

*Comment Anywhere*

User Manual

Karl Miller

Luke Bates

Frank Bedekovich

**Instructor Comments/Evaluation**

# Table of Contents

[1. Table of Contents 3](#_Toc133186058)

[2. Project Overview 5](#_Toc133186059)

[2.1. Motivation 6](#_Toc133186060)

[2.2. Other Differences from Existing Comment Providers 7](#_Toc133186061)

[2.3. Social Implications 7](#_Toc133186062)

[3. System Block Diagram 9](#_Toc133186063)

[3.1. System Block Diagram Explanation 9](#_Toc133186064)

[4. Implementation Details 11](#_Toc133186065)

[4.1. Differences from the Design Document 11](#_Toc133186066)

[4.1.1. Token Transmission 11](#_Toc133186067)

[4.1.2. Schema Changes 11](#_Toc133186068)

[4.1.3. Communication Entity Changes 12](#_Toc133186069)

[4.1.4. Module Database Utilization 13](#_Toc133186070)

[4.2. Challenges During Implementation 14](#_Toc133186071)

[4.2.1. Docker-izing the HTTP Server 14](#_Toc133186072)

[4.2.2. Getting the URL; browser.runtime.Port 14](#_Toc133186073)

[4.2.3. Debugging the Extension 15](#_Toc133186074)

[4.2.4. Testing 15](#_Toc133186075)

[5. Software Engineering Principles 16](#_Toc133186076)

[5.1. Waterfall Methodology 16](#_Toc133186077)

[5.2. Modularity, Encapsulation 17](#_Toc133186078)

[5.3. Abstraction, Don’t-Repeat-Yourself 17](#_Toc133186079)

[5.4. Separation of Concerns 18](#_Toc133186080)

[6. User’s Manual 19](#_Toc133186081)

[6.1. Installing 19](#_Toc133186082)

[6.2. Opening 20](#_Toc133186083)

[6.3. Navigating 20](#_Toc133186084)

[6.4. Registering an Account 21](#_Toc133186085)

[6.5. Logging In 22](#_Toc133186086)

[6.6. Logging Out 22](#_Toc133186087)

[6.7. Viewing Comments 22](#_Toc133186088)

[6.8. Viewing Comments for a Pseudo-URL 23](#_Toc133186089)

[6.9. Replying to Comments 24](#_Toc133186090)

[6.10. Reporting Comments 24](#_Toc133186091)

[6.11. Validating an Email 24](#_Toc133186092)

[6.12. Resetting a Password 25](#_Toc133186093)

[6.13. Changing an Email 25](#_Toc133186094)

[6.14. Changing a Profile Blurb 26](#_Toc133186095)

[6.15. Submitting Feedback 26](#_Toc133186096)

[6.16. Moderator: Viewing Reported Comments 26](#_Toc133186097)

[6.17. Moderator: Moderating Comments 27](#_Toc133186098)

[6.18. Moderator: Banning Users 27](#_Toc133186099)

[6.19. Moderator: Viewing Moderator Actions 28](#_Toc133186100)

[6.20. Admin: Viewing the Users Report 29](#_Toc133186101)

[6.21. Admin: Viewing the Feedback Report 29](#_Toc133186102)

[6.22. Admin: Assigning and Removing a Global Moderator 30](#_Toc133186103)

[6.23. Admin: Assigning Admin Privileges 31](#_Toc133186104)

[6.24. Admin: Viewing Access Logs 31](#_Toc133186105)

[7. Glossary 32](#_Toc133186106)

[8. Bibliography 41](#_Toc133186107)

[9. Appendix 46](#_Toc133186108)

[9.1. Team Details, Individual Contributions 46](#_Toc133186109)

[9.2. Code Listing 47](#_Toc133186110)

[9.3. Workflow Authentication 47](#_Toc133186111)

# Project Overview

Comment Anywhere is a web service that allows users to submit to and view a unique comment section for on any webpage on the Internet. Comments are viewed, posted, and interacted with via the Front End, a downloadable browser extension for Mozilla Firefox. The Back End stores comment data and other relevant data, provides that data to users, and validates user permissions. Comment Anywhere also provides a set of moderation tools for removing objectionable content.

## Motivation

Internet denizens have long found ways to have vibrant communications about a wide variety of content. In the past, more websites supported these conversations through comment sections, but many have shuttered their comment sections in recent years. (1) (1) Instead, the avenues of discourse have become social media sites such as Twitter, Facebook, Reddit, and bulletin board style forums, decoupling the conversation from the content itself.

While these sites may have a higher quantity of overall content than a given news site, they lend themselves to agendas and one-size-fits-all moderation policies while, at the same time, fragmenting the conversation and diluting information available to viewers of the core content. The conversation about a given point of interest on the internet becomes dispersed amongst any number of social media posts, thereby burying much valuable information and well formulated alternate views. While it is usually trivial to move from a conversation on social media to the content discussed, it is much more difficult to move from the content to the decoupled conversation.

We believe there is a usefulness and market desire to see comment sections closely coupled with the content, that will follow the content wherever it goes. Comment Anywhere achieves this by permanently tying a comment section with a given URL and making those comments available immediately to users as they browse the web through a browser extension. We provide a unique comment section for every unique domain and path combination on the internet, allowing users to offer feedback and insights on under-commented areas of the internet, especially news sites, but also government and business websites.

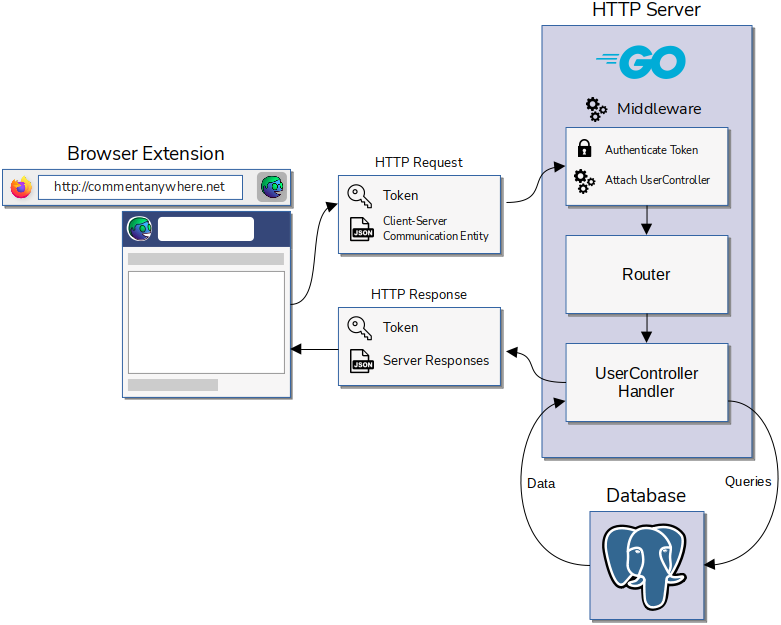
## Other Differences from Existing Comment Providers

