Comment Anywhere

Pennwest California

CSC 490: Senior Project 1

Specification Document

Dr. Chen

11/15/22

Group Members

Luke Bates	Computer Science	Implementation
Frank Bedekovich	Computer Science	Specifications & Analysis
Robert Krency	Computer Science	Requirements
Karl Miller	Computer Science	Design

Instructor comments / Evaluation

Table of Contents

Abstract

We describe the results of Software Specifications Engineering in a Waterfall model as applied to our proposed Software Product, Comment Anywhere, a browser extension for commenting on webpages. In this document, the specifications of the product are described. After characterizing the intended users, we specify the constraints of Comment Anywhere including costs constraints, time constraints, hardware constraints, software constraints and time and resource constraints. Next, we describe testers and acceptance test criteria. We then provide System Modeling through a sample Use Case scenario, a complete description of all Entity classes and a Dynamic State Chart of Front End states. Finally, we provide an example Dataflow Diagram for user registration and activation and list the tools needed to develop the product.

Description of Document

Purpose and Use

The purpose of this document is to finalize all specific requirements for the project. The requirements include hardware, software and cost constraints, how the product will be utilized, and much of the product design. The Comment Anywhere team and the client will use this document to specify what is and isn't allowed to be done with the product. Once the client is satisfied with this document, it will become a binding contract between the team and client.

Intended Audience

The intended audience of this document is the Comment Anywhere developers and the client, the Comment Anywhere business team and Professor Chen. The client will inspect this document to make sure it meets all their needs. If the client doesn't like a section of the document, it will be revised until the client and the development team are satisfied with the terms. These matters must be solved in this stage in order to have the exact specifications for the development of the product which will satisfy the client’s needs.

System Description

Overview

Comment Anywhere is a web service that allows users to submit and view comments on any webpage on the Internet. Comments are viewed by utilizing the Front End, which is a browser extension for Mozilla Firefox and Chromium based browsers. The Back End portion of the service hosts the comments and user information, executes the registering and logging-in of users, the serving of comments, the storing of new comments, filtering prohibited comments, and moderating comments.

Environment and Constraints

End User Profile

Potential End Users for Comment Anywhere include anyone with Internet Access and either a Chromium based browser which supports chrome extensions or Mozilla Firefox installed on their computer. Users must be able to read and comment in English; Localization is infeasible to support for the initial roll out due to time and cost constraints.

The primary target audience will be those who are willing to try a new Social Media Platform with a small user base. As such, the platform will be made appealing to individuals dissatisfied with current social media and commenting options. We will allow free expression and controversial views that are permissible within U.S. Law to attract users who seek corners of the Internet with less censorship.

User Interaction

The bulk of End Users of Comment Anywhere are expected to be what we term Guests. Guests are anyone utilizing the Comment Anywhere Front End who have not created an account.

They are able to automatically request Comments for the web page they are on by opening the Browser Extension or navigating to a new page with the Extension open. They may not post or report comments, and some moderated comments may be invisible to them. They may submit a registration form to create an Account or a login form to log into an existing account.

Users who have created and logged into an account are termed Members. They can still view comments but are additionally able to post comments and configure their personalized settings to view hidden comments that may contain offensive material. They are also able to report comments that contain offensive material.

Administrators are users who are promoted, approved, and assigned by the Comment Anywhere business team. They are responsible for monitoring the performance of Comment Anywhere. They will be able to request and receive reports on User Activity on Comment Anywhere. They can also interact with the software in the same ways as users with basic permissions, viewing comments, posting comments, reporting comments, moderating comments, and so forth. However, they can also select Members to elevate to the status of Moderators, review moderation actions, and remove Moderators.

Moderators are Members who are promoted by Administrators. They are expected to primarily be volunteers, due to cost constraints. They are tasked with reviewing comments that have been reported and permitted to take moderation actions on them. They can remove a comment that violates U.S. Law, which will make its content invisible to most users. They can "hide" comments that are objectionable but not illegal, making them invisible except to Members who have configured their settings to view offensive comments. They may ban Members who have violated the rules of Comment Anywhere. Moderators are divided into Global Moderators and Domain Moderators. Domain Moderators are permitted to moderate only comments on

particular domains and may ban users only from commenting on that domain. Global Moderators may assign additional Domain Moderators or remove them and may moderate all comments across the service. Global Moderators may review moderation actions.

Hardware Constraints

Users of Comment Anywhere will likely only be able to access Comment Anywhere from a PC environment. Mobile browsers do not currently support Browser Extensions. Therefore, Users of Comment Anywhere will need a desktop or laptop capable of running a standard internet browser and must have an internet connection.

Our Server must initially run on hardware capable of handling at least dozens of requests per minute. It must be capable of hosting our database, which will be relatively small initially. We must be able to scale as our user base and database grows, but the hardware requirements for launch are expected to be minimal. More discussion of cloud constraints are discussed in the Costs Constraints section.

One advantage of writing a server for a browser extension is that we do not have to serve HTML, CSS, and JavaScript files. All these files are downloaded by users in advance and stored on their hard drives. We only need to serve small JSON strings in response to HTTP Requests.

Software Constraints

Users must have access to a Chromium or Firefox browser that supports browser extensions to interact with Comment Anywhere. It will function on any operating system that supports those Browsers. It may incidentally support other browsers.

For the Back End, there are software constraints relating to the Cloud. The business team requires flexibility in deployment so that they can pick any number of cloud hosting providers

based on cost and speed needs. For maximum flexibility, the Back End will be designed to run inside a Docker Container, which is a virtualized, semi-independent operating system that packages all dependencies with the code. We will choose a low-cost cloud provider for the initial roll out. This cloud provider must support running of Docker Containers. Examples of Providers that offer Cloud Hosting for Docker containers include:

- Amazon Elastic Container Service
- Microsoft Azure
- Kamatera
- A2 Hosting
- Jelastic
- StackPath
- Google Cloud Run
- Sloppy.io
- HostPresto
- Vultr

Time Constraints

Our project has several deliverables with deadlines. The design document must be completed by November 15th. The coding and testing must be completed by the end of the spring semester in 2023, which will be around May 5th. All members of the Comment Anywhere team are full time students and cannot allocate as much time to one class as a full-time worker could dedicate to coding a single project. Nevertheless we are certain we will be able to complete the project within the deadlines and meet all specifications.

Cost Constraints

Comment Anywhere will be deployed to a Cloud Hosting provider, such as Amazon Web Services or Google Cloud. We estimate that our initial cloud hosting costs could be anywhere from $40 to $200 per month, (Luenendonk) but this is highly dependent on the traffic our server receives. We researched Google Cloud hosting, which has a free tier. We need to keep our hosting costs minimal as we grow, because there will be few or no monetization sources initially. Google Run Free Tier provides up to 180,000 vCPU seconds per month for processing, 360,000 GiB seconds per month for memory, and 2 million requests per month. After that, the price is $0.000024 per vCPU second for processing, $0.0000025 per GiB-second for memory, and $0.40 per million requests ("Cloud Run: Container to production in seconds"). We are not currently able to say for certain by how much our costs, if any, will exceed the free tier. While there are more than 180,000 seconds in a month, for instance, there are different CPU allocation settings available. One setting allows us to only use CPU time when a request is being processed. By selecting this option, we may be able to keep our costs within the free tier. During the implementation phase, we will use Google Cloud to test the resource usage of Comment Anywhere, then determine whether we need to optimize, whether we can continue with Google Cloud, or whether we can utilize another provider, such as HostGator, which is $4.95 per month for 2 CPUs and 2 GB of memory. (HostGator)

We also need a domain name. We purchased the domain commentanywhere.net for two years along with privacy protection, for a price of $45.08.

Other Concerns

Security should be implemented such that any user input is assumed to be malicious. We must guard our system against attacks such as SQL Injection, which is when a malicious actor

supplies user input that will harm a database when it is inserted. We must also ensure data security by encrypting plain text data sent by users, especially passwords. This can be accomplished with HTTPS. We must also secure our data against malicious actors from within, by not storing user passwords in a form accessible by administrators, which can be accomplished with encryption.

Acceptance Test Criteria

Testers

The individuals testing this product will be Karl Miller, Luke Bates, Frank Bedekovich, Robert Krency, and Professor Chen.

Criteria for User Acceptance

Comment Anywhere must be able to perform a number of functions, which vary depending on the access level of the user.

Guest

- A Guest may view and sort comments associated with the current webpage. Guests cannot post, rate, or report comments.
- Register a Comment Anywhere account. This requires a username, email, and password. The user will be emailed a short verification code to verify their email account.
- A Guest can log into a Comment Anywhere account by entering their username and password.
- If they incorrectly specify their username or password, they will be prompted to try again or reset their password. Upon reset, they will be emailed a verification code, which is required to continue the password reset process.

- If the user has forgotten their username, they must provide the verified email address for their Comment Anywhere account. If the provided email has not been previously verified, the user may not continue.

Member

- Members are able to view, rate, and report comments. They may also post comments, and reply to other users' comments.
- Members can rate comments as one or more categories, some of which being informative or funny.
- Members can report comments by providing a reason and clicking submit.
- Members can post comments in the main thread, or reply to other users' comments.
- Members may reset their email by providing their password. Upon success, they must reverify their new email.
- Members may choose to view hidden comments.
- Members may submit feedback, including bug reports and feature requests.

