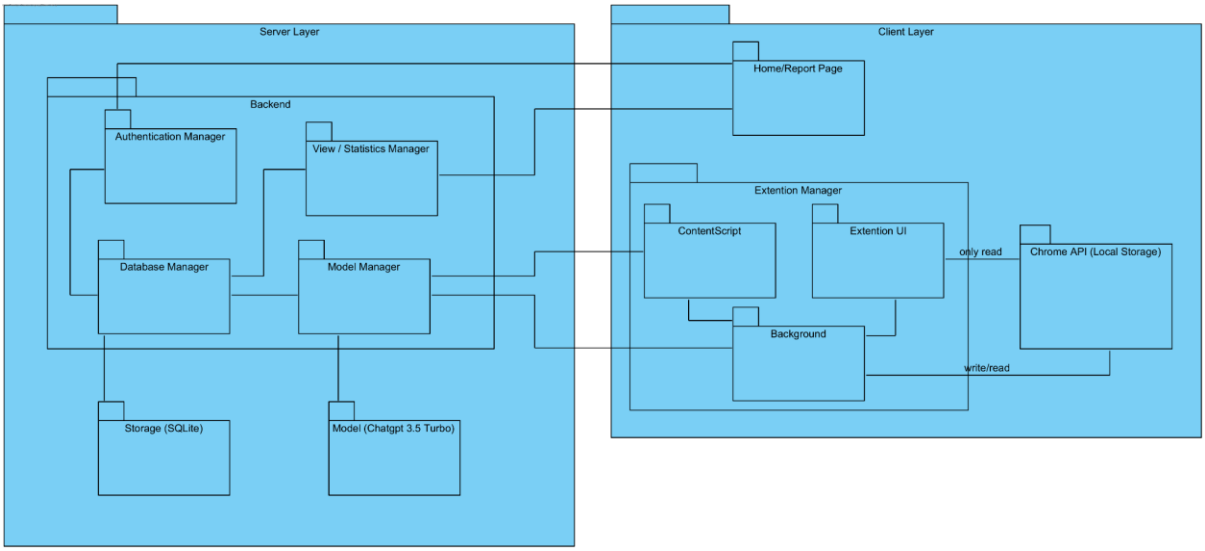
In our project, database management was implemented as two databases that manage user data and reports, hashed with user IDs, using SQLite. In this, we have created an access control system as a single-role user for privacy concerns.

Our Extension Manager oversees extension interfaces, blocking features, and timers, enabling communication between background and content scripts, and handling local storage. The Extention UI is created for user to use attributes like these. Our background uses a content script that is responsible for DOM manipulation classification requests, tweet content extraction, and data transmission to the backend. Using Django and Python in the backend we are creating reports and categorization and blocking attributes. OpenAI 3.5 Turbo as an NLP model for categorization. We are using OAuth for authentication. Using all of these we have the workings of our extension and home page.

Our Home Page, is designed for the user to be able to see the data of their usage of our extension. It reports the keyword, and category blocker’s data in charts and graphs as well as more about tweets that are blocked. It reports to the user their usage of that data. It uses Firebase authentication using Twitter account of the user to log in. And uses cookies to bring the data of the user to this page.

All in all, we have created an extension that effectively categorizes and keyword blocks our content with timers as well as a home page that reports these datas. Other than implementing this project we have also documented reports for this extension with CS 491/2 reports.

3. Final Architecture and Design Details



The final project architecture uses Server-Client architecture that is divided into two layers: Server Layer and Client Layer. The server layer runs on the main computer (a member's computer, which can also be moved to a server with Windows ops) and manages authentication, users, usage statistics, and databases while using the NLP model to serve the clients. The client layer runs on the users' browsers and manages DOM manipulation, restrictive measures (timers and blockers), user settings, reports, and local storage.

3.1 Hardware/Software Mapping

ContentGuard needs a server as a base of operations for the hardware part of the server layer. We decided on the server layer to run on a team member's computer. The choice is made according to financial constraints and technical feasibility. However, the system could easily moved to a cloud service (Google Cloud Platform).

For the software part of the server, JavaScript is used for the most part. This is to make the communication with the client innately JavaScript-based, like every Chrome extension. The database is an SQL-based system (SQLite is determined). As the NLP model, GPT 3.5 Turbo from OpenAI was chosen. This decision was made because of the speed, the computational power of the model, and it's financially more sensible. However, for the future of this project, our own model can be trained, which also won't be dependent on the site Twitter, which can be applied to the future sites that we plan to implement (Threads, Reddit, YouTube, etc.).

There is no hardware on the client side of the project other than an electronic device (PC, Mac, smartphone, etc.) that can run a Chromium-based browser that supports extensions and has an internet connection.

The software part of the Client Layer is entirely in JavaScript (with HTML and CSS for UIs). Again, JavaScript will be used at every opportunity to make communication easier with the extension that must be written in JavaScript.

3.2 Persistent Data Management

There will be two databases at the Server Layer. The user database will keep general user data like username, ID, mail, personal usage information, generated reports, and user statistics. The reports will be hashed with user IDs. and stored separately on a different structure. But both of these use SQLite and are managed by it.

At the Client Layer, chrome will keep all the persistent data in the browser's storage. Chrome API. The data to be kept consists of user settings and the state of the extension's activities from closing the browser until the next browser startup.

3.3 Access Control and Security

Our application will operate with a single user role, designated as the regular user. Regular users are allowed to use the application, and the scope of data access will be limited to their files and data. Each user can access, modify, and delete their data reports, timers, and filters. There will be no interaction between users therefore,

users will not be able to access or modify other people’s data. This limitation is set to meet the users' privacy, and our application does not require user interaction.

Rather than incorporating an admin user interface, we have managed administrative tasks discreetly through a secure backdoor access mechanism. This approach ensures that sensitive administrative functions remain accessible solely to authorized personnel, significantly reducing the potential for unauthorized access or misuse.

Access to the data will be only through the server, including backdoor admin interventions. Consumed content data will be transferred to the server to classify the content under a predefined category, and data will be deleted after achieving that goal. The resulting classification will be transferred back to the client.

Lastly, each user will have access to their data through unique tokens provided during the authentication process. These tokens will be stored in Chrome's cache and utilized as needed. Each token will be distinct and assigned to the user upon each sign-in and subsequently deleted upon logout to ensure security and privacy.

4. Development/Implementation Details

This section provides an overview of the services contained within the previously mentioned layers. Within each layer, there are managers tasked with overseeing various services specific to that layer, and implementation details will be explained accordingly.

4.1 Client Layer

The Client Layer is in charge of the services that run on the client’s browser via the Chromium extension. This layer accounts for UI interfaces, and its actions also deal with local storage. It includes one extension manager to provide these services. It also has a Home/Report Page and a React project website that runs in the client’s local environment.

4.1.1 Extension Manager

The extension manager oversees the extension interfaces, core blocking functionalities, and timers. It facilitates the execution of actions, the exchange of communications between the background and content script modules, and their interaction with local storage mechanisms. This is contained in the content-guard subfile of the project, and it contains a React environment, which is the pop-up UI of the extension.

4.1.1.1 Extention UI:

The Extention UI user interface (UI) serves as a platform through which clients can configure their settings, implement keyword blockers or category filters, add timers to them, and administer them accordingly. It consists of several UI components which run in a React environment. KeywordBlocker.js and CategoryBlocker.js can access the local storage (Chrome API) to read user settings, filter settings, and keywords. These files also communicate with the background script to update the user setting as a result of the user's interaction with the Extension UI.