There are a few other features, besides permanent, content-tied comment sections that make Comment Anywhere unique. Posts have three dimensions of ratings; “funny”, “factual”, and “agree”, allowing users to sort on several dimensions that they are interested in. Other comment sections have only one dimension of rating (e.g. Reddit, Twitter) or limit sorting options (e.g. Facebook). Comment Anywhere also has the notion of Domain Moderators, users with privileges to moderate comments, but only for a specific domain. This allows us to create and enforce tailored moderation policies with respect to comments posted on a particular domain, when necessary, without having to apply those moderation policies universally across the website.

## Social Implications

At their best, comments are a forum for lively debate of the issues reported on or expressed in a piece, as well as a source for personal experiences and further information related to the content. The idea that information should be freely accessible runs deep within the ideology of the internet, and the notion that speech should be free is central to the values of this nation. Comment Anywhere provides voice to alternate and silenced opinions that challenge entrenched agendas across the world, facilitating the speaking of truth to power and enabling societal change. While we must use moderation tools to limit libel and harmful comments about private citizens, varied opinions about public figures and institutions should be accessible to all. Discourse is critical to the democratic process.

# System Block Diagram



*Figure 1.1: System Block Diagram*

## System Block Diagram Explanation

Figure 1.1 outlines the communicational process between the client-side browser extension and server-side HTTP server. When the user interacts with the browser extension by, for example, requesting to login, changing a setting, or posting a comment, an HTTP request is sent to the server's static IP address, containing the user's token and a JSON object containing data relevant to particular request.

At the server, the HTTP request first passes through middleware which authenticates the token sent with the request. If the token verifies a user's identity, that user's UserController (e.g. MemberController, AdminController, etc.) is attached to the request context Otherwise, a new GuestController is created for the user and attached to the request context.

Next, the request reaches the server router which will select a server API endpoint handler method based on the path of the api endpoint. For example, "commentanywhere.net/login" will select *Server.postLogin*.

The endpoint handler will extract the expected communication entity from the request. For example, "login" expects a type of *communication.Login* containing a username and password.

The endpoint handler next extracts the controller that was previously attached to the request context by the middleware. It calls the associated handler method on that controller. For example, *Server.PostLogin* will call *Controller.HandleCommandLogin,* passing the extracted communication entity. All controllers share an interface, and the method called is polymorphically selected.

Next, the selected UserController handler queries the database to retrieve, update, or add relevant data, depending on the type of request. In the process, it will typically add Server-Client communication entities to the Controller's *nextResponse* array, such as *communication.Messages* and *communication.LoginResponses*.

When the handler is finished, a final middleware once again extracts the controller from the request context and populates the HTTP Response with the data within the controller's *nextResponse* array. The request finishes and the HTTP Response is dispatched back to the client.

When the client receives this HTTP response, it invokes functions to realize the necessary changes to the user interface based on the array of communication entities it received from the server.

# Implementation Details

## Differences from the Design Document

In the 165-page design document, we tried to anticipate every module, class, method, and message type we would need for Comment Anywhere, but we still had to make several changes to achieve the functionality we desired.

### Token Transmission

Users prove their identity using encrypted JSON Web Tokens (JWT). We originally planned to transmit these tokens as cookies, which is a common practice. While implementing the register, login, and logout system, we found that we could not use cookies as we had planned. This was due to Cross Origin Resource Sharing (CORS) security policies within the browser. Our browser extension front-end runs locally in a user’s browser, and its *fetch* requests to the server are fundamentally cross-origin. Mozilla and other browsers prohibit CORS requests from accessing cookies. Instead, we authorized a custom HTTP Header in our preflight request, and had the Front End write its token there.

### Schema Changes

Our schema changes were minor, such as adding “NOT NULL” and “DEFAULT” to various columns, such as timestamps indicating a row’s creation. The primary key of the *VoteRecords* table had to be altered from being a composite primary key consisting of the comment ID and category to also including the user ID of the voter. In the *PasswordResetCodes* table, the code itself was made the primary key rather than a separate column, for simplicity. Some columns were renamed for consistency. For example, *DomainBans.user* was renamed to *DomainBans.user\_id* and *Comments.pathid* was renamed to *Comments.path\_id*.

### Communication Entity Changes

Many of our changes related to communication entities, which are shared types between the front and back end that describe the structure of the JSON strings that the front end and back end pass to each other.

The *Ban* structure sent to the server indicating a user to ban was changed from requiring a user ID to a username, to simplify the transformation of the front end form to the communication entity and to reduce database queries. It was also given a Boolean *Ban.ban* field, which can be false if the action is an unban.

The fields *Communication.Moderate* and *Communication.ReportID*, both of which reference an associated comment report by its 64-bit id, were changed from type int64 to a pointer to an int64. This allowed the use of “nil” values, in case the action was taken without an associated report.

We got rid of the *PasswordResetCode* client-server communication entity and combined its field into the *SetNewPass* entity, so that we wouldn’t need to maintain a state indicating user’s progression through the set new password process. Instead, they submit a new password at the same time they submit their code. This was done to simplify that pipeline.

We added a *ProfileUpdateResponse* entity that is dispatched to a user whenever their profile is changed. We were able to use this one entity whenever they change their own profile, such as by changing their profile blurb, but also dispatch it to them when their information changes in some other way, such as if they have been granted moderator permissions.

We added a *FullPage* entity for responding to a user’s first request for comments for a page. It transmits all comments for that page as an array, as well as the Domain and Path of the page. This was always needed and omitting it from the design document was an oversight.