Moderator

- Moderators are able to review the list of reported comments, where they can claim a comment to review. They have the ability to hide or remove any comment. Moderators must provide a reason when removing comments or banning users.
- The Comment Anywhere extension interface offers more features for moderation as compared to a standard member.

Administrator

- Administrators can assign global and domain moderators via the Comment Anywhere extension interface.

- Administrators can view logs containing user IP addresses and the website they were viewing.

GUI

- Installing the Comment Anywhere extension will add a button on the browser, which a browser user can click to open the Comment Anywhere interface.

Integration of Separate Parts and Installation

Users must install the browser extension. While we will endeavor to place Comment Anywhere in the Extension marketplaces of Firefox and Chrome, acceptance is not guaranteed. Users will therefore also be able to download the extension from our statically hosted website. By installing and activating our extension, a button is made available in the user's browser which initializes communication with the server.

The Server has two main parts that must be integrated; the Database and the HTTP Server. These may be configured to run in separate Virtual Machines or in a single Virtual Machine. The Database and the HTTP Server must be able to communicate. This Virtual Machine(s) must be installed on the cloud servers of the hosting provider we select.

The Front End integrates with the Back End through HTTP Requests and Responses. The Front End will be configured to respond to HTTP Requests by relevant changes in the Front End that realize the display of data requested. The Server is responsible for parsing requests and providing the relevant data.

System Modeling

Upon using our browser extension, it will first load up the comments for the current webpage at a Guest accessible level. If the Guest wants to continue and login, they will also be presented with a method to sign in or sign up to be a user. When a user is logged in they will be presented with the ability to view all publicly available comments and make a comment themselves.

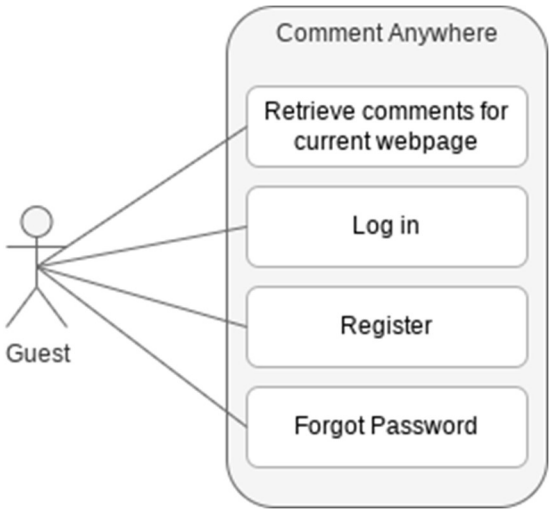


Figure 1. Flow of the average Guest on startup

Figure 1 above depicts the ways the Guest can interface with the extension upon startup. The Guest can see the comments for the current page they are on in a limited capacity. The Guest can also log in if they already have an account to become a guest or if they do not have an account, they can register to become a Member.

Run Scenario

- The guest opens the Comment Anywhere extension. They can view comments but cannot interact with them.
- The guest may register, log in, or reset their password.
    - If registration is chosen
        - They must provide their desired email, username, and password. Upon submission, this data will be associated with their newly created account.
        - They may exit registration at any time.
    - If login is chosen
        - They must provide their correct username and password.
        - They will be logged into their Comment Anywhere account.
    - If reset password is chosen
        - The user must provide the email associated with their account
        - A password reset link will be sent to their email.

Exception Scenario

- The guest opens the Comment Anywhere extension.
- The guest may register or log in.
    - If registration is chosen
        - They provide an existing username, email or insecure password.
        - They are prompted to try again.
    - If login is chosen
        - They provide a nonexistent username or incorrect password.

        - They are prompted to try again.
    - If reset password is chosen
        - The user provides the incorrect email.
        - No email is sent, and the user is not notified that the email does not exist in the database.

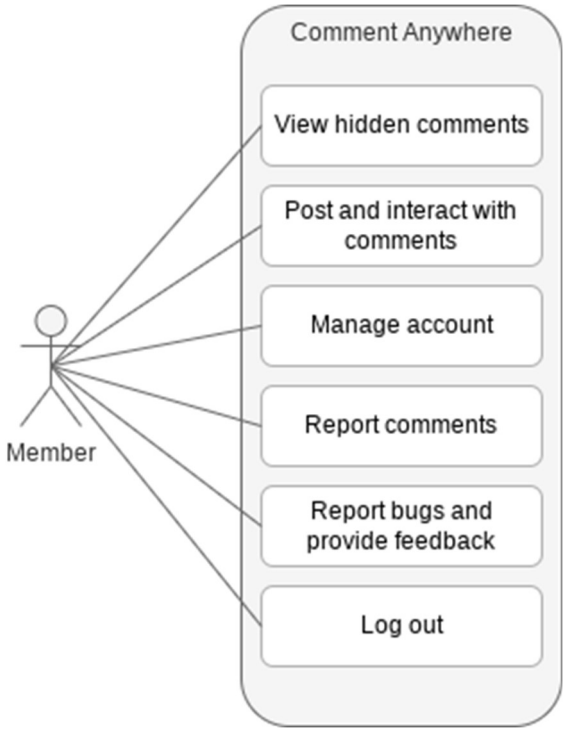


Figure 2. Flow of a User after just logging in

Figure 2 is of a Member that has just logged in to the extension. A member is able to see hidden comments that guests are unable to see. They can also create their own comments to be added to the database. They can interact, rate, reply, and report to other comments through various means. Users can also manage their own account, report any bugs that they find in the product, and log out of the extension.

Run scenario

- User opens the extension

- User is presented with some options
    - ❖ View hidden comments
    - ❖ Post and interact with comments
    - ❖ Manage account
    - ❖ Report comments
    - ❖ Report bugs and provide feedback
- If view hidden comments is chosen
    - ❖ The extension will pull up all comments for the current webpage
- If post and interact with comments is chosen
    - ❖ The user can reply to the comment
    - ❖ The user can rate the comment on a set of criteria
- If manage account is chosen
    - ❖ The user is able to change their account settings
    - ❖ The user can also go back at anytime from here
- If report comments is chosen
    - ❖ The user reports the selected comment through a feedback textbox and it is sent to the database to be reviewed
- If report bugs and provide feedback is chosen
    - ❖ The user is given a feedback textbox for reporting the bug and is sent to the database

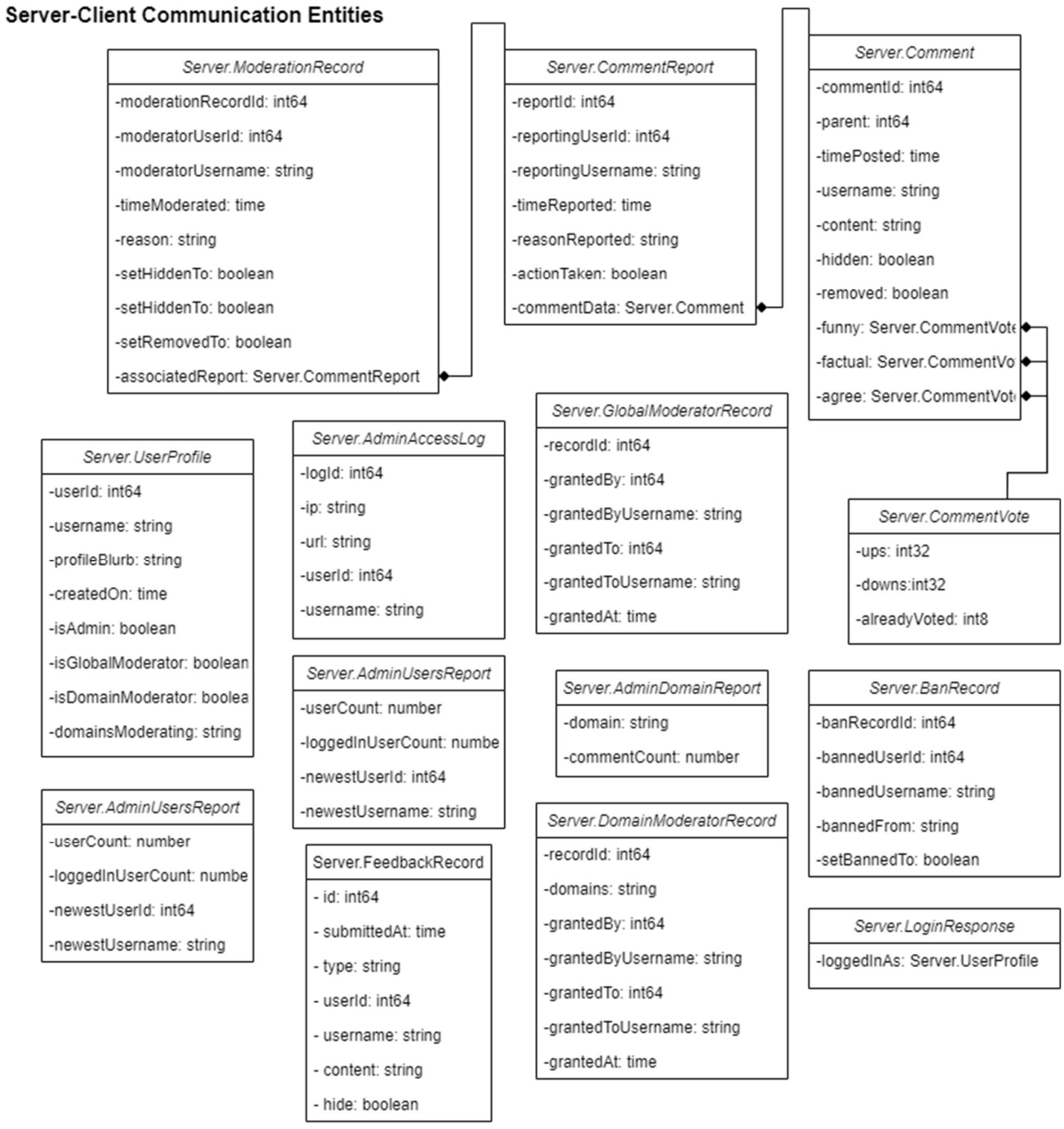
Entity Class Diagrams

Entity Classes model long-lived information. Our entities are subdivided into five categories. Server-Client Communication Entities are entities which are constructed by the