4.1.1.2 Background:

The Background script maintains records of content scripts and orchestrates the transmission of tweets to the server layer for categorization through communication with the extension manager. Additionally, it monitors settings updates and activations of keyword blockers and category filters, triggering events to be sent to the respective content scripts or broadcast to all the content scripts. The background script also oversees timing functionalities, calculating the activation instances of blockers or filters. It also updates the user settings to the local storage (Chrome API).

4.1.1.3 Content Script:

The Content script is responsible for executing DOM manipulation functionalities. It extracts tweet content and forwards it to the Backend to obtain the corresponding tweet category. It also manages classification requests by buffering them with queues to streamline the process. Subsequently, it collects blocked tweet information and transmits it to the Backend. Additionally, it responds to requests

originating from the background script and fulfills the requested events. Developed in JavaScript, it directly interacts with the HTML structure of Twitter. Moreover, the Content script dispatches data obtained from executing these events to the backend for further report processing.

4.1.2 Home/Report Page

The Home/Reports Page serves as an interface facilitating the viewing of provided reports and the submission of requests for new report generation. This interface presents tables and charts to visualize the client's interaction within specified categories on Twitter, the keywords they've blocked, categories they decided to block, and what and how many categories they interacted with. It is a React project that is currently employed locally, but with the provided server, it can also be deployed to a wireless connection, which also enables its communication with the distributed Backend. This page communicates with the Backend for the authentication of the users and getting their data for the reports that will be showcased on this page. This page also manages the cookies and interacts with Firebase for authentication. Firebase communicates with Twitter API and creates the related cookies according to the permissions. With these cookies, our Home/Report Page handles navigating to the dashboard and also sends user information to the Backend.

4.1.3 Chrome API

The local storage system utilizes Google Storage to persistently store user settings filters with their corresponding categories and keywords, as viewed by the client. Additionally, it serves as a repository for temporarily storing user’s account information.

4.2 Server Layer

The Server layer oversees authentication procedures, user management, tracking usage statistics, and conducting database operations. Additionally, it is tasked with employing the NLP model for tweet classification purposes.

4.2.1 Backend

The backend system assumes responsibility for receiving requests from the content script for tweet categorization, responding to these requests, and coordinating them for multiple users. It retrieves activity data from content scripts, generates reports, and stores them in the database, transmitting these reports to the local layer on the Home/Report Page. Additionally, it offers various services, including statistical computations, database management, model administration, and user-related operations. This backend system is situated within the server layer infrastructure. Where it is contained in our project under the content_guard_server subfile. It uses Django to store the models, and it also utilizes Django functionalities. It is implemented in Python and uses various libraries of Python.

4.2.1.1 Authentication Manager:

The Authentication Manager handles user authentication securely. It integrates OAuth, manages the authentication flow, and securely stores access tokens. It also handles token renewal error handling and prioritizes security, ensuring users can securely log in to their Twitter accounts through the extension. This is also the backend part that communicates with the House/Report Page for the authentication.

4.2.1.2 View/Statistics Manager:

The View/Statistics Manager oversees the analysis of reports, executing essential calculations, and conducting thorough analyses. It also seamlessly interfaces with the database manager to store reports and associated calculations. It is also the part that communicates with the Home/Report page to send report data upon request and further report-related functionalities.

4.2.1.3 Database Manager:

This manager oversees database operations, providing essential database modification and access functionalities. It enables efficient interaction with stored data, facilitating tasks like insertion, retrieval, updating, and deletion. Additionally, it

ensures data security and integrity, implementing necessary measures to safeguard sensitive information and comply with regulations.

4.2.1.4 Model Manager:

The Model Manager interfaces with the external NLP model (ChatGpt 3.5 Turbo), handling the preprocessing of tweet data to align it with the model's requirements. It also handles the model's responses and communicates with the database manager to store related tweet category types with tweet IDs. If the exact tweet is requested again, it can retrieve the information from the database.

4.2.2 Storage (SQLite)

This database stores individual user data, including unique user IDs and relevant identification details such as Twitter account names. Additionally, it serves as a repository for reports and statistics associated with each user, facilitating comprehensive data management and retrieval functionalities.

4.2.3 Model (ChatGpt 3.5 Turbo)

In the Django application's workflow for tweet content categorization, the `TweetView` class orchestrates the process upon receiving HTTP POST requests containing tweet data. Leveraging the capabilities of ChatGPT 3.5 Turbo, the `categorize_tweet` method is invoked to categorize the tweet content efficiently. Constructing a message with the tweet content, this method communicates with ChatGPT 3.5 Turbo via its API endpoint, prompting the model to swiftly process the input and generate a response containing the predicted category. This seamless integration enables real-time interaction between the Django application and ChatGPT 3.5 Turbo, facilitating quick and responsive categorization of tweets. As a result, the model's optimized latency ensures that tweet categorization occurs rapidly, enhancing the overall user experience in real-time applications like the Django web application.

5. Test Cases and Results

Test ID	TC-1	**Category**	Functional	**Severity**	Major
Objective	Test case for registering using Twitter without the Twitter signed-in browser				
Steps	1. Click the “Register with Twitter” button. 2. Enter the valid credentials (username/email and password) to the opened pop-up or new tab window. 3. Click the “Authorize” button to authorize the extension to use for registering. 4. Verify the user is registered.				
Expected	1. The extension should open a new pop-up for Twitter authentication. 2. Valid Twitter credentials should result in a successful login and registration. 3. During the registration, extensions should function without crashing or freezing.				
Date-Result	03/2024 - Pass				

Test ID	TC-2	**Category**	Functional	**Severity**	Major
Objective	Test case for registering using Twitter with the Twitter signed-in browser				
Steps	1. Click the “Register with Twitter” button. 2. In the opened pop-up or new tab window, verify the account to be registered as the signed-in account in the browser. 3. Click the “Authorize” button to authorize the extension to use for registering. 4. Verify the user is registered.				

Expected	1. The extension should open a new pop-up for Twitter authentication. 2. Valid Twitter credentials should result in a successful login and registration. 3. During the registration, extensions should function without crashing or freezing.
Date-Result	03/2024 - Pass

Test ID	TC-3	**Category**	Functional	**Severity**	Major
Objective	Test case for signing in using Twitter without the Twitter signed in in the browser				
Steps	1. Click the “Sign In with Twitter” button. 2. Enter the valid credentials (username/email and password) to the opened pop-up or new tab window. 3. Verify the user is signed in.				
Expected	1. The extension should open a new pop-up for Twitter authentication. 2. Valid Twitter credentials should result in a successful login. 3. During the sign-in, extensions should function without crashing or freezing.				
Date-Result	03/2024 - Pass				

Test ID	TC-4	**Category**	Functional	**Severity**	Major
Objective	Test case for signing in using Twitter with the Twitter signed in in the browser				

Steps	1. Click the “Sign In with Twitter” button. 2. In the opened pop-up or new tab window, verify the account to be signed in as the signed-in account in the browser. 3. Verify the user is signed in.
Expected	1. The extension should open a new pop-up for Twitter authentication. 2. Valid Twitter credentials should result in a successful login. 3. During the sign-in, extensions should function without crashing or freezing.
Date-Result	03/2024 - Pass

Test ID	TC-5	**Category**	Functional	**Severity**	Major
Objective	To verify the proper functionality of the delete account feature in the PostgreSQL database.				
Steps	1. Retrieve user information from the database's user table. 2. Delete the user's account from the website application. 3. Verify the absence of the user's information in the user table. 4. If applicable, check for any associated binary data stored in the database. 5. Ensure the removal of any password reset requests associated with the user.				
Expected	1. The user's information should no longer be present in the user table. 2. If applicable, any associated binary data should be properly removed or flagged as deleted. 3. Password reset requests for the user should no longer be present.				