We renamed *ViewModRecords* to *ViewModActions* and added several fields to *ViewModActions* to allow filtering on several dimensions. *ForDomain* filters for actions related to a particular domain, *ByUser* filters for actions performed by a user, and *From* and *To* allow the passing of a date range. These filters are only applied if they are not empty, so some or none of them can be applied. This was to make the viewing of mod records more useful, not just providing the moderator with every action ever taken every time.

### Module Database Utilization

We added many methods to the object *Store* within module *Database* on the back end, starting about halfway through back-end implementation. We had noticed that we were often calling a series of database queries in our routes, and some of these patterns were being repeated. *Store* was a natural choice to abstract repetitive database queries, since the object *Queries*, which directly accessed the database, was a member of *Store*. In the design document, we had only provided *Store* with *New*, *Connect*, and *Disconnect*, but it received huge additions, gaining approximately 40 methods. For example, *Store.GlobalUnban(userId, bannedBy, reason)* could verify that a user was currently banned, unban the user globally, and insert the ban record associated with the unbanning. We took care that *Store* never depended on *Server*, and only returned and took as parameters primitives and communication entities.

## Challenges During Implementation

### Docker-izing the HTTP Server

When we first deployed to the cloud, we attempted to install Go on our Virtual Private Server and run the HTTP Server as a regular system process. This worked to an extent, but we soon realized that unless we ran it headlessly, the process would terminate as soon as the SSH terminal attached to it was closed. Running headlessly was not entirely straightforward, so we decided to implement the HTTP Server as a docker container on our cloud server, since we were already using docker to run our database. This required creating a Dockerfile to build an image for our server and using docker network to connect our server to our database. We also had to implement some flags that the server binary could parse when it was run that would indicate whether it was running in docker or not, causing it to use the environment variables differently. Those flags allowed us to continue to test the server outside of docker outside of a container as we iterated.

### Getting the URL; browser.runtime.Port

The browser extension popup does not directly have access to the URL the user is currently viewing. To access the URL of the user’s current page, we needed to create an additional script called a background script, which is a type of script that only an extension can run. The background script has access to the *tabs* browser API, and permits access to the URL of the active tab. To communicate this information to the popup portion of the extension, we had to create a type of event listener called a browser.runtime.Port for both the popup script and the background script. Using that listener, the popup is able to signal to the background script when it needs the current URL and the background script is able to send that information to the popup. The popup script receives that URL and executes the fetch request to get the comment data from the server for the current page and display it. We also had to configure Vite, our bundling tool, to produce these two scripts instead of one bundle and we had to alter our manifest.json to tell Mozilla how to use them. We also had to implement checks for the front end to see if it is in fact running in a browser, so that we wouldn’t disable running the extension as a webpage, which is a faster iteration when testing some kinds of features.

### Debugging the Extension

Extensions do not log messages to the browser console like regular webpages do. That renders all print statements (*console.log*) effectively useless and makes debugging more difficult. To get around this, we created a debug class that places text directly on the DOM as a fixed-position overlay. We set an environment variable that indicates whether the extension is in debug mode or not. If it is, the global *console.log* is rerouted to our *debug.log*, and all printed messages show on a clearable transparent overlay atop the extension popup content. This was particularly important when implementing the background scripts and validating serialization and deserialization from the browser storage API.

### Testing

We wanted to thoroughly unit test every component, but with a huge list of features, a tight deadline, and a small team, we were not able to write the number of unit tests and benchmarks we had hoped for. Building tests is involved for both servers and for webpages.

Thoroughly testing servers require simulating HTTP Requests and HTTP Responses. While Go does provide a testing framework that allows the use of faked response writers, there are two primary issues that made it too involved to implement prior to the alpha release. First, the effect of a request is dependent on a user’s access level, such as whether they are logged in, so we would have had to log in a user prior to most tests and save the token. Secondly, most of the effects of a request result in changes to the database, so we would have had to implement the rebuilding of a test database that mirrored our regular database. While we did implement the creation of a test database, we did not have the time to write unit tests for each of the API endpoints. We were able to write tests for other components of the server, such as the *util.ExtractPathParts* function that uses a Regular Expression to extract the domain and path from an URL.

On the Front End, tests require simulating a DOM environment. While this is possible through Vitest, which integrates nicely with our bundler Vite, it was ultimately a new and involved paradigm. With only two coders and many features to push, we were not able to make the time investments to learn Vitest paradigms.

# Software Engineering Principles

## Waterfall Methodology

We followed the Waterfall software development method as called for by the curriculum. During our first semester, we produced a Proposal, Requirements, Analysis, and Design document. We strove to embrace the Waterfall paradigm and were thorough in our documentation. During that first semester, we also produced two prototypes, one for the back and the front end, to foresee challenges and anticipate our design needs. By doing so, we were able to produce a 168-page Design document describing almost every class and method we needed, which we followed closely throughout implementation. In particular, we defined a standardized means of communicating between the front-end and the back-end, and we were able schedule implementation based around these communication pipelines.

## Modularity, Encapsulation

We broke our systems into smaller, more manageable components. The front end was divided into classes, most of which inherited from *UIComponent*, *CafeSection*, or *CafeWindow*. We kept UI classes dedicated to a single representation, while Sections and Windows were effectively simple containers. Indeed, the base *UIComponent* class has a generic type parameter, as its purpose would always be to represent some server-client communication entity received by the front end or some Client-Server communication entity to be transmitted to the back end. For example, *UIComponent<Server.Comment>* was extended to create the *CafeComment* class. On the back end, we used structs (Go does not have classes) to separate out *UserControllers*, *Server*, *Store*, *UserManager*, *Page*, and *PageManager*, as well as a submodule for querying the database, for reading the .env variable, for general utility, and for maintaining communication entity types.

## Abstraction, Don’t-Repeat-Yourself

We found several excellent abstractions throughout the course of implementation that quickened our progress. On the back end, we made heavy use of the *Store* struct, which was responsible for querying the database through its member *Queries*, which was an object procedurally generated from SQL queries using a library called sqlc. We added methods to *Store* for performing several sequential queries, often returning a communication entity. In this way, we were able to keep the *Server* and *Controller* components responsible for routing requests while delegating the database work to *Store*. These methods saw a lot of re-use. For example, *Store.GetCommUser*, which generates a *communication.UserProfile* type after making a few queries is called in at least seven places.