Server to send information to the Client in response to some request, and which the client uses to display data to the user. Client-Server Communication Entities are entities which are constructed by the Client in response to user input and are sent to the server to request or update data. Front End entities are entities which are stored locally in the client system. They may be accessed by Front End controllers as it assembles Client-Server Communication Entities but are not sent to the server directly. Database Entities are the raw entities which correspond to our Database schema, and which the Server utilizes in the construction of Server-Client communication entities and Cache Entities. Cache Entities are entities used to cache database information into server RAM to avoid excessive database queries.

Server-Client Communication Entities

The Server must serve the Front End client data which the front-end can display. It accomplishes this by querying the database and sending information within HTTP Responses in the form of JSON strings. Because the Client cannot query the database, the Back End must serve data which is the result of all necessary queries. For example, while Comment data on the Server needs only a User ID, because the Server can find the associated user in the database using that ID, when the Server sends the data to the client, it must provide the Client with the username, so that the client can display it. Therefore, it must query the Users table in advance to get the associated username as it builds the response. The Server may send a number of these entities as a collection in a single response, depending on the type of data the client needs. These classes are prefixed with the namespace “Server”, to distinguish them from similar identifiers.



Server.AdminAccessLog

Server.AdminAccessLog contains data needed by Admins to see an access log.

- **ip**: The IP of anyone accessing the logs.
- **logId**: The ID of the current log.

- **url**: The full URL that was accessed.
- **userId**: The ID of the user this access record is associated with, if the user was logged in. Otherwise, this field will be empty or 0.
- **username**: The username of the accessor If applicable

Server.AdminDomainReport

Server.AdminDomainReport contains data needed by Admins to see information about activity on a particular domain.

- **commentCount**: number of comments on a domain.
- **domain**: the string of the domain.

Server.AdminUsersReport

Server.AdminUsersReport is dispatched when an Admin requests the Users report.

- **loggedInUserCount**: The amount of users that are currently logged in.
- **newestUserId**: The newest user's ID to be created.
- **newestUsername**: The username of the newest user.
- **userCount**: The amount of users that have been made.

Server.BanRecord

Server.BanRecord contains data about a banning or unbanning which occurred, which is used by Admins to see information about Moderator actions in certain reports.

- **bannedFrom**: used for when a user is banned from a specific domain and not the extension as a whole.

- **banRecordId**: creates a record whenever someone is banned.
- **bannedUserId**: the id of a user that is banned.
- **bannedUsername**: the username of a user that is banned.
- **setBannedTo**: Whether the user was banned (true) or unbanned (false).

Server.Comment

Server.CommentVote provides the data the Front End needs to render a comment.

- **Agree**: An instance of Server.CommentVote, reflecting the number of "agree" and "disagree" votes.
- **commentId**: A number corresponding to the comment's unique ID.
- **content**: The text content of the comment.
- **factual**: An instance of Server.CommentVote, reflecting the number of "factual" and "not factual" votes.
- **funny**: An instance of Server.CommentVote, reflecting the number of "funny" and "unfunny" votes.
- **hidden**: A boolean value, true if the comment was moderated to be hidden.
- **parent**: The ID of the comment that is the parent of this comment, or 0 if it is a root-level comment.
- **removed**: A boolean value, true if the comment was removed.
- **timePosted**: The time the comment was posted.
- **username**: The username of the user who posted the comment.

Server.CommentReport

Server.CommentReport contains data the Front End needs to render a CommentReport, which are reports submitted by users and which Moderators can review and take action on.

- **actionTaken**: If the report has been addressed
- **commentData**: The data of a comment.
- **reasonReported**: The reason for reporting a comment.
- **reportId**: The unique ID of the report.
- **reportingUserId**: The unique ID of the user who reported the comment.
- **reportingUsername**: The name of the user that made the comment.
- **timeReported**: The time that a comment was reported at.

Server.CommentVote

Server.CommentVote contains data for the number of votes on a comment.

- **alreadyVoted**: Whether the user requesting the data has already voted on the comment. It will be -1 if they have already voted down, 0 if they have not voted, and 1 if they have already voted up.
- **downs**: The number of downvotes on the comment.
- **ups**: The number of upvotes on the comment.

Server.DomainModeratorRecord

Server.DomainModeratorRecord contains data needed by Admins to see information about DomainModerator assignments.

- **domains**: the domains the moderator is allowed to moderate
- **grantedAt**: When the user became aDomainModerator.
- **grantedBy**: The ID of the admin or GlobalModerator that promoted the user to a DomainModerator.
- **grantedByUsername**: The username of the admin or GlobalModerator that promoted the user to a DomainModerator.
- **grantedTo**: The ID of the DomainModerator.
- **grantedToUsername**: The username of the DomainModerator.
- **recordId**: The ID of the DomainModerators record.

Server.FeedbackRecord

Server.FeedbackRecord contains data the Front End needs to render a FeedbackRecord, which is a record of a user-submitted feedback, viewed by an Admin, such as a feature request, or bug report.

- **content**: The text of the feedback, limited to 1000 characters.
- **hide**: Whether or not this feedback is hidden, that is to say, the admins do not want to see it again by default.
- **id**: int64
- **some** other type of feedback, “general”.
- **submittedAt**: The time this feedback was submitted.
- **type**: Indicates whether this feedback is a feature request, “feature”, bug report “bug”, or

- **userid**: The ID of the user who submitted the feedback.
- **username**: The username of the user who submitted the feedback.

Server.GlobalModeratorRecord

Server.GlobalModerator record contains data needed by Admins to see information about GlobalModerator assignments.

- **grantedAt**: When the user became a GlobalModerator.
- **grantedBy**: The ID of the admin that promoted the user to a GlobalModerator.
- **grantedByUsername**: The username of the admin that promoted the user to a GlobalModerator.
- **grantedTo**: The ID of the GlobalModerator.
- **grantedToUsername**: The username of the GlobalModerator.
- **recordId**: The ID of the GlobalModerators record.

Server.LoginResponse

Server.LoginResponse is sent to the client when they successfully log in.

- **loggedInAs**: The profile of a user that logged in.

Server.ModerationRecord

Server.ModerationRecord contains data the Front End needs to render a ModerationRecord, which is a record of a moderator action, such as hiding or removing a comment.

- **associatedReport**: contains data the Front End needs to render a CommentReport, which are reports submitted by users and which Moderators can review and take action on.

- **moderationRecordId**: The ID of the moderator's past actions.
- **moderatorUserId**: The id of the moderator.
- **moderatorUsername**: The username of the moderator.
- **reason**: the reason the moderator decided to take action on the comment.
- **setHiddenTo**: boolean If the moderator has decided to hide the comment.
- **setRemovedTo**: boolean If the moderator has decided to remove the comment.
- **timeModerated**: The current time that the moderator took action on a reported comment.