Date-Result	03/2024 - Pass

Test ID	TC-6	**Category**	Functional	**Severity**	Major
Objective	To verify that the user can successfully set filtering options for keyword blocking.				
Steps	1. Open the extension interface by clicking on the extension icon. 2. Within the extension interface, locate and navigate to the "Blockers" section. 3. Identify and navigate to the "Keyword Blocker" option. 4. Set filtering options for keyword blocking by either entering keywords or selecting existing keywords from the list to block. 5. Save the changes made by clicking "Apply". 6. Verify that the entered keywords are successfully blocked.				
Expected	1. Keywords entered for blocking are saved successfully. 2. Upon accessing a page containing any of the blocked keywords, the content related to those keywords is blocked or hidden.				
Date-Result	03/2024 - Pass				

Test ID	TC-7	**Category**	Functional	**Severity**	Major
Objective	To verify that the user can successfully set filtering options for category blocking.				

Steps	1. Open the extension interface by clicking on the extension icon. 2. Within the extension interface, locate and navigate to the "Blockers" section. 3. Identify and navigate to the "Category Blocker" option. 4. Set filtering options for category blocking by selecting categories to block. 5. Save the changes made by clicking "Apply". 6. Verify that the selected categories are successfully blocked.
Expected	1. Categories selected for blocking are saved successfully. 2. Upon accessing a page containing content from the blocked categories, the content is blocked or hidden.
Date-Result	03/2024 - Pass

Test ID	TC-8	**Category**	Functional	**Severity**	Major
Objective	To verify that the user can successfully set filtering options for a timer.				
Steps	1. Open the extension interface by clicking on the extension icon. 2. Within the extension interface, locate and navigate to the "Blockers" section. 3. Identify and navigate to the "Timer" option. 4. Set filtering options for the timer by specifying a duration and start delay for blocking content. 5. Save the changes made by clicking "Apply". 6. Verify that the timer functions as expected, blocking content for the specified duration.				
Expected	1. The timer settings are saved successfully. 2. Content is blocked according to the specified duration after start delay, and access is restored after the timer expires.				

Date-Result	03/2024 - Pass

Test ID	TC-9	**Category**	Functional	**Severity**	Major
Objective	To verify that the user can successfully set multiple filtering options.				

Steps	1. Open the extension interface by clicking on the extension icon.
	2. Within the extension interface, locate and navigate to the "Blockers" section.
	3. Within the "Blockers" section, identify and navigate to the following filtering options:
	  1. Keyword Blocker
	  2. Category Blocker
	  3. Timer
	4. Set filtering options for each of the above options separately.
	5. Save the changes made for each filtering option by clicking "Apply".
	6. Verify that multiple filtering options are enabled simultaneously.
	7. Test by accessing Twitter and observing the application's behavior with the applied filtering options.				

Expected	1. Users can navigate to the "Blockers" section within the extension interface without difficulty. 2. Filtering options for keyword blocker, category blocker, and timer are clearly presented and accessible within the "Blockers" section. 3. Users are able to set and customize filtering options for each category separately. 4. Changes made to filtering options are saved successfully without errors. 5. Multiple filtering options work together seamlessly, ensuring that all specified filters are applied simultaneously when accessing Twitter.				

Date-Result	03/2024 - Pass

Test ID	TC-10	**Category**	Functional	**Severity**	Major
Objective	To verify that the ContentGuard extension can effectively filter tweets from the tweet feed based on the user's specified filtering settings.				
Steps	1. Open the extension interface by clicking on the extension icon. 2. Navigate to the "Blockers" section within the extension interface. 3. Set filtering options for keyword blocker, category blocker, and timer according to user preferences. 4. Save the changes made to the filtering settings. 5. Access the Twitter feed within the browser. 6. Observe the tweet feed and verify that tweets that match the specified filtering settings are filtered out from the feed. 7. Verify that tweets that do not match the specified filtering settings are displayed as usual.				
Expected	1. Users can successfully set filtering options for keyword blocker, category blocker, and timer within the extension interface. 2. Changes made to the filtering settings are saved without errors. 3. When accessing the Twitter feed, tweets that match the specified filtering settings are filtered out and not displayed in the feed. 4. Tweets that do not match the specified filtering settings are displayed as usual, ensuring that relevant content is still accessible to the user.				
Date-Result	03/2024 - Pass				

Test ID	TC-11	**Category**	ML	**Severity**	Major
Objective	To verify the ML model correctly identifies the category of a group of tweets				
Steps	1. Enter a set of verification data to the input file of the ML model. 2. Run the ML model with the indicated inputs. 3. Compare the verification data's result that is human-verified and the ML model's result				
Expected	The ML model needs to get a high accuracy result for the given tweets				
Date-Result	03/2024 - Pass				

Test ID	TC-12	**Category**	Non ML	**Severity**	Major
Objective	To verify the ML model responds in a timely matter				
Steps	1. Run the ML model with test data 2. Record the time to read the test data and return the response				
Expected	The runtime of the ML side of the project should be less than 100 milliseconds for the users to have a seamless experience.				
Date-Result	03/2024 - Fail ~3.07 seconds per tweet for the first 100 tweets in this database: [10]				

Test ID	TC-13	**Category**	Functional	**Severity**	Major
Objective	To test if the user's inputted filter is correctly added to the filter list				
Steps	1. Open the extension window 2. Click to the button to see the filters 3. Click to the button to add a filter 4. Choose filter type (keyword/topic) 5. Enter the filter				
Expected	The added filter should be added to the filter list and stored. The user can see the newly added filter via the filter page on the extension.				
Date-Result	03/2024 - Pass				

Test ID	TC-14	**Category**	Functional	**Severity**	Major
Objective	To test if the blank filter is allowed				
Steps	1. Open the extension window 2. Click to the button to see the filters 3. Click to the button to add a filter 4. Choose filter type (keyword/topic) 5. Enter the filter as "" or " "				
Expected	The entered filter should not be added to the filter list and the user needs to see a warning text.				
Date-Result	03/2024 - Pass				

Test ID	TC-15	**Category**	Non Functional	**Severity**	Moderate
Objective	To test if Twitter feed lags while scrolling with the extension's executing speed.				
Steps	1. Open Twitter. 2. Scroll really fast.				
Expected	There should not be any tweets peeking from filters, where it is shown for a small amount of time then gets hidden.				
Date-Result	03/2024 - Pass				

Test ID	TC-16	**Category**	Functional	**Severity**	Moderate
Objective	User defined timer constraints for filtering content types cannot cause integer overflow				
Steps	1. Go to the timer page in the extension page. 2. Choose a filter to add extension. 3. Set the time for a really big number that causes integer overflow.				
Expected	Maximum allowed duration of a timer should be a limited, non-problematic value.				
Date-Result	03/2024 - Pass (100 hours = 360.000.000 milliseconds can be exactly represented by JavaScript's double-precision floating-point numbers) [11]				

Test ID	TC-17	**Category**	Functional	**Severity**	Moderate

Objective	To test the timer, it should be able to continue when it comes to an unexpected halt
Steps	1. Add a timer for a filter 2. Before the timer runs out, force quit the browser. 3. Reopen Twitter to check if the filter continues to work.
Expected	The filter should continue with the current time not resetted.
Date-Result	03/2024 - Pass