On the front-end, we did a lot with abstracting Document Object Model (DOM) functions. Our first core abstraction was the *Dom* namespace, which provided functions such as *div, button*, *el* and so forth, accepting optional element types, text content, and CSS class names as parameters. We later added a second abstractor, called *CafeDom*, which called *Dom*, but supplied some predefined CSS classes that we were reusing a lot. For example, *CafeDom.formSubmitButtonSmall* returns a button with the CSS class “form-submit-button-small”. Each of these abstractions only saved us perhaps 3-4 lines of code per call but, considering we created hundreds of HTML Elements, it added up.

## Separation of Concerns

We had a challenge early in design when deciding how to separate our concerns on the front end during the processing of user interactions. When a user clicked a button on a deeply nested element, there might have to be a request dispatched to the server. We didn’t want every nested button making requests itself and abstracted all server communication to a class called *Fetcher*. However, we needed to figure out how to access *Fetcher* from a deeply nested object. At first, we thought we might pass down a reference to the top level *Cafe* to every object, so they could all access everything, but this meant tightly coupling the entire code base and providing the opportunity to introduce very confusing logic by having deeply nested UI endpoint classes calling top level methods.

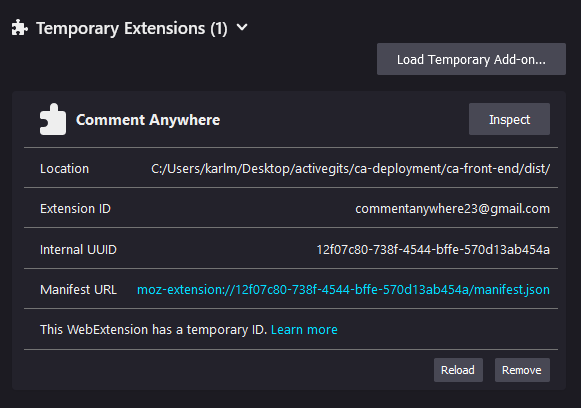
We solved this by structuring the entire front-end around custom events. Every HTML element has access to a reference to the global *document* object, which can register and signal event listeners. While typically used for events such as “onload”, *document* can also register listeners for custom events. We decided to create types for our own custom events, whose names would always be names of client-server communication entities and whose data would always be the content of a client-server communication entity. For example, when a submit button is clicked on the Login form, an object of type *Client.Login* is created and populated with the form data. A custom event named “Login” is dispatched to the *document* object, with that *Client.Login* data as the payload. We delegated the listening of events to our top level *Cafe* object, which would simply pass the data along to *Fetcher*.

On the back end, one significant way we achieved separation of concerns is by running our database as a completely different process that is not at all dependent on the functionality of the HTTP Server. This is a common practice in back-end implementations, but it is nevertheless a valuable one. As a result of having our database run in a separate container unconcerned with the HTTP Server, we can update and re-run our server at will without losing any data.

# User’s Manual

## Installing

At this time, Comment Anywhere is only available in alpha release, pending approval from Mozilla to be added to the extension store. It can be downloaded from <http://commentanywhere.net>. After downloading, extract the .zip file on your computer. In Mozilla Firefox, type the following in the URL bar: “about:debugging”. A special developer pane in Mozilla will open. Click on the button on the left labeled “This Firefox”. Click “Load Temporary Addon”. Navigate to the folder previously extracted and double click the file named “manifest.json”. Comment Anywhere will be added to the browser.

  
*Figure 6.1: Loading an Extension*

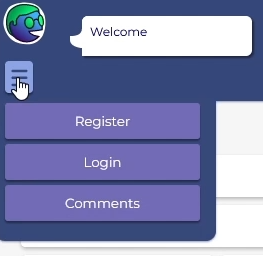
## Opening

After installing, Comment Anywhere guy will appear as a button to the right of the URL bar in Mozilla Firefox. Clicking him will pop-down Comment Anywhere.



*Figure 6.2. The Comment Anywhere Guy icon.*

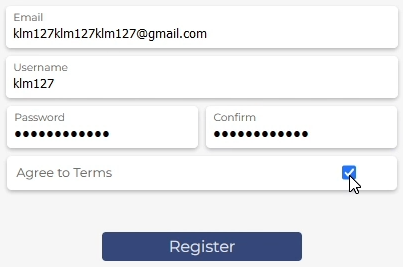
## Navigating



*Figure 6.3. The Navigation Panel*

Click on the hamburger button to access the navigation pane. The user may select the desired page to traverse to.

## Registering an Account



*Figure 6.4. Registering an Account*

Navigate to the Registration page using the navigation pane. You must provide an email, username, password, and an agreement to our terms to create a Comment Anywhere account. These details must hold against various requirements. Your email must have the correct format and be unique. Your username must only contain alphanumeric characters and be unique. Your password must be at least 8 characters long.

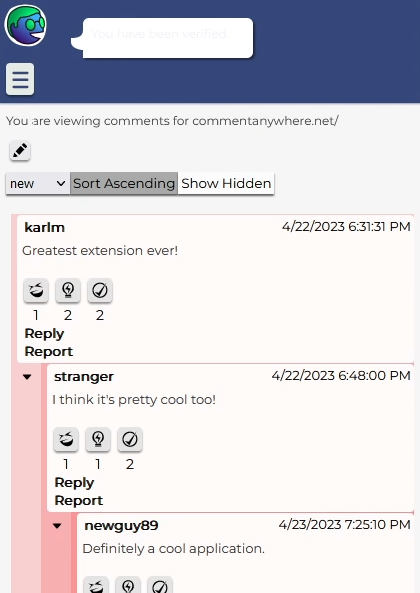
## Logging In

Navigate to the Login page using the navigation pane. Provide your previously registered username and password. Press the login button to continue. If your login credentials are causing issues, divert your attention to the Comment Anywhere Guy’s speech bubble. Validate that there is no error fetching from the server. Also, ensure that your credentials are correct. If you forgot your password, you may reset your password from the Login page. However, you may only reset your password if you previously validated your email.

## Logging Out

Click on the hamburger button to access the navigation pane. If you were previously logged in, a Logout button will be revealed in the navigation pane. Select this button to log out of your account.