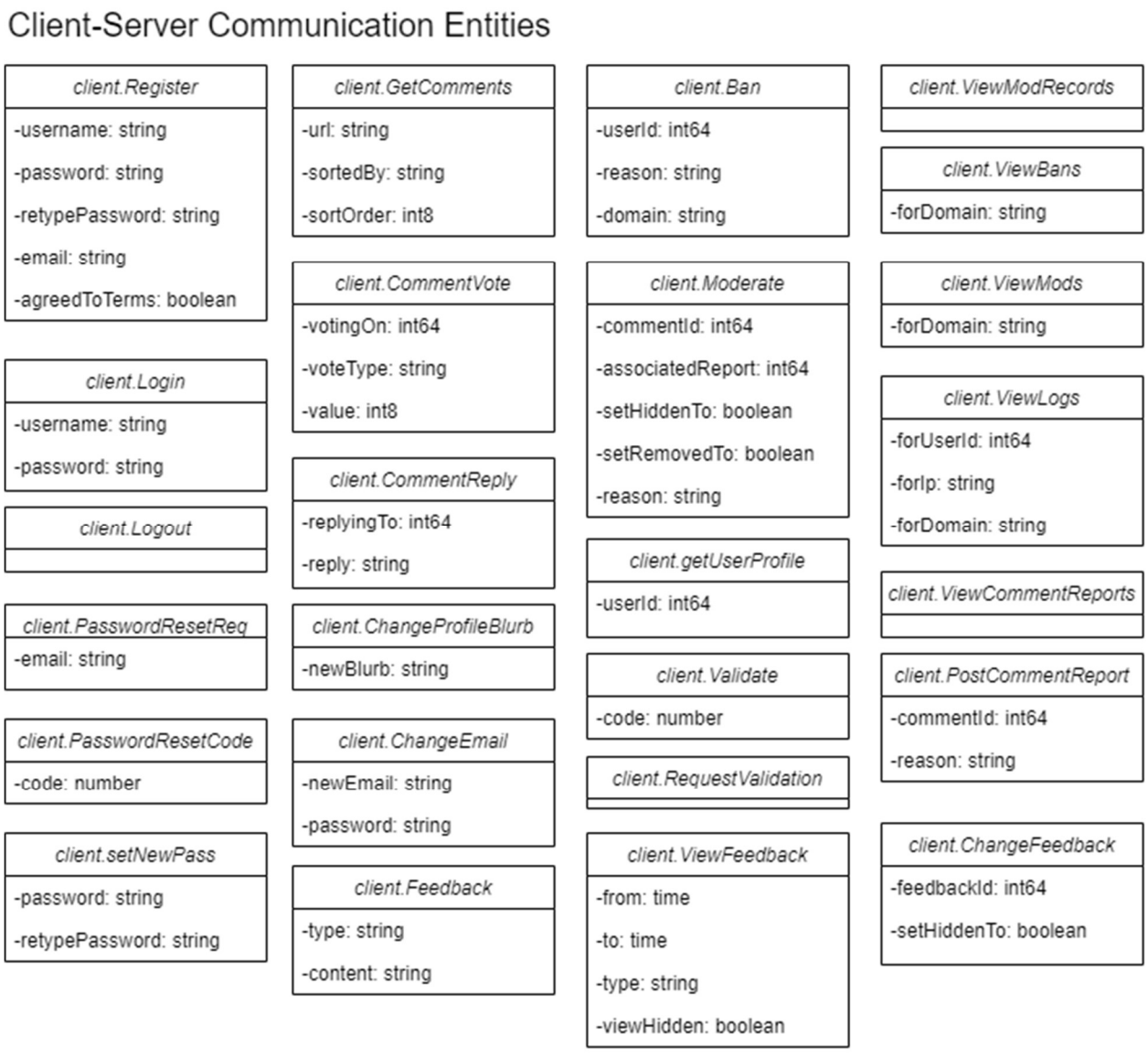
Server.UserProfile

Server.UserProfile contains data needed by the Front End to display a profile for a user.

- **createdOn**: The date that the user's account was created on.
- **domainsModerating**: The server will generate a comma separated list of all domains that the user is responsible for moderating, if applicable. Otherwise, this will be an empty string.
- **isAdmin**: If the user is Admin.
- **isDomainModerator**: If the user is DomainModerator.
- **isGlobalModerator**: If the user is GlobalModerator.
- **profileBlurb**: The profile of the user.
- **userId**: The ID of the user.
- **username**: The name of the user.

Client-Server Communication Entities

Client-Server Communication Entities are entities which are created by the Client to reflect some action that the Client has taken, such as replying to a comment, upvoting a comment, or requesting a report. These entities are sent to the server as JSON strings, and the Server will parse them with its controller classes to realize changes to the database and other Back End data. The Client does not need to provide their username or user ID, because they will be sending a token which the Server will be able to associate with the logged-in user.



client.Ban

client.Ban is dispatched when a moderator or administrator bans a user.

- **domain**: Name of domain the user will be banned from. An empty string will indicate a global ban.
- **reason**: Reason for the ban as provided by the moderator or administrator.
- **userId**: The user ID that will be banned.

client.ChangeEmail

client.changeEmail is dispatched to the server when a client wants to change their email. They must supply the correct password as well.

- **newEmail**: The new email to associate with the client.
- **password**: The user's password.

client.ChangeFeedback

client.ChangeFeedback is dispatched to the Server when an admin wants to remove or hide a Feedback record from being shown to them again.

- **delete**: Whether or not to delete the Feedback.
- **feedId**: A number representing the ID of the Feedback to change.
- **setHiddenTo**: Whether to hide or unhide the Feedback.

client.ChangeProfileBlurb

client.changeProfileBlurb is dispatched to the server when a client updates their profile blurb.

- **newBlurb**: The new text which will be displayed in the user profile.

client.CommentReply

client.CommentReply is dispatched to the server when a logged-in user submits a reply to an existing comment or posts a new root-level comment on a page.

- **reply**: A comment made in a response to another comment.
- **replyingTo**: The id of the original comment the reply leads to.

client.CommentVote

client.CommentVote is dispatched to the server when a logged-in user votes on a comment.

- **value**: 1, if a user is voting up on a rating dimension. -1, if a user is voting down on a rating dimension, or 0, if a User is canceling their previous vote on a rating dimension.
- **voteType**: The different types of ratings a comment. There are currently 3 planned types; “funny” for funny/unfunny comments,“factual” for factual/non factual comments, and “agree” for if the user agrees/disagrees.
- **votingOn**: The ID of the comment the user is rating.

client.Feedback

client.Feedback is dispatched to the Server when a user submits feedback on Comment Anywhere.

- **content**: The text of the feedback, limited to 1,000 characters.
- **type**: Either “Bug”, “Feedback”, or “General”, depending on how the user categorizes their feedback.

client.GetComments

client.getComments is dispatched to the server when a user opens the Browser Extension or when they navigate to a new page with the browser extension over. It is a request for all comments associated with the given url.

- **sortedBy**: Several different methods to sort the comments by, there is “time” for sorting by either latest or oldest time, “funny” for sorting by the most or least funny comments, “factual” for comments that are the most or least factual and “agree” for the most agreed or disagreed comments.
- **sortOrder**: Allows the user to choose if they want to sort the comments in ascending or descending order this allows for sorting by most or least relevant to the topic you're sorting by for example sorting by the oldest or newest comments.
- **url**: The url of the current website the user is using Comment Anywhere on.

client.GetUserProfile

client.getUserProfile is dispatched to the server when the user needs to see a user’s profile.

- **userId**: The user ID of the target user profile.

client.Login

client.Login is dispatched to the server when the client clicks “Submit” on the login form.

- **password**: The password of the desired account.
- **username**: The account name the user desires to login as.

client.Logout

client.Logout is dispatched to the server when the client clicks “Logout”. It does not carry any additional data.

client.Moderate

client.Moderate is dispatched to the server when a moderator or admin takes a moderation action on a comment.

- **associatedReport**: The report associated with this moderation action. 0 if no report.
- **commentId**: The comment Id the moderation action is being taken on.
- **setHiddenTo**: The value to set hidden to.
- **setRemovedTo**: The value to set removed to.
- **Reason**: The reason this moderation action was taken.

client.PasswordResetCode

After clicking “Forgot My Password”, users may enter the code emailed to them. When they subsequently click the “submit” button, this request is dispatched to the server.

- **code:** The code that was sent to the user.

client.PasswordResetReq

client.passwordReset is dispatched to the server when a password reset is requested. The client supplies the email associated with their account.

- **email**: The email associated with the user to reset.

client.PostCommentReport

client.PostCommentReport is dispatched to the server when the user reports a comment.

- **commentId**: The id of the comment being reported.
- **reason**: The reason the user supplied for making the report.

client.Register

client.Register is dispatched to the server when the client clicks “Submit” on the register form.

- **agreedToTerms**: Indicate if the user agreed to the terms of service.
- **email**: Email to be associated with the new account.
- **password**: The user’s password for their new account.
- **retypePassword**: Retyped password for comparison.
- **username**: The user’s chosen name for their new account.

client.RequestValidation

- If a client does not validate their account in a timely fashion, the validation code expires. The client may request a new validation code through their settings tab. When they do so, this entity is created and dispatched to the server.

client.SetNewPass

After submitting a valid password reset code, users are prompted to set a new password. When they subsequently click “submit”, this request is dispatched to the server.

- **password:** The new password for the user.
- **retypePassword:** The password, typed out again.

client.ViewBans

client.viewBans is dispatched to the server when an admin requests records of banned users.

- **forDomain**: Name of the domain. An empty string represents all domains.

client.ViewCommentReports

client.viewCommentReports is dispatched to the server when a moderator requests comment reports. It does not have any data. The server will always respond to this with all reports which have not already been moderated. If the client is a DomainModerator, the server will filter appropriately and does not require additional information from the client.

client.ViewFeedback

client.ViewFeedback is dispatched to the Server when an admin wishes to view feedback submitted by users of Comment Anywhere.

- **from**: A time indicating the lower bound of feedback submission times.
- **to**: A time indicated the upper bound of feedback submission times.
- **type**: Either “Bug”, “Feedback”, “General”, or “All”, depending on which types of feedback the admin is interested in viewing.

client.ViewLogs

client.ViewLogs is dispatched to the server when an admin requests access logs.