Test ID	TC-18	**Category**	Functional	**Severity**	Moderate
Objective	To test the timer, it should be able to continue when the user changes time zones				
Steps	1. Add a timer for a filter 2. Before the timer runs out, force quit the browser. 3. Reopen Twitter to check if the filter continues to work.				
Expected	The filter should continue with the current time not resetted or changed.				
Date-Result	03/2024 - Pass (Time zones are irrelevant to the code)				

Test ID	TC-19	**Category**	Functional	**Severity**	Moderate
Objective	To test if the user with no data can generate a report without problems				

Steps	1. Register to the extension. 2. Without ever browsing Twitter, go to the extension page. 3. Go to creating a report page. 4. Generate report
Expected	The user should be able to create a report with no browsing, filter or timer data.
Date-Result	03/2024 - Pass

Test ID	TC-20	**Category**	Back end	**Severity**	Major
Objective	To test if the backend server handles unexpected requests gracefully.				
Steps	1. Send a malformed request to the backend server 2. Check server logs for error handling and response status.				
Expected	The back end server should respond with an appropriate error message and status code.				
Date-Result	03/2024 - Pass				

Test ID	TC-21	**Category**	Back end	**Severity**	Major
Objective	To verify backend server stability under heavy load.				
Steps	1. Send a large number of concurrent requests to the backend server. 2. Monitor server response times and resource utilization.				

Expected	The backend should handle the load gracefully without crashing or significantly slowing down.
Date-Result	03/2024 - Fail (Server is currently running in our relatively weak personal computers. For large loads, more processing power and memory is needed.)

Test ID	TC-22	**Category**	Back end	**Severity**	Major
Objective	To test database backup and recovery procedures.				
Steps	1. Simulate a database failure or corruption. 2. Attempt to restore the database from backup.				
Expected	The database should be successfully restored to its previous state without data loss.				
Date-Result	03/2024 - Fail (Currently, there is no automatic data loss prevention in case of a failure.)				

Test ID	TC-23	**Category**	Functional	**Severity**	Major
Objective	To test extension behavior on different operating systems.				
Steps	1. Install and run the extension on various operating systems (Windows, macOS, Linux). 2. Test extension functionality and performance.				
Expected	The extension should work consistently across different operating				

	systems without platform-specific issues.
Date-Result	03/2024 - Pass (Extension runs on every OS that has Chrome browser.)

Test ID	TC-24		Functional		Major
Objective	To test website compatibility across different browsers.				
Steps	1. Access the website using various browsers (Chrome, Firefox, Safari, etc.). 2. Test extension functionality and performance.				
Expected	The website should function consistently and display correctly across different browsers.				
Date-Result	03/2024 - Pass Since it utilizes Chrome API and that is specific to Chromium-based browsers such as Google Chrome, Microsoft Edge, Opera, and others that use the Chromium engine, it will work on these browsers.				

Test ID	TC-25	**Category**	Functional	**Severity**	Major
Objective	To verify extension behavior when multiple instances are running simultaneously.				
Steps	1. Open multiple browser windows or tabs with the extension installed. 2. Perform actions in one instance and observe the impact on others.				
Expected	The extension should handle multiple instances gracefully without data corruption or unexpected behavior.				
Date-Result	03/2024 - Pass Background can deal with multiple tabs also backend can sustain all requests because runs asynchronously				

Test ID	TC-26	**Category**	Functional	**Severity**	Major
Objective	To test extension performance impact on browser startup time.				
Steps	1. Measure browser startup time with and without the extension enabled.				
Expected	The extension should not significantly increase browser startup time.				
Date-Result	03/2024 - Pass				

Test ID	TC-27	**Category**	ML	**Severity**	Major
Objective	To test the ML model's ability to handle ambiguous tweets.				
Steps	1. Input tweets with ambiguous or mixed content to the ML model.				
Expected	The ML model should handle ambiguous tweets gracefully, providing accurate categorization or flagging them for further review.				
Date-Result	03/2024 - Pass It identifies categories correctly.				

Test ID	TC-28	**Category**	ML	**Severity**	Major
Objective	To test the ML model's performance with real-time tweet streams.				
Steps	1. Input a continuous stream of tweets to the ML model.				
Expected	The ML model should process real-time tweet streams efficiently, providing timely categorization results without delays.				
Date-Result	03/2024 - Pass It works on a continuous tweet stream because it was designed to work in that situaion.				

Test ID	TC-29	**Category**	Functional	**Severity**	Moderate

Objective	To test the extension's compatibility with browser extensions.
Steps	1. Install various browser extensions alongside the Twitter management extension. 2. Test the functionality of both extensions simultaneously.
Expected	The extension should work seamlessly alongside other browser extensions without causing conflicts or performance issues.
Date-Result	03/2024 - Pass It works with other extentions

Test ID	TC-30	**Category**	ML	**Severity**	Moderate
Objective	To test the robustness of the ML model against noisy data.				
Steps	1. Inject noisy or irrelevant data into the ML model.				
Expected	The ML model should demonstrate resilience to noisy data and produce accurate results.				
Date-Result	03/2024 - Pass The categorization processes works accordingly.				

Test ID	TC-31	**Category**	Stress	**Severity**	Major
Objective	To test system response time under maximum load.				
Steps	1. Gradually increase concurrent user requests to the system maximum. 2. Measure response time for tweet extraction, categorization, and rendering.				
Expected	Swift and efficient tweet extraction from the user's DOM. Smooth and accurate categorization process. Prompt rendering of categorized tweets in extension interface. Overall system maintains acceptable response times, ensuring smooth user experience under maximum load.				

Date-Result	03/2024 - Pass

Test ID	TC-32	**Category**	ML	**Severity**	Major
Objective	To test the ML model's sensitivity to different dialects.				
Steps	1. Input tweets in various dialects to the ML model.				
Expected	The ML model should accurately classify tweets across multiple dialects.				
Date-Result	03/2024 - Pass Keyword block looks for the keyword that is entered and this does not change the result in different dialects.				

Test ID	TC-33	**Category**	Stress	**Severity**	Major
Objective	To test system recovery time after a catastrophic failure.				
Steps	1. Simulate a catastrophic failure, such as a complete system outage, which interrupts the extraction and categorization processes of the Chrome extension for Twitter. 2. Measure the time taken to detect the failure and initiate recovery procedures. 3. Identify the steps taken to restore the system to full functionality, including re-establishing connections with the server hosting the NLP model.				
Expected	1. The system promptly detects the catastrophic failure and initiates automatic recovery procedures. 2. Upon detection, the system triggers alerts or notifications to relevant personnel to expedite the recovery process. 3. The system efficiently restores functionality, including re-establishing connections with the server hosting the NLP model. 4. The recovery process is swift and effective, minimizing downtime and ensuring that the Chrome extension resumes its extraction and categorization tasks without significant delays. 5. Overall, the system demonstrates robust disaster recovery				

	capabilities, guaranteeing the continuity of service and preserving the user experience even in the face of major disruptions.
Date-Result	03/2024 - Fail This was not implemented.

Test ID	TC-34	**Category**	Security	**Severity**	Major
Objective	To test user session expiration and logout functionality.				
Steps	1. Log in to the extension with a user account. 2. Wait for the session to expire due to inactivity. 3. Attempt to perform actions requiring authentication.				
Expected	1. After the session expires, the user should be automatically logged out and prompted to re-authenticate for security reasons.				
Date-Result	03/2024 – Pass Because Content Guard uses the OAuth session does not expire and if the cookies that Firebase manages remained system remembers the user and allow login. And if the cookies are disabled it requires authentication again.				