## Viewing Comments



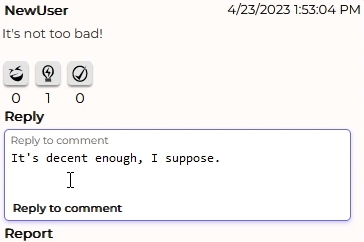
*Figure 6.7. Viewing Comments*

Guests and registered members have the ability to view comments for any given webpage. Using Firefox, navigate to any desired webpage. Open the extension and ensure that you are on the Comments page. If not, open the navigation pane, and navigate to the Comments page. Users may only interact with comments if they are logged in. If you are unable to view comments, ensure that you are on a valid webpage. If problems persist, you may attempt to utilize the Pseudo URL feature, which will be discussed shortly.

## Viewing Comments for a Pseudo-URL

Navigate to the Comments page using the navigation pane. Select the pencil icon towards the top of the application. You will be presented with an input field, where you may specify a URL to view comments for. Click the search button, represented by an hourglass, to search for your specified URL. You may hide this element by clicking on the pencil icon again, but the Pseudo URL will still be active. To cancel the Pseudo URL, click the cancel button.

## Replying to Comments



*Figure 6.9. Replying to Comments*

Navigate to the Comments page using the navigation pane. If there are comments present, you may select reply to reply to that comment. Type out your message and select “reply to comment”.

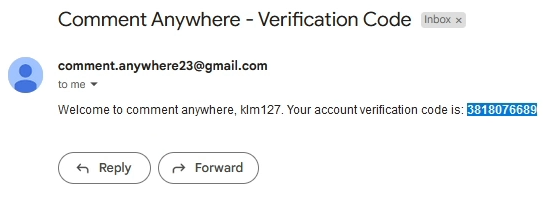
## Reporting Comments

Navigate to the Comments page using the navigation pane. If there is a comment you find to be inappropriate for our platform, you may select the report button. Provide your reasoning, and press “submit report”. You may not report that comment for a second time.

## Validating an Email

Navigate to the Settings page using the navigation pane. Select the “Request code” button. Leave the browser extension, and log into your email account. Select the email from [commentanywhere23@gmail.com](mailto:commentanywhere23@gmail.com), and copy the code provided in the email. Navigate back to the extension, and paste the code into the “verification code” box, and select verify. The server should report that your email has been verified.





*Figure 6.11. Validating an Email*

## Resetting a Password

If you are logged out, navigate to the login page from the navigation pane. Select “forgot your password”, and provide your previously verified email. You will be emailed a verification code, and may use it to change your password. If you are logged in, navigate to the settings page, and select “Request password reset”. You will now go through the similar process of someone who is logged out.



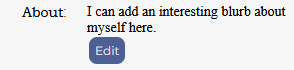
*Figure 6.12. The Reset Password Button*

## Changing an Email

Navigate to the Settings page from the navigation pane. Select the “change email” button and provide your password and new email. If your password is correct, and your email is formatted correctly, the change will be emitted.

## Changing a Profile Blurb

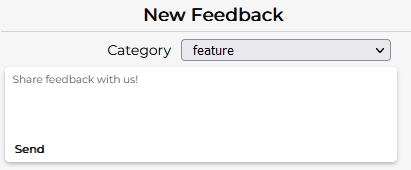
Navigate to the Settings page from the navigation pane. Select the “edit” button presented next to the “about” label. Enter the desired contents of your blurb and click submit.



*Figure 6.14. Changing a Profile Blurb*

## Submitting Feedback

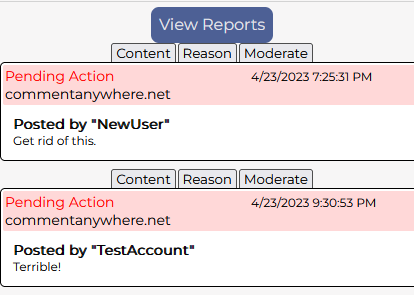
Navigate to the Settings page from the navigation pane. Select the category of feedback you would like to provide and fill out the text section with your desired response. Click submit to post your feedback to be viewed by an administrator.



*Figure 6.15. Submitting Feedback*

## Moderator: Viewing Reported Comments

If you are logged in as a moderator, navigate to the moderate page from the navigation pane. Focus your attention on the “Comment Reports” section. Click the button to query a list of reported comments. You may view the contents of the reported comment, and the reason for why it was reported.



*Figure 6.16. Viewing Reported Comments*

## Moderator: Moderating Comments

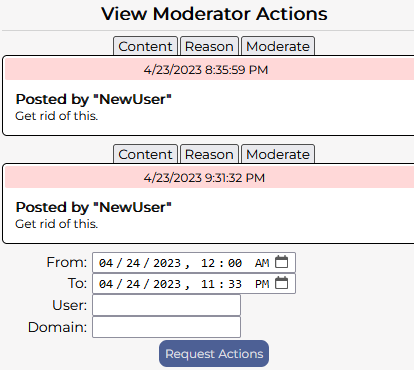
If you are logged in as a moderator, navigate to the moderation page from the navigation pane. Upon your query of reported comments, you may select the moderate tab, and be presented with the options to hide or remove the relevant comment. Provide a reason for your decision and submit.

## Moderator: Banning Users

If you are logged in as a moderator, navigate to the moderation page from the navigation pane. Focus your attention on the “Ban Users” section, in which you will be presented with a series of input fields. It is required that you specify an existing user to ban. You may provide a domain to ban the user from. You may also leave this field blank to emit a global ban on this user. Specify your reason, and submit.

## Moderator: Viewing Moderator Actions

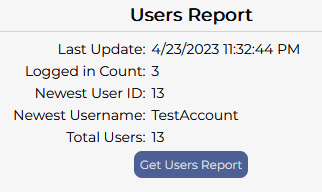
To view moderator actions, the moderator may input a From and To date range to filter for. They may also optionally filter by User and Domain. A series of UI element will appear displaying moderator actions that have been taken that meet the requested criteria. The viewing moderator can view the original content and the reason it was moderated by clicking through the tabs.



*Figure 6.19.* *Viewing Moderator Actions*

## Admin: Viewing the Users Report

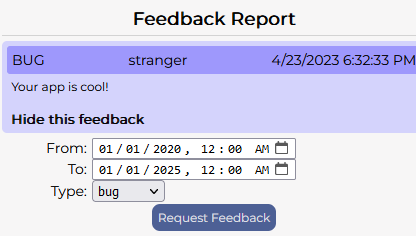
To view the user report, the admin clicks “Get Users Report”. This will populate the fields of the user report area, indicating to the admin the total number of users, newest user, and how many are currently logged in.