- **forDomain**: Name of the domain. An empty string represents all domains.
- **forIp**: Select a specific IP address. An empty string returns all IPs.
- **forUserId**: Select a specific user. Zero returns all users.

client.ViewModRecords

- client.viewModerationRecords is dispatched to the server when an admin requests moderation records. It does not have any data. The server will always respond to this with all moderation records, sorted from newest to oldest.

client.ViewMods

client.ViewMods is dispatched to the server when an admin requests records of who has been assigned as moderators.

- **forDomain**: Name of the domain. An empty string represents all domains.

Front End Local Entities

There are a few local entities used by the Front End which do not require communication with the server.

Settings

Settings contains user settings data which are stored locally.

- **sortBy**: Describes what the user would like to sort by, by default. Either “agree”, “funny”, “factual”, or “date”. This information will be attached to the Client.GetComments request when sent.

- **sortOrder**: Describes the sort order the user would like to view comments in, by default, either “ascending” or “descending”. This information will be attached to the Client.GetComments request.
- **showHidden**: A boolean value indicating whether the user wants to view the text of hidden comments.

State

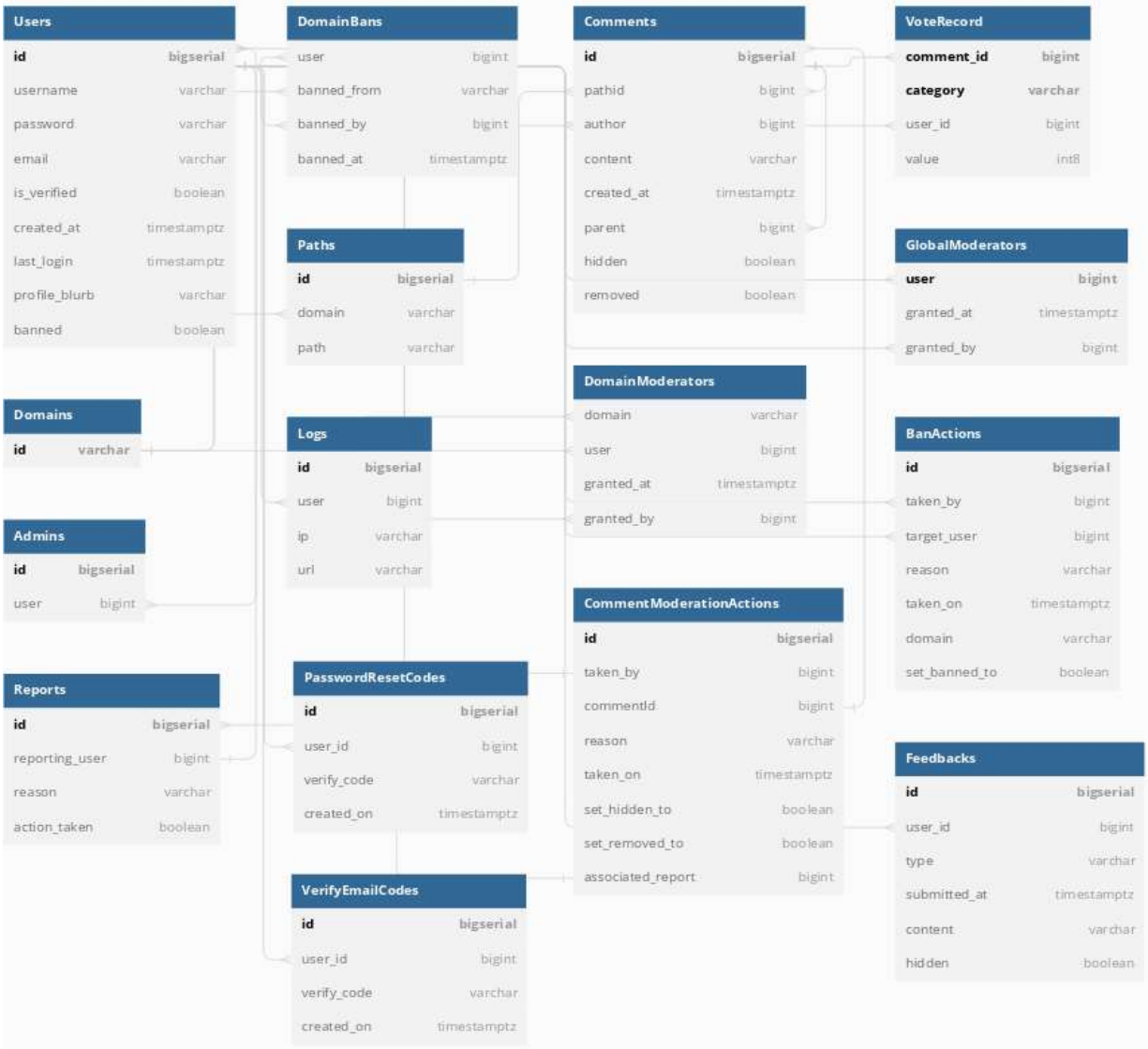
State describes the current state of the Front End. It determines what buttons and windows are viewable by the user in the front end. See the “Front End States” section of this document for a description of available states. The Server can send instructions to the client to put it in another state.

- **currentWindow**: Either “register”, “login”, “forgotpassword”, “comments”, “settings”, “moderation”, or “reports”. This is what window the user is currently viewing.
- **loggedInAs**: Either “none”, “member”, “moderator”, or “admin”. This affects what navigation buttons are visible to the user, and what effects clicking those buttons trigger.

Database Entities - Back End

Our persistent data will be saved in a database, and many of our database tables translate directly into Entities on the Back End. We intend to use the automation to generate source code which will convert tables and queries into objects, such as with sqlc (Conroy). The database

schema is illustrated in the diagram below.



Admins Table, AdminAssignmentRecord Entity

The admins table holds information about Administrators. It is referenced when instantiating a User entity to determine if the User has Administrator Access.

- **id:** Unique instance ID
- **user:** Unique ID to specify the privileged user.

BanActions Table, BanActionRecord Entity

The BanActions table holds information about user bans, either from domains or globally.

- **id**: Unique instance ID.
- **taken_by**: User ID of the moderator who claimed the ban.
- **target_user**: The ID of the user that has been banned.
- **reason**: The reason an admin or moderator banned the user.
- **taken_on**: The day the action was taken on.
- **domain**: If this is a domain ban, this will be a string corresponding to a domain. An empty string indicates a global ban.

set_banned_to: bool If this is an unban, this will set it to false.

CommentModerationActions Table, CommentModerationRecord Entity

Moderator actions taken on reported comments.

- **ID:** The unique action ID.
- **taken_by:** The moderator that claimed to review the comment.
- **comment_id:** The ID of the comment in question.
- **reason:** The moderator's reasoning for taking action.
- **taken_on:** The date the moderator claimed the comment for review.
- **set_hidden_to:** A boolean that states if the moderator flagged the comment as hidden.
- **set_removed_to:** A boolean that states if the moderator removed the comment.
- **associated_report:** The report associated with the moderator's review.

Comments Table, Comment Entity

The Comments Table stores the most important data of Comment Anywhere - the comments. When a user requests to see comments for a particular page, data is gathered by querying this table.

- **id:** A unique number identifying the comment.
- **path_id:** A unique number identifying the location on the internet where the comment is posted.
- **author:** The unique user ID of the user who posted the comment.
- **content:** The text content of the comment.
- **created_at:** A timestamp reflecting the time the comment was posted at.
- **parent:** A unique ID corresponding to the comment this comment is replying to, or 0 if it is a root level comment.
- **hidden:** A boolean value determining whether or not this comment is hidden.
- **removed**: A boolean value determining whether or not this comment is removed.

DomainBans Table, DomainBans Entity

DomainBans are a record of which users are banned from which domains.

- **user:** The unique ID of the User that was banned.
- **banned_from:** The domain from which the user was banned.
- **banned_by:** The user ID of the Moderator who banned this user.
- **banned_at**: A timestamp of when this domain ban occurred.

DomainModerators Table, DomainModerationCreationRecordEntity

The DomainModerators Table holds information about Domain Moderators. One user may be a domain moderator for more than one domain and may have multiple Domain Moderator entries associated with them. The primary key of this table is a two-field key of domain and user.

- **domain:** A string indicating the top-level domain that this moderator authority applies to.
- **user:** The unique user ID of the user to whom this privilege is granted.
- **granted_at:** The timestamp representing when this privilege was granted.
- **granted_by:** The unique user ID of the user who created this privilege.

Domains Table

The Domains table records all Domains that have interacted with Comment Anywhere. They form the base of a path, which corresponds to the unique location on the internet where a comment is posted.

- **domain:** A unique domain of a website. For example, the domain for https://golang.google.cn/doc/tutorial/getting-started would be “golang.google.cn”.

Feedbacks Table, Feedback Entity

The Feedbacks table stores data about feedback submitted by users of Comment Anywhere. Feedbacks have the following properties.

- **id**: A unique ID for this feedback submission.
- **type**: Either “bug”, “general”, or “feature”, depending on how the user categorizes the feedback.
- **user_id**: The unique ID of the user who submitted the feedback.
- **submitted_at**: The time the feedback was submitted.

- **content**: The content of the feedback, limited to 1000 characters.
- **hidden**: Whether the feedback is hidden in reports by default.

GlobalModerators Table, GlobalModeratorCreationRecord Entity

The GlobalModerators Table holds information about Global Moderators. These are assigned by Admins and are allowed to assign DomainModerator status to users. They can take moderation actions on all comments.