Test ID	TC-35	**Category**	Model	**Severity**	Moderate
Objective	To test the ML model's generalization capability across different Twitter users and profiles.				
Steps	1. Train the ML model with data from various Twitter users with diverse tweeting habits.				
Expected	The ML model should generalize well and accurately classify				

	tweets from different users
Date-Result	03/2024 – Fail Our own model is not implemented Chatgpt 3.5 Turbo was saficient enough. But training a model could be required for other sites when future site versions implemented.

Test ID	TC-36	**Category**	Model	**Severity**	Major
Objective	To verify the ML model's ability to adapt to changing trends in Twitter content.				
Steps	1. Train the ML model with historical Twitter data. 2. Test the model's performance with recent Twitter data.				
Expected	The ML model should accurately classify tweets even with evolving trends and topics.				
Date-Result	03/2024 - Fail Our own model is not implemented Chatgpt 3.5 Turbo was saficient enough. But training a model could be required for other sites when future site versions implemented.				

Test ID	TC-37	**Category**	Performance	**Severity**	Major
Objective	To test the extension's memory usage over a long period.				
Steps	1. Monitor the extension's memory usage while it's active for an extended period (e.g., several hours).				
Expected	The extension's memory usage should remain stable over time,				

	without significant memory leaks or excessive consumption.
Date-Result	03/2024 – Pass

Test ID	TC-38	**Category**	Performance	**Severity**	Major
Objective	To test the responsiveness of the extension interface under heavy load.				
Steps	1. Simulate simultaneous user interactions with the extension interface.				
Expected	The extension interface should remain responsive and fluid even under heavy concurrent usage.				
Date-Result	03/2024 – Pass Relatively to our current hardware that runs backend it responding good for the our test users. Deploying to server would improve this				

Test ID	TC-39	**Category**	Performance	**Severity**	Major
Objective	To test the extension's performance on low-end devices.				
Steps	1. Install the extension on low-end devices with limited hardware resources.				
Expected	The extension should remain functional and responsive on low-end devices without significant performance degradation.				

Date-Result	03/2024 – Pass

Test ID	TC-40	**Category**	Model	**Severity**	Major
Objective	To test the accuracy of tweet classification using ML.				
Steps	1. Input a variety of tweets into the ML model and compare classifications with ground truth.				
Expected	The ML model should accurately classify tweets into relevant categories with high precision and recall.				
Date-Result	03/2024 – Pass Chatgpt 3.5 Turbo is very good at classifications				

Test ID	TC-41	**Category**	UI	**Severity**	Major
Objective	To test the responsiveness of UI elements across different screen sizes.				
Steps	1. Resize the browser window and observe UI element responsiveness.				
Expected	UI elements should adapt smoothly to different screen sizes without overlapping or becoming too small.				
Date-Result	03/2024 - Pass				

Test ID	TC-42	**Category**	Database	**Severity**	Moderate
Objective	To test database indexing and query optimization.				
Steps	1. Execute a series of complex queries against the database, simulating typical usage scenarios of the Chrome extension for Twitter. These queries may involve retrieving categorized tweets, analyzing user engagement statistics, or accessing historical data. 2. Measure the performance of each query, recording metrics such as execution time, resource consumption, and query throughput. 3. Analyze the impact of database indexing on query performance by comparing the execution times of queries with and without indexing. 4. Identify any bottlenecks or inefficiencies in query execution, pinpointing areas where optimization may be necessary.				
Expected	1. Database queries should execute efficiently, with acceptable response times and minimal resource consumption, ensuring smooth operation of the Chrome extension. 2. Indexing should significantly improve query performance where applicable, reducing the time required to retrieve relevant data from the database. 3. Queries involving commonly accessed data, such as tweet categories or user engagement statistics, should benefit from indexing, resulting in faster retrieval times and enhanced user experience. 4. Any identified bottlenecks or inefficiencies should be addressed through query optimization techniques, such as query rewriting, database schema redesign, or indexing adjustments, to improve overall database performance.				
Date-Result	03/2024 - Pass				

Test ID	TC-43	**Category**	Stress	**Severity**	Major
Objective	To test database performance under high concurrency.				
Steps	1. Simulate a high number of concurrent database transactions. 2. Measure database response times and throughput.				
Expected	The database should handle concurrent transactions efficiently without significant performance degradation.				
Date-Result	03/2024 – Pass Relatively to our currently hardware that runs backend it responding good for the our test users. Deploying to server would improve this				

Test ID	TC-44	**Category**	Stress	**Severity**	Major
Objective	To test password strength requirements during user registration.				
Steps	1. Attempt to register with a weak password (e.g., fewer than some number of characters, no special characters).				
Expected	The registration should fail, and the user should be prompted to choose a stronger password meeting the minimum requirements.				
Date-Result	03/2024 – Pass Because Content Guard uses the OAuth password setting is not required and it has very sequre tokenazation, hence we would not directly interact with user's password				

Test ID	TC-45	**Category**	Stress	**Severity**	Major
Objective	To test failover and recovery mechanisms during system failures.				
Steps	1. Simulate system failures or crashes and observe failover procedures.				
Expected	Failover mechanisms should activate seamlessly, and the system should recover from failures with minimal disruption to service.				
Date-Result	03/2024 - Pass				

Test ID	TC-46	**Category**	Security	**Severity**	Critical
Objective	To test session management and protection against session hijacking.				
Steps	1. Attempt to hijack user sessions by stealing session tokens or cookies.				
Expected	Session tokens and cookies should be securely managed and protected against unauthorized access or theft.				
Date-Result	03/2024 - Pass (Twitter Authentication is safe.)				

Test ID	TC-47	**Category**	UI	**Severity**	Major
Objective	To verify website accessibility compliance.				

Steps	1. Test website accessibility using manual inspection.
Expected	The website should adhere to accessibility standards, ensuring usability for users with disabilities.
Date-Result	03/2024 - Pass

Test ID	TC-48	**Category**	Security	**Severity**	Major
Objective	To test system resilience against denial-of-service (DoS) attacks.				
Steps	1. Simulate various types of DoS attacks targeting the system's infrastructure or resources. This may include flooding the server with excessive requests, overwhelming network bandwidth, or exploiting vulnerabilities in the system. 2. Monitor system performance and response times during the simulated DoS attacks, observing any degradation in service quality or availability. 3. Analyze the effectiveness of the system's defenses against DoS attacks, including mechanisms for detecting and mitigating such threats. 4. Assess the system's ability to maintain service availability for legitimate users despite the ongoing DoS attacks.				