*Figure 6.20. Viewing the Users Report*

## Admin: Viewing the Feedback Report

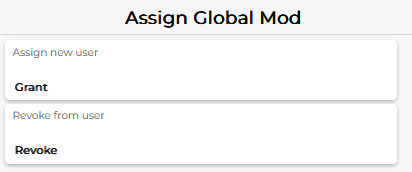
To view the Feedback report, the Administrator selects a date range they would like to view. The date inputs default to the previous week. They may select a type of feedback to filter for, or select “all” to see all. After clicking “Request Feedback”, feedback that users have submitted will be visible. The admin can click “Hide this feedback” to prevent a feedback from showing on the next report, if they don’t want to see it again.



*Figure 6.21. Viewing the Feedback Report*

## Admin: Assigning and Removing a Global Moderator

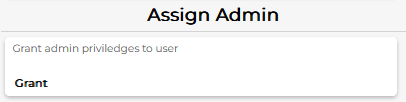
To assign or remove a global moderator, the Admin types the user’s name in the relevant text area under where it says either “Assign” or “Revoke”. They then click “Grant” or “Revoke”, respectively. Global Moderator permissions will be immediately granted to or revoked from the user that they named, if that user exists.



*Figure 6.22. Assigning and Removing Global Moderators*

## Admin: Assigning Admin Privileges

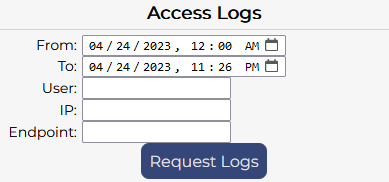
To assign admin privileges to a user, the Admin types the user’s name in the text area under where it says “Grant admin privileges to user.” They then click the bolded Grant button. If the user they named exists, that user will be granted admin privileges immediately or on the next login.



*Figure 6.23. Assigning an Admin*

## Admin: Viewing Access Logs

Admins can view access logs which provide some information about every IP address that has accessed Comment Anywhere. They can filter by a date range, a user, an IP, or an endpoint that was accessed. These filters are not applied if left blank. Each input is filled out with text or by selecting a date and the “Request Logs” button is clicked to retrieve the logs from the server, which are then displayed.



*Figure 6.24. Viewing Access Logs*

# Glossary

**API**

An API (Application Programming Interface) is a software interface that allows computers to request, retrieve, and exchange data and information in a standardized way. (2)

Back End

A Back End is any part of a website or software program the users do not see. It contrasts with the Front End, which refers to a program or website’s user interface

Browser Extension

“An extension adds features and functions to a browser. It is created using familiar web-based technologies – HTML, CSS, and JavaScript. It can take advantage of the same web APIs as JavaScript on a web page, but also has access to its own set of APIs.” (3) (3)

Class

Classes are a template for creating objects.

Container

“A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package that includes everything needed to run an application.” (4) (4)

Context (Go)

“The context type carries… request-scoped values across API boundaries and between processes.” (5)

**Cookies**

“An HTTP cookie (web cookie, browser cookie) is a small piece of data that a server sends to a user's web browser. The browser may store the cookie and send it back to the same server with later requests. Typically, an HTTP cookie is used to tell if two requests come from the same browser—keeping a user logged in, for example. It remembers stateful information for the stateless HTTP protocol.” (6)

Cross Origin Resource Sharing

“Cross-Origin Resource Sharing is an HTTP-header based mechanism that allows a server to indicate any origins and other than its own from which a browser should permit loading resources. CORS also relies on a mechanism by which browsers make a “preflight” request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.” (7)

Docker

“Docker is an open platform for developing, shipping, and running applications [and it] provides the ability to package and run an application in a loosely isolated environment called a container.” (8)

Dockerfile

“A Dockerfile is a text document that contains all the commands [necessary to] assemble a [container].” (9)

**Front End**

“HTML, CSS, and JavaScript are front-end (or client-side) languages, which means they are run by the browser to produce a website front-end that your users can use.”

Generic (Typescript)

**Go**

“The Go programming language is an open source project to make programmers more productive. Go is expressive, concise, clean, and efficient. Its concurrency mechanisms make it easy to write programs that get the most out of multicore and networked machines, while its novel type system enables flexible and modular program construction. Go compiles quickly to machine code yet has the convenience of garbage collection and the power of run-time reflection. It's a fast, statically typed, compiled language that feels like a dynamically typed, interpreted language.” (10) (10)

Headless process

A headless, or detached process is a process which is not attached to a terminal or other user interface. It runs in the background of the system.

**HTMLElement**

“The HTMLElement interface represents any HTML element. Some elements directly implement this interface, while others implement it via an interface that inherits it.” (11) (12)

**HTTP**

“Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests.” (13)

**HTTP Header**

“HTTP headers let the client and the server pass additional information with an HTTP request or response. An HTTP header consists of its case-insensitive name followed by a colon (:), then by its value. Whitespace before the value is ignored.” (14)

**HTTP Request**

“Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called requests” (15) ” HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be safe, idempotent, or cacheable.” (16) Request methods include : “GET HEAD POST PUT DELETE CONNECT OPTIONS TRACE PATCH” (16)

**HTTP Response**

“Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called requests and the messages sent by the server as an answer are called responses.” (15) “HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes: Informational responses (100 – 199) Successful responses (200 – 299) Redirection messages (300 – 399) Client error responses (400 – 499) Server error responses (500 – 599)” (17)

**JSON**

“JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).” (18)

JSON Web Token

“JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.” (19)

**manifest.json**

“The manifest.json file is the only file that every extension using WebExtension APIs must contain. Using manifest.json, you specify basic metadata about your extension such as the name and version, and can also specify aspects of your extension's functionality (such as background scripts, content scripts, and browser actions). It is a JSON-formatted file, with one exception: it is allowed to contain "//"-style comments.” (20)

**Method**

“A method is a function which is a property of an object. There are two kinds of methods: instance methods which are built-in tasks performed by an object instance, or static methods which are tasks that are called directly on an object constructor.” (21)

**Module**

“A module is just a file. One script is one module. As simple as that. Modules can load each other and use special directives export and import to interchange functionality, call functions of one module from another one” (22)

**Mozilla Firefox**

“Mozilla Firefox is a free open-source browser whose development is overseen by the Mozilla Corporation. Firefox runs on Windows, OS X, Linux, and Android. First released in November 2004, Firefox is completely customizable with themes, plug-ins, and add-ons. Firefox uses Gecko to render webpages, and implements both current and upcoming Web standards.” (23)

**namespace (typescript)**

“Namespaces are a TypeScript-specific way to organize code. Namespaces are simply named JavaScript objects in the global namespace. This makes namespaces a very simple construct to use. Unlike modules, they can span multiple files, and can be concatenated using outFile. Namespaces can be a good way to structure your code in a Web Application, with all dependencies included as <script> tags in your HTML page.” (24)

NodeJS

NodeJS is “an asynchronous event-driven JavaScript runtime.” (25) It is used in Comment Anywhere to access the node package manager (npm) in order to implement TypeScript, WebPack, and other dependencies the Front End needs

Npm

Npm is the dependency manager for NodeJS.