- **user:** The user granted this privilege. This is also the primary key of the table, as a user cannot be assigned GlobalModerator privileges more than once.
- **granted_at:** The timestamp when this privilege was granted.
- **granted_by:** The user ID of the Admin who granted this privilege.

Logs Table, Log Entity

Save logs regarding our users and services. This information can be viewed by administrators.

- **id:** Unique ID of the log.
- **user:** Recorded user ID.
- **ip:** The user's recorded IP.
- **url:** The url the user is currently viewing.

Password Reset Codes Table

The Password Reset Codes Table contains all active validation codes that allows the user to reset their password. This code will be active for 10 minutes upon user request.

- **ID:** Unique instance ID.

- **user_id:** The user who possesses the code.
- **verify_code:** A string containing the verification code.
- **created_on:** The time the code was created. Will only be active for 10 minutes.

Paths Table, Path Entity

Paths are the unique location on the Internet where a comment may be posted. They are associated with a domain from the Domains table and some sub-path strings.

- **id:** The unique id of this path.
- **domain:** The unique domain that corresponds to the base of the URL.
- **path:** The path taken from that base URL. For example, the path for https://golang.google.cn/doc/tutorial/getting-started would be "doc/tutorial/getting-started"

Reports Table, CommentReport Entity

Reports contain all reported comments across all domains.

- **ID:** The unique ID of a reported Comment.
- **comment_id:** ID of the target comment.
- **reporting_user:** The User ID that reported the comment.
- **reason:** The reason, provided by the User, for reporting the comment.
- **action_taken:** A boolean which indicates Moderator review.

Users Table, User Entity

The Users table stores data about users of Comment Anywhere. Users have the following properties.

- **banned**: Whether or not the user is globally banned.
- **id:** A unique identification number for each user.
- **created_at:** A timestamp reflecting when the user was created.
- **email:** The user's email, for performing password resets.
- **is_verified:** Indicates if the user's provided email has been verified.
- **last_login:** A timestamp recording the last time the user logged in.
- **password:** The user's password, stored safely as a hashed and salted value.
- **profile_blurb**: An optional text a user may supply describing themselves to other users.
- **username:** A unique name for the user which will be displayed to other users and is used to log in.

Verify Email Codes Table

The Verify Email Codes Table contains all active validation codes that allows the user to verify their email. This code will be active for 10 minutes upon user request.

- **ID:** Unique instance ID.
- **user_id:** The user who possesses the code.
- **verify_code:** A string containing the verification code.
- **created_on:** The time the code was created. Will only be active for 10 minutes.

VoteRecord Table, VoteRecord Entity

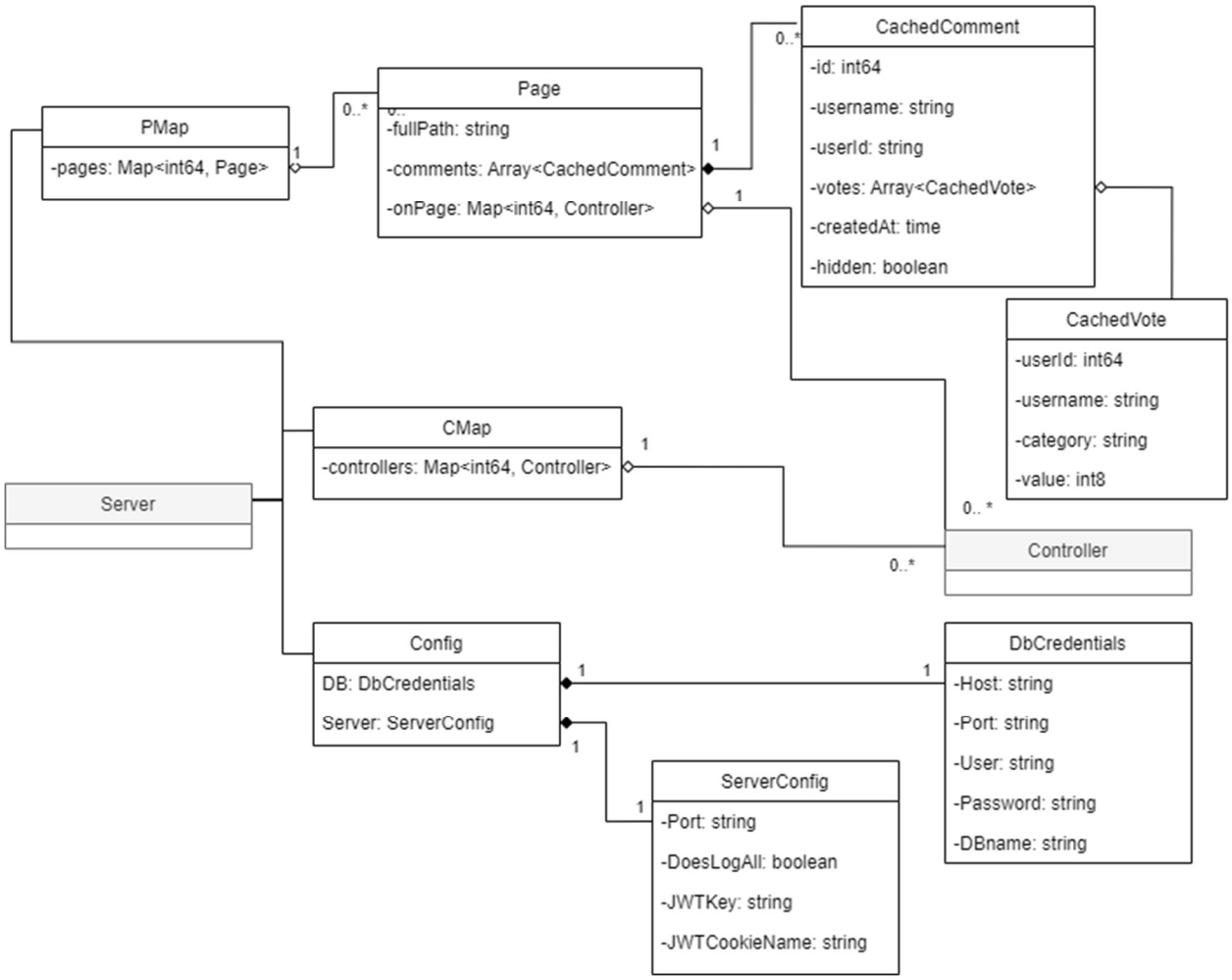
The VoteRecord Table stores upvotes and downvotes users have submitted. It has a composite primary key, one that is a foreign key referencing the associated comment, and one that describes the category of vote.

- **comment_id**: A unique number corresponding to a comment's id.
- **category**: "funny", "factual", or "agree", depending on what category this vote is for.
- **user_id**: The user ID of the user who submitted this vote.
- **value**: Whether this is an "up" (1) or a "down" (-1).

Caching Entities - Back End

Some data will be cached in the server to prevent excessive database querying. When a user visits a page, all the comments on that page are loaded into the CachedPage instance, for example. The instance is removed when no users are viewing that page anymore. This diagram references Controller and Server, both of which are Control classes that will be described in the Design document.

Caching Entities, Back End



CachedComment

CachedComment contains data on a comment.

- **userId**: The ID of the user who posted the comment.
- **parent**: The ID of the parent comment.
- **username**: The name of the user who posted the comment.
- **votes**: An array of CachedVote data, representing all votes that have been made on the comment.
- **createdAt**: The time when this comment was posted.

- **hidden**: Whether the comment is set to hidden or not.

CachedVote

CachedVote contains cached data about a comment vote.

- **userId**: The ID of the user who cast the vote.
- **username**: The name of the user who cast the vote.
- **category**: The vote category, e.g., “funny”, “factual”, “agree”
- **value**: How the user voted; -1 if it was a downvote and 1 if it was an upvote.

CMap

CMap contains a map of all controllers, which represent logged-in users. Controllers will be described in the design section.

- **controllers**: A map of all active Controllers. The key is the userID, and the value is a Controller instance.

Config

Config contains settings for the server and database.

- **DB**: A dbCredentials instance
- **Server**: A ServerConfig instance.

DbCredentials

DbCredentials contains information the Server needs to communicate with the database.

- **Host**: The location of the database. If run locally, this would be “localhost”.
- **Port**: The port where the database instance is running. For example, “5432”.
- **User**: A username required to access the database.

- **Password**: A password required to access the database.
- **DBname**: The name of the database.

Page

Page contains data retrieved from the database.

- **fullPath**: The full path for this page, with both the domain and subsequent path.
- **comments**: An array of CachedComments containing comment data for comments posted on this page.
- **onPage**: A map of all controllers, representing users, currently visiting this page. The ID is the userID of the Controller.

PMap

PMap contains a map of all currently loaded Pages. The key is the ID of the associated path.