Expected	1. The system should detect and promptly respond to DoS attacks, implementing countermeasures to mitigate their impact and maintain service availability. 2. During DoS attacks, the system should dynamically adjust resource allocation and prioritize legitimate user requests to ensure continued access to essential functionalities, such as tweet extraction and categorization. 3. Mechanisms such as rate limiting, IP blocking, or traffic filtering should be employed to mitigate the effects of DoS attacks and prevent unauthorized access to the system. 4. The system should be resilient against various types of DoS attacks, demonstrating the ability to withstand sustained attacks without significant degradation in service quality or availability. 5. Continuous monitoring and proactive measures should be implemented to identify and address potential vulnerabilities that could be exploited in future DoS attacks, ensuring ongoing protection of the system's infrastructure and resources.
Date-Result	03/2024 - Fail

Test ID	TC-49	**Category**	Performance	**Severity**	Moderate
Objective	To test the extension's resource consumption during normal operation.				
Steps	1. Monitor CPU and memory usage while using the extension.				
Expected	The extension should consume reasonable system resources and not significantly impact system performance.				
Date-Result	03/2024 - Pass				

Test ID	TC-50	**Category**	Model	**Severity**	Major
Objective	To verify the ML model's scalability with increasing data volume.				
Steps	1. Increase the volume of training data used by the ML model.				
Expected	The ML model should maintain performance and accuracy even with large datasets.				
Date-Result	03/2024 - Pass				

Test ID	TC-51	**Category**	Security	**Severity**	Major
Objective	To test data validation and sanitization processes.				
Steps	1. Input malicious or malformed data into input fields.				
Expected	The system should sanitize inputs and reject potentially harmful data to prevent injection attacks or other security vulnerabilities.				
Date-Result	05/2024 - Pass				

Test ID	TC-52	**Category**	Functional	**Severity**	Minor
Objective	Verifying that the user can activate the extension by clicking the activate button				
Steps	1. Click on the activation button on the processing page.				
Expected	The extension should appear in the browser toolbar due to activation and the feed must be changed by extension.				
Date-Result	05/2024 - Pass				

Test ID	TC-53	**Category**	Functional	**Severity**	Major
Objective	Verifying that the user can deactivate the extension by removing it.				

Steps	1. Remove the extension from the browser’s extension page.
Expected	The extension should disappear in the browser toolbar due to activation, and the feed must return to the unchanged version. Extension’s features should not be visible.
Date-Result	05/2024 - Pass

Test ID	TC-54	**Category**	Functional	**Severity**	Major
Objective	To verify that the user can deactivate the extension by disabling it				
Steps	1. Open the browser. 2. Access the browser's extensions management page. 3. Locate the extension in the list of installed extensions. 4. Click on the option to disable the extension.				
Expected	1. The extension icon should be grayed out in the browser toolbar indicating deactivation. 2. The Twitter feed content should return to its original state without any manipulation. 3. The extension's features should no longer be accessible.				
Date-Result	05/2024 - Pass				

Test ID	TC-55	**Category**	Functional	**Severity**	Major
Objective	Verify that the user can deactivate the extension by signing out of the browser.				
Steps	1. Sign out from the account in the browser.				
Expected	The feed content should return to an unchanged state.				
Date-Result	05/2024 - Pass				

Test ID	TC-56	**Category**	Functional	**Severity**	Major
Objective	To test if the code segment to mutate data structures can be				

	injected
Steps	1. Open the extension window 2. Click to the button to see the filters 3. Click to the button to add a filter 4. Choose filter type (keyword/topic) 5. Enter the filter as “`var filters = [];`”
Expected	The filter should not introduce a bug by executing in the code.
Date-Result	03/2024 - Pass

6. Maintenance Plan and Details

Maintaining the ContentGuard system ensures its smooth operation, security, and adaptability to evolving user needs and technological advancements. The maintenance plan encompasses several key areas to address ongoing system upkeep, including software updates, security measures, user support, and potential enhancements. This section outlines the maintenance plan and details for ContentGuard:

6.1 Software Updates:

Regular software updates are essential to incorporate bug fixes, performance improvements, and feature enhancements. The development team will release periodic updates to the system's client and server layers. These updates will be distributed seamlessly to users, ensuring they can access the latest functionalities and optimizations. Additionally, updates will address compatibility issues with browser updates, ensuring continued functionality across various platforms.

6.2 Security Measures:

Ensuring the security of user data and system integrity is paramount. The maintenance plan includes regular security audits and vulnerability assessments to identify and mitigate potential threats. The server layer will be regularly patched to

address any known security vulnerabilities, while encryption protocols will be continuously evaluated and updated to maintain robust data protection measures. User authentication mechanisms will undergo periodic review to bolster security against unauthorized access attempts.

6.3 User Support:

Efficient user support is essential to address user inquiries, troubleshoot issues, and guide system usage. A dedicated support team will be available to assist users via email, chat, or a ticketing system. User feedback will be actively solicited and incorporated into future updates and enhancements to improve the overall user experience. Comprehensive documentation and tutorials will be provided to empower users to utilize the system's features and functionalities effectively.

6.4 System Monitoring and Performance Optimization:

System performance and usage metrics will be monitored to identify areas for optimization and performance tuning. This includes monitoring server resources, database performance, and network latency to ensure optimal system responsiveness and reliability. Performance optimizations will be implemented proactively to maintain high system availability and user satisfaction.

6.5 Disaster Recovery and Contingency Planning:

Robust disaster recovery and contingency plans will be established to mitigate the impact of potential system outages or data loss events. Regular backups of critical system data will be performed, with redundant storage solutions implemented to ensure data integrity and availability. Additionally, contingency protocols will be developed to enable rapid system restoration during unforeseen disruptions, minimizing downtime and ensuring business continuity.

The maintenance plan outlined above underscores our commitment to delivering a reliable, secure, and feature-rich user experience with ContentGuard. By prioritizing software updates, security measures, user support, potential

enhancements, system monitoring, and disaster recovery, we aim to ensure the long-term success and sustainability of the ContentGuard system.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

7.1.1 Constraints

7.1.1.1. Time Constraint

- Our project has a time constraint like all the projects. However, since we were doing a finishing project that needed to be finalized at the end of this term, we needed to plan all of our work to finish at the end of the semester, more specifically until our demo and CS Fair.
- To ensure that our time constraint was met, we made a Gantt Chart of our semester and when to start tasks to ensure that they would be finalized by our deadline. In this Gantt Chart, we assigned the tasks, the time that we needed to start them, their importance to our project, how they affect the rest of the task, and how early they must be started.
- According to this plan, we allocated our tasks in the order that was decided (with the flexibility to change them if necessary) to the teammates that have the most proclivity and experience on the subject, and if nobody has these, to one of the teammates that can create time for the task.
- We checked where we are on our project in weekly meetings and updates on WhatsApp to see where we are and what steps to take to make the most of our project in time.
- In this, we needed to be flexible because writing the code, training the model, etc., is a time-consuming task that must be done early on [3].

7.1.1.2. Resource Constraint

- We needed resources to make our project, and some of these are servers and APIs, and some of these are budget since the technologies we use might need one[3].

- To make our project work, we needed to use many technological tools. We used OpenAI's GPT-3.5 Turbo to train the ML model, SQLite for database management, Firebase for authentication, React.js for UI development and many libraries. We tried to choose the most available resources and easily acquired them.
- Also, because of the budget resource constraint, we are using GPT from OpenAI GPT-3.5 Turbo to train the ML model. Since it is free, it can cause a lag in the system as it might work slowly.
- Because Twitter API is used when Firebase connects with it for authentication but was not a resource the we acquired.

7.1.1.3. Compatibility Constraint

- Since we are creating an extension for Twitter, it needs to work with Twitter. Especially since we provide the sign-in with the authentication of Twitter. This makes it necessary for our extension to be compatible with Twitter. We have achieved this.
- To provide this, we are using the systems that are already compatible with Twitter while creating our extension like Firebase. Since there are many other Twitter extensions, we can also learn from them what works and what doesn’t.
- We also try all the codes in the already-built extension to make sure it will be compatible with Twitter.
- Software compatibility between components of the server layer, such as JavaScript and SQLite systems, must be compatible with the hardware used. To ensure this, we aim to choose them to be compatible from the start by researching and we have achieved this[3].

7.1.1.4. Technology Constraint

- For our project, we need technologies for ML training, authentication, etc. as mentioned in 7.1.1.2 and the technologies mentioned here were used.
- For the software part of the server, JavaScript is used for the most part. Databases are planned as an SQL-based system and SQLite was chosen. As the NLP model, GPT 3.5 Turbo from OpenAI was chosen. This decision was made because of the speed, the computational power of the model, and it's financially more sensible . This technology was chosen to be compatible and most useful for our purposes.