**Queries (Database)**

“Using a query makes it easier to view, add, delete, or change data in your Access database.” (26) “A query can pull the information from various tables and assemble it for display in the form or report. A query can either be a request for data results from your database or for action on the data, or for both. A query can give you an answer to a simple question, perform calculations, combine data from different tables, add, change, or delete data from a database.” (26)

**Router (Go)**

“It implements a request router and dispatcher for matching incoming requests to their respective handler.” (27) “Like the standard http.ServeMux, mux.Router matches incoming requests against a list of registered routes and calls a handler for the route that matches the URL or other conditions.” (27)

**Schema**

“schema is a Go package providing access to database schema metadata, for database/sql drivers.” (28)

**Server**

“Servers are computers that store webpages, sites, or apps. When a client device wants to access a webpage, a copy of the webpage is downloaded from the server onto the client machine to be displayed in the user's web browser.” (12)

Server-Client communication entity

**SQL**

“SQL Server provides commands that are not Transact-SQL statements, but are recognized by the sqlcmd and osql utilities and SQL Server Management Studio Code Editor. These commands can be used to facilitate the readability and execution of batches and scripts. GO signals the end of a batch of Transact-SQL statements to the SQL Server utilities.” (29)

**Sqlc**

“sqlc generates type-safe code from SQL.” (30)” You write queries in SQL. You run sqlc to generate code with type-safe interfaces to those queries. You write application code that calls the generated code.” (30)

**struct (Go)**

“Structs are a way to structure and use data. It allows us to group data.” (31) ” define a struct type using the type keyword.” (31)

**Table (Databases)**

“Tables are essential objects in a database because they hold all the information or data. For example, a database for a business can have a Contacts table that stores the names of their suppliers, e-mail addresses, and telephone numbers.” (32)

**URL**

“With Hypertext and HTTP, URL is one of the key concepts of the Web. It is the mechanism used by browsers to retrieve any published resource on the web.” (33) ” URL stands for Uniform Resource Locator. A URL is nothing more than the address of a given unique resource on the Web.” (33)

**Vite**

“Vite (French word for "quick", pronounced /vit/, like "veet") is a build tool that aims to provide a faster and leaner development experience for modern web projects. It consists of two major parts: A dev server that provides rich feature enhancements over native ES modules, for example extremely fast Hot Module Replacement (HMR). A build command that bundles your code with Rollup, pre-configured to output highly optimized static assets for production.” (34)

**Vitest**

“Vitest provides a compatible API that allows you to use it as a drop-in replacement in most projects. It also includes the most common features required when setting up your unit tests (mocking, snapshots, coverage).” (35)

**VPS**

“A virtual private server(VPS) is a machine that hosts all the software and data required to run an application or website. It is called virtual because it only consumes a portion of the server's underlying physical resources which are managed by a third-party provider. However, you get access to your dedicated resources on that hardware.” (36)

**Waterfall Design**

“A “waterfall” project management method is also sequential. One of the traditional project management tools, it calls for a strictly linear progression of tasks that leads to project completion. It heads only in one direction, like a waterfall. The waterfall method works best when team members rely on the completion of other tasks before starting their own. It also works best if there aren’t likely to be many big surprises or changes along the way. A waterfall method allows projects to scale well, as it is a design-heavy and disciplined approach that can keep even large projects in check.” (37)

# Bibliography

1. **Bode, Karl.** Killing Website Comment Sections Wasn't the Brilliant Move Many Newsroom Leaders Assumed. [Online] Techdirt, November 8, 2021. [Cited: April 23, 2023.] https://www.techdirt.com/2021/11/08/killing-website-comment-sections-wasnt-brilliant-move-many-newsroom-leaders-assumed/.

2. **IEEE (Institute of Electrical and Electronics Engineers).** What is an API? | IEEE Xplore. [Online] IEEE (Institute of Electrical and Electronics Engineers). [Cited: 23 04, 2023.] https://developer.ieee.org/What\_is\_an\_API.

3. **MDN Contributors.** What are extensions? - Mozilla | MDN. [Online] November 15, 2022. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/What\_are\_WebExtensions.

4. **Docker.** What is a Container? - Docker. [Online] December 4, 2022. https://www.docker.com/resources/what-container/.

5. **Go contributors.** context package - Context - Go packages. [Online] [Cited: April 23, 2023.] https://pkg.go.dev/context.

6. **Mozilla Corporation.** Using HTTP cookies - HTTP | MDN. [Online] Mozilla Corporation, 04 10, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

7. **Cross-Origin Resource Sharing. *mdn web docs.* [Online] April 10, 2023. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.**

**8. Docker. Docker overview | Docker Documentation. [Online] [Cited: April 23, 2023.] https://docs.docker.com/get-started/overview/.**

**9. —. Dockerfile reference. [Online] [Cited: April 23, 2023.] https://docs.docker.com/engine/reference/builder/.**

**10. GO Corporation. Documentation - The Go Programming Language. [Online] GO Corporation. [Cited: 04 23, 2023.] https://go.dev/doc/.**

**11. Mozilla Corporation. HTMLElement - Web APIs | MDN. [Online] Mozilla Corporation, 04 13, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement.**

**12. —. How the web works - Learn web development | MDN. [Online] Mozilla Corporation, 03 20, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Learn/Getting\_started\_with\_the\_web/How\_the\_Web\_works.**