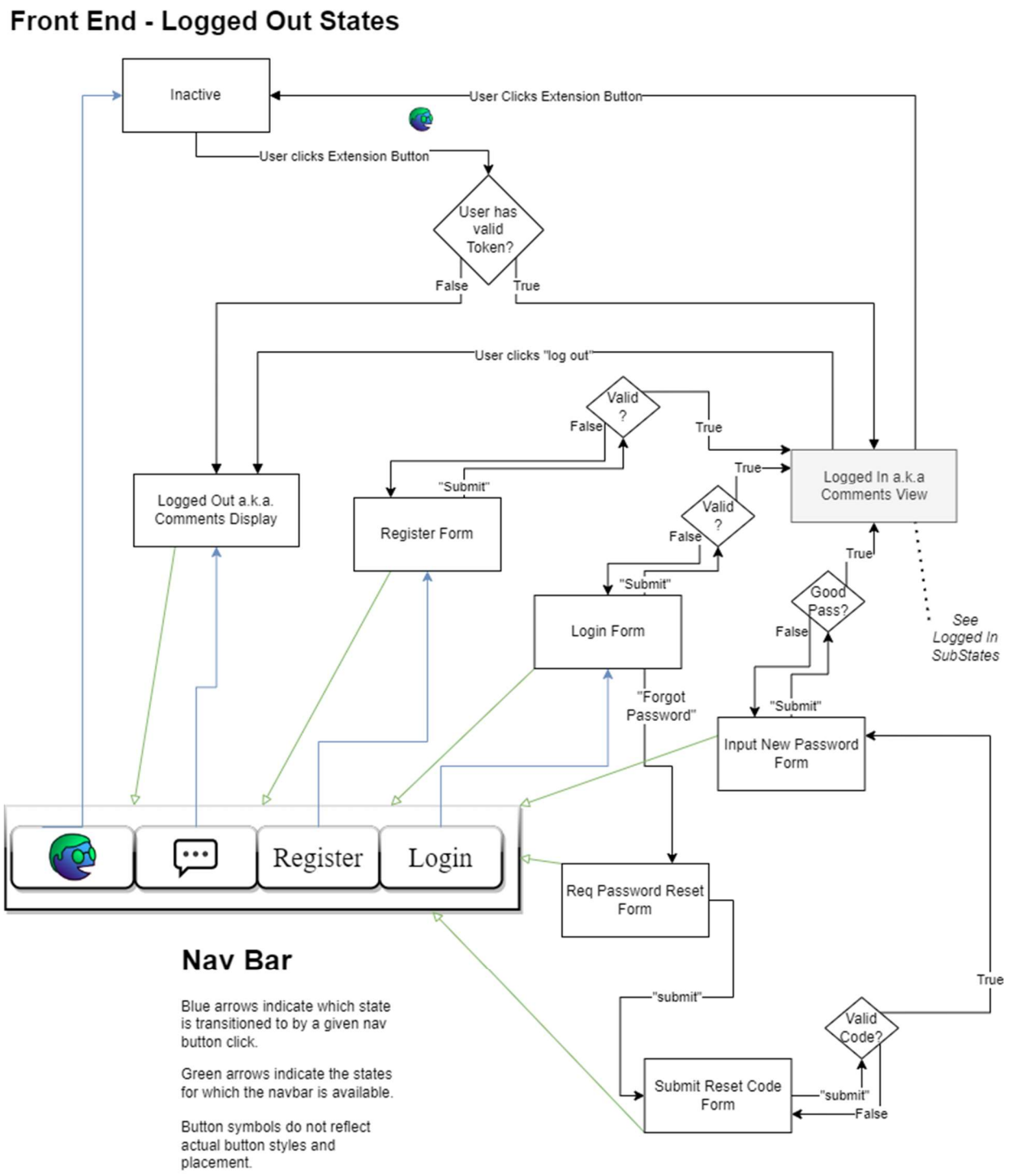
- **pages**: A map of all instanced pages, with the key as the associated path.

ServerConfig

ServerConfig contains settings the Server uses for miscellaneous tasks.

- **Port**: The port where the server is listening. E.g. “8080”
- **DoesLogAll**: A boolean value indicating whether the server logs all data it receives to the console, for debugging purposes.
- **JWTKey**: A string used to encode and decode tokens supplied to users.
- **JWTCookieName**: The name of the cookie that will be written to HTTP Responses and read from HTTP Requests to determine if a user is logged in and, if so, who they are.

Dynamic Statechart



Individual users of Comment Anywhere transition through states as they perform actions on the User Interface. The current state is stored in the local entity *State* and realized through changes to what UI Elements are displayed and the creation and population of boundary classes with Server data. Because there are many redundant state transitions, these diagrams have been simplified by including a Nav Bar symbol collection. These symbols describe buttons that cause state transitions which are available to a user in a given state.

Inactive

Inactive is the default state of the Front End. No code is running, but a button exists on the browser which the user can click. When this button is clicked, the Server will check the User's cookies to see if they have a valid token. If they do, it will respond with data to put the User in an Extension Active: Logged In state. Otherwise, it will respond with data which will cause the Front End to transition to the Extension Active: Logged Out state.

Navbar (non-state)

The Navbar widgets are the same for each Logged Out state. Clicking the extension button will set the state to Inactive. Clicking the Comment button will set the state to Logged Out a.k.a. Comments Display. Clicking the Register button will set the state to Register Form. Clicking the Login button will set the state to Login Form.

Logged Out

In the Logged Out state, Comments for the current page are displayed in a sidebar popout on the side of the browser's viewport. Users can see comments for the page but cannot reply or report to them.

Register Form

In this State, the User is presented with a Registration form. If they submit a valid form, the Server will create the User in the database and log them in, responding with data which will cause the Front End to transition to a Logged In State. The Server will provide a token in the form of a cookie or similar for the User to store client-side, so they don't need to repeat logging in.

Login Form

In this State, the User is presented with a Login form. If they submit a valid form, the Server will log the User in and respond with data which will cause the Front End to transition to the Logged In State. The Server will also provide a token in the form of a cookie or similar for the User to store client-side, so they don't need to repeat logging in. If the User cannot recall their password, they may click the "Reset Password" button to transition to the Req Password Reset Form state.

Req Password Reset Form

In this State, the User is presented with a Password Reset form. They can enter their username to have a password reset code emailed to them. Upon submission of this form, the Front End will transition back to the Submit Reset Code Form state.

Submit Reset Code Form

In this state, the User is presented with a form for inputting the code that was emailed to them. Upon submitting a valid code, they will transition to the Input New Password Form state.

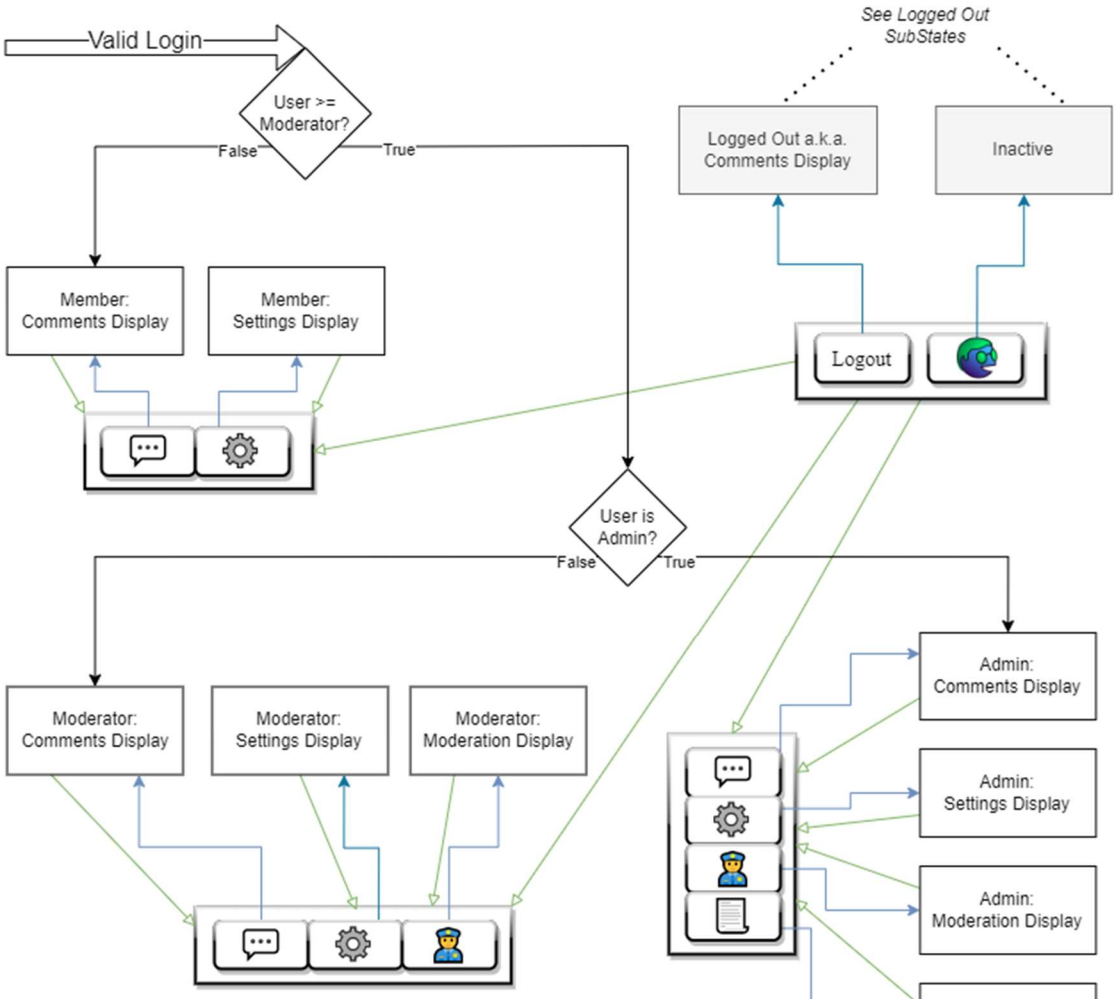
Input New Password Form

In this state, the User is required to submit a new valid password. Upon doing so, the Server will log the User in and respond with data causing a transition to a Logged In State.