7.1.1.5. Security Constraint

- Since we are using user's data and making them sign in and register with Twitter, we must protect their data with encryption or authentication and check what is available. We used Firebase authentication for this.

7.1.2 Standards

7.1.2.1. Data Security Standards

To ensure robust data security standards for user data stored in the database, our project will implement OAuth authentication through Twitter for secure user login. For authentication we used Firebase. We used cookies to transfer sensitive user data from Twitter and our extension. We established access controls and authorization mechanisms specifically for tasks using these systems. Furthermore, regular data backups will be conducted to prevent loss or corruption, and comprehensive audit trails and logging will monitor database activity, ensuring accountability and traceability [3].

7.1.2.2. User Privacy Standards

For user privacy standards, our project adopts a privacy-centric approach by storing tweet content consumed by users locally in Chrome storage, ensuring sensitive interactions remain under the user's control. Following categorization, filtering, and blocking operations, related outputs such as category results, filtering options, and blocking options will be stored in the database, but with anonymization and encryption so that data should not be linkable to the user. This strategy will safeguard user privacy, preventing unauthorized access to sensitive information even in the event of a security breach [3].

7.1.2.3. HTML Scraping Standards

Our project was designed to adhere to ethical and legal HTML scraping standards by respecting Twitter's terms of service and adhering to its robots.txt directives. We prioritize data accuracy and integrity, validate scraped data sources, and implement error-handling mechanisms to conduct web scraping responsibly and foster trust with content providers and users [3].

7.1.2.4 Regulatory Compliance

Our project is designed and aimed to ensure adherence to relevant laws and regulations governing data privacy, security, and online interactions. We will prioritize compliance with KVKK and other regional data protection laws to safeguard user privacy and rights. Furthermore, we will ensure compliance with platform-specific regulations of social media platforms, guaranteeing that our application aligns with the standards established by Twitter and other platforms. This commitment to regulatory compliance will not only help mitigate legal risks but also underscore our dedication to operating ethically and lawfully [3].

7.1.2.5 Ethical Standards

Our project is one that stands by values of integrity, transparency, and safeguarding user rights. We place a high priority on protecting user privacy and data, and handling information with cautious attention and solely for its designated purposes. Moreover, we will uphold transparency in our operations, offering users clear insights into data collection and its utilization. By tightly adhering to these ethical standards, our goal is to cultivate trust among users and foster a positive impact within the digital community [3].

7.2. Ethics and Professional Responsibilities

- While processing the feed of the user, we will process the user's original paragraphs, sentences, and words, which are the product of their creative process. Because this data is sensitive and it is the user's right to share it, we will protect them and not share the data with any other third-party company, person, etc.
- We will store the data in a secure way to be professional in our act of protecting user’s rights.
- When storing the login data for the user, we will store them securely and use hashes to store the passwords to increase privacy and security.
- After a user logs in to Twitter, we are considering giving the user more control by asking them if they want to launch the application or other measures of control like stopping the application for once choice and stopping for always choice might be given to the user.

- Also, after launching the application to be used by people on the internet, we will make documentation that shows the user to informs the user about these ethical constraints so that they can choose freely to download or not.
- The data will not be stored so that its source can be linked.
- The data will be analyzed and grouped to minimize biases in the subject, and experts may be consulted for determination.
- Our team will follow the Ethics of the National Society of Professional Engineers in this project [8].
- If some features are decided to be paid features or the subscription is added to the Twitter Content Tracker in the future by the development team, it is understood by us that the payments system should be secure, and the team will make their efforts to make it as such [9].

7.3. Teamwork Details

7.3.1. Contributing and functioning effectively on the team

We, as a team, see a considerable part of our responsibility in carrying the Content-Guard project to success lies in functioning effectively in the team. Thus, we have decided to track our progress weekly (sometimes two times a week) with discussions and work allocation updates. We made an initial work allocation with our team members before the beginning of the semester and made a Gantt chart (which we update in our weekly discussions if necessary) to allocate the work most effectively. Deciding the best part for a teammate by making an assessment, according to the part's urgency, our teammate's experience, and their proclivity to the part we made our current work allocation in a way we believe each teammate can contribute and function effectively the most as an individual and as a team. Following is a brief explanation of our current work allocation [3]:

İlayda Zehra Yılmaz: İlayda is dealing with the front end of the project and finding suitable datasets for the training of the training model. She was responsible for the Content Guard Home page's front-end code and she helped in reports as well as initial research of the project. She has worked as a front-end developer in internships and projects and is training herself in the datasets.

Gülin Çetinus: Gülin is dealing with the testing and the back end of the project, mainly on the extension's development. She has worked on the back end of the projects for both of her internships. She also worked on this project's extension part she established keyword blocker core functionalities and helped with authentication. Also helped with timer implementation.

Zeynep Derin Dedeler: Zeynep is dealing with HTML scraping for the NPL model and generally focuses on the backend, background script, and content script development. She worked with data analysis during her second internship. She helped with designing extension structure and helped establising keyword blocker core functionalities. Implemented timer functionalities and related UI

Bengisu Buket Karadoğan: Bengisu is dealing with the NLP model for tweet classification purposes and communication of the model with extension. She is interested in the fields of machine learning and natural language processing. She was responsible for the Backend, implemented most of it, and created communication between the extension and the home page. Also helped with authentication, report functionalities and timestamping them.

Burak Öztürk: Burak is dealing with common industry technologies and practices research, overall project structure planning/designing, and extension development. He is interested in customer software development and game design and completed an internship on each of those topics. He Improved the total performance and worked on the extension part. He also created the structure of the extension part and did the authentication, helped with timer implementation and related UIs.

7.3.2. Helping creating a collaborative and inclusive environment

As mentioned above in 7.3.1, we tried to create a collaborative environment through weekly discussions and work allocations. To explain these discussions more, we were all included to explain or ask questions and all of the members could bring something to the table because we would lead the conversations in such a way that everybody would be able to talk. These discussions and work allocations that were performed by asking everybody's opinion and expertise helped everybody feel

included and created a collaborative environment where we would help each other with our tasks and questions. To increase the inclusion and collaboration in our team, we frequently use WhatsApp and Discord for our questions. We upload our code to GitHub and use Asana to allocate our work. Being in contact always helps us collaborate faster if a critical problem occurs as well. We also share our work according to the task's greatness and sometimes work together on the same task. We also use face to face discussions and studies. We also give the reins on a subject to a team member that is knowledgeable, and we also take turns in leading team meetings so that all the members feel the responsibility and we can move forward effectively with the best ideas. These are all some of the ways that we use to increase collaboration and inclusiveness as everybody being heard and being helpful to each other is a very important part of our team structure [3].