**13. —. HTTP | MDN. [Online] Mozilla Corporation, 03 02, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Web/HTTP.**

**14. —. HTTP headers - HTTP | MDN. [Online] Mozilla Corporation, 04 15, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/web/http/headers.**

**15. —. An overview of HTTP - HTTP | MDN. [Online] Mozilla Corporation, 04 10, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.**

**16. —. HTTP request methods - HTTP | MDN. [Online] Mozilla Corporation, 04 10, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.**

**17. —. HTTP responce status codes - HTTP | MDN. [Online] Mozilla Corporation, 04 10, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.**

**18. —. Working with JSON - Learn web development | MDN. [Online] Mozilla Corporation, 01 23, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON.**

**19. Jones, M, et al. ietf.org. [Online] Internet Engineering Task Force (IETF), May 2015. [Cited: April 23, 2023.] https://datatracker.ietf.org/doc/html/rfc7519.**

**20. Mozilla corporation. Manifest.json - Mozilla | MDN. [Online] Mozilla corporation, 03 07, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json.**

**21. Mozilla Corporation. Method - MDN Web Docs GLossary : Definintitions of Web-related terms | MDN. [Online] Mozilla Corporation, 02 21, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Glossary/Method.**

**22. Ilya Kantor. Modules, Introduction. [Online] Javadcript.Info, 10 14, 2022. [Cited: 04 23, 2023.] https://javascript.info/modules-intro.**

**23. Mozilla corporation. Mozilla Firefox - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. [Online] Mozilla corporation, 02 21, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Glossary/Mozilla\_Firefox.**

**24. Microsoft. TypeScript Documentation - Namespaces and Modules. [Online] Microsoft Corporation, 04 11, 2023. [Cited: 04 23, 2023.] https://www.typescriptlang.org/docs/handbook/namespaces-and-modules.html.**

**25. NodeJS Contributors. About | Node.js. [Online] [Cited: April 23, 2023.] https://nodejs.org/en/about/.**

**26. Google. Introduction to queries - Microsoft Support. [Online] Google LLC. [Cited: 04 23, 2023.] https://support.microsoft.com/en-us/office/introduction-to-queries-a9739a09-d3ff-4f36-8ac3-5760249fb65c.**

**27. Majumdar, Arkaprabha. Golang Mux Router. [Online] Golang Docs, 09 26, 2020. [Cited: 04 23, 2023.] https://golangdocs.com/golang-mux-router.**

**28. Jim Smart. schema package - github.com/jimsmart/schema - Go packages. [Online] Go corporation, 01 05, 2023. [Cited: 04 23, 2023.] https://pkg.go.dev/github.com/jimsmart/schema@v0.2.1.**

**29. Microsoft. SQL Server Utilities Statements - GO - SQL Server | Microsoft Learn. [Online] Microsoft, 02 28, 2023. [Cited: 04 23, 2023.] https://learn.microsoft.com/en-us/sql/t-sql/language-elements/sql-server-utilities-statements-go?view=sql-server-ver16.**

**30. Kyle Conroy. sqlc package - github.com/kyleconroy/sqlc - Go Packages. [Online] GO Corporation, 02 23, 2023. [Cited: 04 23, 2023.] https://pkg.go.dev/github.com/kyleconroy/sqlc#section-readme.**

**31. Chakraborty, Sutirtha. Structs in Golang - Golang Docs. [Online] Golang Docs, 01 01, 2020. [Cited: 04 23, 2023.] https://golangdocs.com/structs-in-golang.**

**32. Microsoft. Introduction to tables - Microsoft Support. [Online] Microsoft Corporation. [Cited: 04 23, 2023.] https://support.microsoft.com/en-us/office/introduction-to-tables-78ff21ea-2f76-4fb0-8af6-c318d1ee0ea7#:~:text=In%20a%20well-designed%20database%2C%20each%20table%20stores%20data,data%2C%20such%20as%20text%2C%20numbers%2C%20dates%2C%20and%20hyperlinks..**

**33. Mozilla Corporation. What is a URL? - Learn web development | MDN. [Online] Mozilla Corporation, 02 23, 2023. [Cited: 04 23, 2023.] https://developer.mozilla.org/en-US/docs/Learn/Common\_questions/Web\_mechanics/What\_is\_a\_URL.**

**34. Vite. Getting Started | Vite. [Online] Vite Corporation. [Cited: 04 23, 2023.] https://vitejs.dev/guide/.**

**35. Vitest. Why Vitest | Guide | Vitest. [Online] Vitest Corporation, 07 14, 2022. [Cited: 04 23, 2023.] https://vitest.dev/guide/why.html.**

**36. Amazon Web Services. What is VPS? - Virtual Private Server Explained - AWS. [Online] Amazon Corporation. [Cited: 04 23, 2023.] https://aws.amazon.com/what-is/vps/.**

**37. Microsoft 365 Team. Starter guide for project management. [Online] Microsoft Corporation, 10 19, 2019. [Cited: 04 23, 2023.] https://www.microsoft.com/en-us/microsoft-365/business-insights-ideas/resources/guide-for-project-management.**

**38. Wheeler, Sean, Lombardi, Mikey and Richie, Peter. about Environment Variables Powershell Microsoft Learn. [Online] Microsoft, 11 29, 2022. [Cited: 04 23, 2023.] https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\_environment\_variables?view=powershell-7.3.**

**39. Google. Chrome Extentions: Design the user interface - Chrome Developers. [Online] Google LLC, 04 27, 2022. [Cited: 04 23, 2023.] https://developer.chrome.com/docs/extensions/mv3/user\_interface/.**

**40. NodeJS Contributors. About | Node.js. [Online] [Cited: December 4, 2022.] https://nodejs.org/en/about/.**

# Appendix

## Team Details, Individual Contributions

Karl Miller wrote the Project Overview, Motivation, Implementation Details, and Software Engineering Principles of this document. He coded 98% of the Back End and 80% of the Front End.

Luke created the System Block Diagram. He coded 20% of the Front End and 2% of the Back End. He created all artwork used for Comment Anywhere, had invaluable ideas for unique features, and was the key designer of the entire user interface style.

Frank contributed some of the glossary definitions. He created power points for some of the weekly presentations. He submitted a rough draft for the moderation panel.

## Code Listing

The Comment Anywhere source code is proprietary. It will be submitted separately to the professor.

## Workflow Authentication