7.3.3. Taking lead role and sharing leadership on the team

As a team, we believe that strong and responsible leadership is the wind that gives the ship that is our project its speed. It enables our collaborative and inclusive team to be more effective. Therefore we allocated a leadership system in which every team member leads a specific part of the project, and all of us are responsible for giving the heads up to the full participation tasks like reports. Because we all utilize our leadership skills in different tasks, we all become much more experienced when one of us has to lead others on some new tasks and we can decide who is fit best to lead that task. We can all trust that leaders are capable of guiding us and giving us speed in their subjects. For the weekly meetings generally, one of us starts asking everybody about their work, and that meeting is led by that person largely, although there is not a rule for this. We, in fact, still keep being the leader in our parts if we have an idea. The tools used in 7.3.1 and 7.3.2 help us run our tasks smoothly and efficiently, with all of us collaboratively leading our respective parts. Since we can all take the reins and lead the next step of our project management process, we all feel responsible for our success and more confidently take the lead role in the subject we feel confident in. This ensures our team members are always engaged since we can have an active leading position on a subject or collaborate with the current leader on the subject. This ensures effectiveness since we stay active. Of course, to ensure efficiency when there is an emptiness in leading all the tasks, one of us goes forward and takes the lead role to start a discussion, propose a meeting,

or motivate others. We take this seriously, so we accomplish no empty lead roles left to protect our project from ever going astray. The constant feedback of the leaders and our members who take all the parts of the project seriously is a huge blessing to our team and leads us to success in sharing the leadership on the team [3].

7.3.4. Meeting objectives

To meet our objectives we have designed a Gantt chart as mentioned above and we have created deadlines for our tasks in Asana. We have checked that if our objectives would be met in the time we had for our project and decided the importance and the priority of each task. Thanks to this approach we have known the stakes of the tasks at hand and how they affected the chain of tasks that came in the future. So even in the busy times of the semester, we did lay the groundwork for the next objective so that we would be capable of implementing most of the initial and vital features of our project. We have kept in touch regularly using Whatsapp texting and Zoom video calls to plan our meetings, check our progress, and allocate our work. And we gave all of the teammates the power to call the team into action when the team is losing its productivity in reaching the objectives. We realized we needed to put this project into higher priority than most of our lesson's work at certain times to move forward. All of the team members being collaborative and helpful as well as responsible enough to put work on this project helped us meet our objectives.

7.4 New Knowledge Acquired and Applied

We have learned team work and how to function as a team as well as reaching our goals thanks to this project. We have learned how to effectively use collaboration tools like Asana as well as effectively doing work allocations to each member. We have learned to how to communicate in a group environment effectively. Since our project uses many different technologies we learned about the ones we didn't have prior knowledge and we have learned more about the ones we had knowledge about. We have learned Server-Client architecture and we used this technology for the server managing authentication, databases, and NLP tasks, while the client handles UI and local storage. We have learned about extensions specifically Chrome extensions. We used these to apply Chrome API to our browser extension functionality and DOM manipulation to interact with web page content. Wr have used how to use a template and used Vision UI template for our home page. Wh have learned how to make Firebase authentication and wrote our authentication using that so we could authenticate with user's Twitter account using the Twitter API authentication. Some of us learned how to use React as well as some of us needed to learn more about it as using it on an extension with the complex tasks we had was a whole new skill. We have learned and

used Django for backend development as well as SQLite for database management. We have learned and integrated ChatGPT 3.5 Turbo for our NPL tasks.

8. Conclusion and Future Work

In conclusion, ContentGuard emerges as a multifaceted solution meticulously crafted to address the complex challenges of information overload and time management prevalent in the dynamic landscape of Twitter. With its array of features and functionalities, ContentGuard aims to empower users with unprecedented control over their social media experience. As we look forward to the future of this project, several key areas for further development and enhancement emerge:

Future Work:

1. Integration of Historical Data and Reports:

- Incorporate older reports into the system to enable users to track their engagement trends over time.
- Provide insights into long-term patterns and behaviors to facilitate informed decision-making.

2. Behavioral Analysis and Personalization:

- Implement advanced algorithms to analyze user behavior and preferences.
- Offer personalized recommendations for content filtering and time management based on individual usage patterns.

3. Continuous Performance Optimization:

- Focus on optimizing content categorization algorithms to improve processing speed and accuracy.
- Enhance overall system efficiency to ensure a seamless user experience even during peak usage periods.

4. In-House Model Development:

- Explore the possibility of developing a proprietary machine learning model for tweet categorization.
- Reduce reliance on external services like ChatGPT 3.5 Turbo, providing greater flexibility and control over the classification process.

5. Expansion to Other Social Media Platforms:

- Extend ContentGuard's functionality to support additional platforms such as Threads, Reddit, and YouTube.
- Adapt the system to cater to the unique content consumption patterns and challenges of each platform.

Model Database:

In preparation for potential in-house model development, the establishment of a dedicated model database is paramount. This database would serve as a central repository for:

- Storing essential data required for training and testing the machine learning model.
- Facilitating ongoing maintenance and optimization of the model through access to historical training data and performance metrics.
- Supporting seamless integration with the ContentGuard system, ensuring efficient communication and data exchange between the model and other system components.

By pursuing these avenues for future work and investing in the development of advanced functionalities and infrastructure, ContentGuard aims to evolve into a versatile and indispensable tool for effective content management and user control across diverse social media platforms.

9. Glossary

Information Overload: A state in which an individual is overwhelmed by the amount of information available, making it difficult to process effectively.

Artificial Intelligence (AI): The simulation of human intelligence processes by computer systems.

Machine Learning (ML): A field of AI that focuses on making computers learn and improve from experience without being directly programmed to do so.

Natural Language Processing (NLP): A field of AI focused on training ML language models to understand, interpret, and generate human language.

Server-Client Architecture: A network architecture where tasks or processes are divided between servers (providing role) and clients (requesting role).

10. References

[1] M. Gomez-Rodriguez, K. P. Gummadi, and B. Schoelkopf, “Quantifying information overload in social media and its impact on social contagions,” Proceedings of the International AAAI Conference on Web and Social Media, vol. 8, no. 1, pp. 170–179, May 2014, doi: 10.1609/icwsm.v8i1.14549.

[2] K. Bontcheva, G. Gorrell, and B. Wessels, “Social media and information overload: survey results,” *arXiv.org*, Jun. 04, 2013. https://arxiv.org/abs/1306.0813

[3] G. Çetinus, Z. D. Dedeler, B. B. Karadoğan, B. Öztürk, and İ. Z. Yılmaz, “T2321 Content Guard Detailed Design Report,” Mar. 15, 2024

[4] “What is machine learning (ML)?,” IBM, https://www.ibm.com/topics/machine-learning#:~:text=Resources-,Take%20the%20next%20step,learn%2C%20gradually%20improving%20its%20accuracy. (accessed Mar. 15, 2024).

[5] “What is natural language processing?,” IBM, https://www.ibm.com/topics/natural-language-processing (accessed Mar. 15, 2024).

[6] F. Churchville, “What is User Interface (UI)? definition from searchapparchitecture,” App Architecture, https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI (accessed Mar. 15, 2024).

[7] “Kişisel Verileri Koruma Kurumu: KVKK: Personal Data Protection Authority,” KVKK, https://kvkk.gov.tr/Icerik/6586/Personal-Data-Protection-Authority (accessed Mar. 15, 2024).

[8] NSPE, “NSPE Code of Ethics for Engineers,” National Society of Professional Engineers, Jul. 2019. https://www.nspe.org/resources/ethics/code-ethics (accessed Nov. 16, 2023).

[9] G. Çetinus, Z. D. Dedeler, B. B. Karadoğan, B. Öztürk, and İ. Z. Yılmaz, “T2321 ContentGurad Analysis and Requirement Report,” Dec. 8, 2023

[10] “Favorited_tweets - dataset by A2liz,” data.world, https://data.world/a2liz/favorited-tweets/workspace/file?filename=favorite-tweets.jsonl (accessed May 13, 2024).

[11] “Double-precision floating-point format,” Wikipedia, https://en.wikipedia.org/wiki/Double-precision_floating-point_format#Precision_limitations_on_integer_values (accessed May 13, 2024).