Login State

In this state, the User is logged in. They will start in a Comments Display substate according to their access level. From all substates, “Logout" is available to transition them to the Extension Active: Logged Out state, and they can always click the Extension button to bring them to the Extension Inactive state.

Logged In SubStates



Nav Bars

Blue arrows indicate which state is transitioned to by a given nav button click.

Green arrows indicate the states for which the navbar is available, and indicate NavBar inheritance.

Button symbols do not reflect actual button styles and placement.

Member: Comments Display

In this substate, comments for the current page are visible. The member may click “reply” or “new comment” to add a new comment. They may vote on and report comments. These actions do not change the state. The member may click the “Settings” button to transition to the Member Settings state.

Member: Settings Display

In this substate, a member’s settings are visible. There may be settings to change the appearance of the program. There will be a setting to allow Members to view comments which have been moderated as offensive and are hidden by default to Guests and Members who do not select this option. There will be a button to allow them to change their password, after entering their current password, and change their email, after entering their current password. They may click the Comments button to transition to the Member: Comments Display state.

Moderator: Comments Display

This substate is similar to the Member: Comments Display substate, but moderators have access to an additional button for each comment, allowing them to take moderation actions on comments. They also have an additional navigation button, “Moderation”, which will transition them to the Moderator: Moderation substate.

Moderator: Moderation Display

This substate provides the moderator with a list of reports within their domain, if they are a Domain Moderator, or globally, if they are a Global moderator. They can view the reports, reported comments, and take moderation actions in response to them as needed.

Moderator: Settings Display

This substate is like the Member: Settings Display substate, but it may include some additional moderator specific settings.

Admin: Comments Display

This substate is very similar to the Moderator: Comments Display substate, but the Admin has an additional navigation button available in all their states for navigating to the Admin: Reports substate.

Admin: Moderation Display

This substate is similar to the Moderator: Moderation substate but there may be some additional features only available to admins here.

Admin: Settings Display

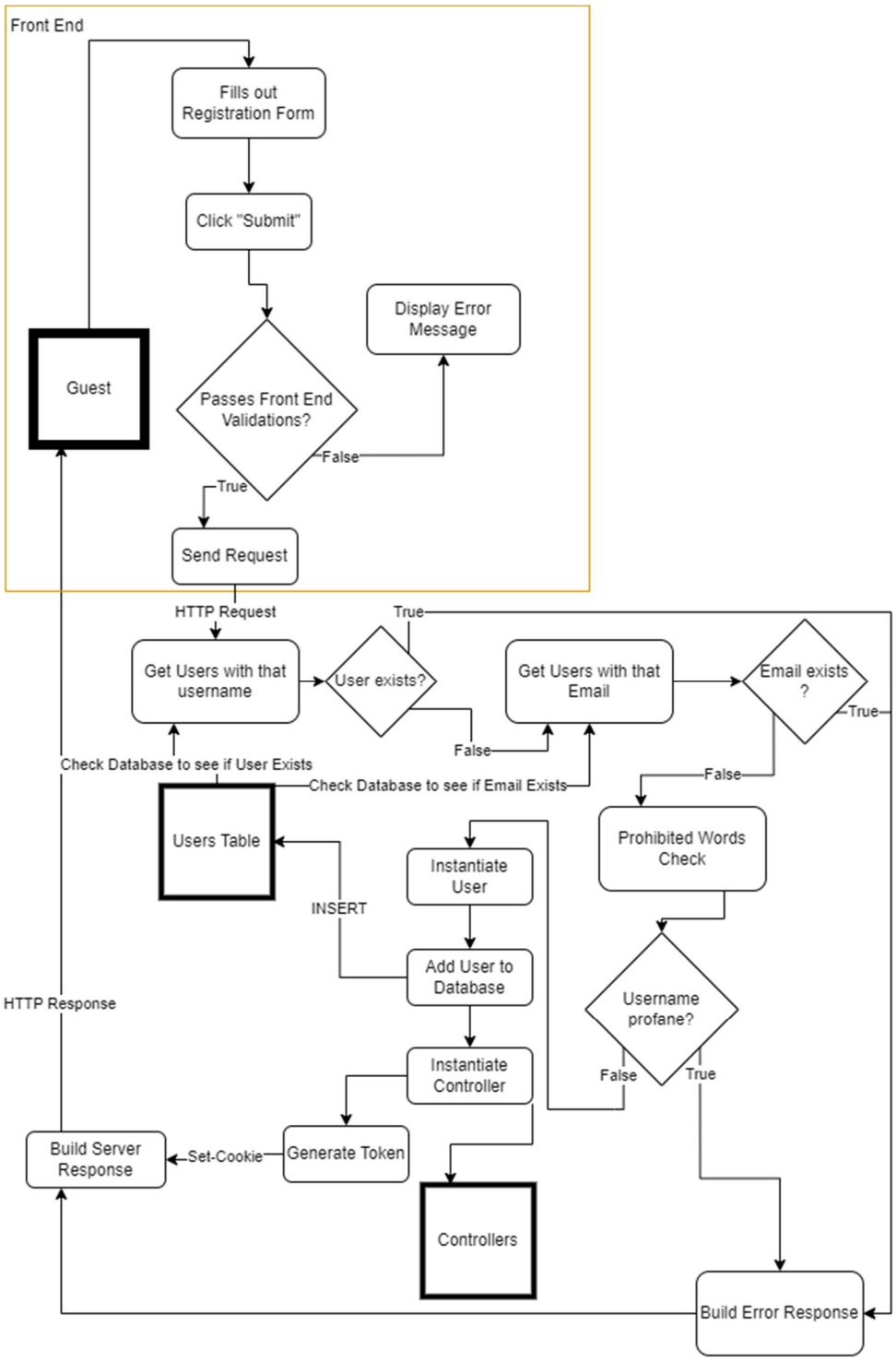
This substate is similar to the Moderator: Settings substate, but there may be additional admin features, including some server settings.

Admin: Reports Display

This substate allows Admins to view reports. They can see recent comments that have been posted, the number of users logged in, the total number of users, and access logs. They may have additional reports available to be determined at a later time.

Dataflow Diagrams

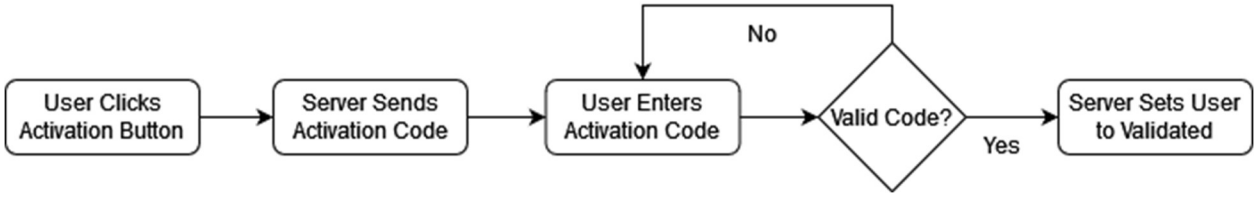
Registration Dataflow



This Dataflow maps how a Registration request is processed.

1. A Guest makes a registration request, after clicking the “Register” button, filling out the form, and clicking “Submit”.
2. The Server receives the registration request. It queries the Users table of the Database to see if a user with that username already exists. If a user already exists, the Server will respond with an error. If a user with the supplied email already exists, the Server will respond with an error.
3. The Server checks the password to determine whether it is strong enough. It may compare the password to rainbow tables or simply check the length. If it is insufficiently strong, the Server will respond with an error.
4. The Server performs a check to ensure that the Username is unique and conforms to rules and sends a response to the Front End.
5. If all these validation checks have passed, the Server will tell the database to insert the new record for the user. It will log the user in and generate a unique token for the session.
6. Finally, the Server will build the response string, consisting of JSON Data which the Front End needs, including the token, which will likely be transmitted as a cookie header. It will dispatch that response to the client, which will modify the Front End state accordingly.

Activation Dataflow



This Dataflow maps how a User Account becomes verified and active after Registration.

1. A User sends an activation request to the backend. This is done after a User has Registered an account and is done through the Settings menu in the Extension.
2. The backend sends mail to the email address attached to the User's Account containing a unique activation code, which is only usable for a set time period.
3. The User enters the activation code into the field now showing in the Extension.
4. The extension sends the code to the server, whereupon it is verified by the server.
5. Upon valid activation code, the server sets the User Account to verified and sends a confirmation mail to the attached email.

Components and Tools Needed

Development Requirements

Visual Studio Code

Visual Studio Code (https://code.visualstudio.com/) is a streamlined code editor with support for development operations like debugging, task running, and version control. Visual Studio Code utilizes third-party extensions to enhance development; this product requires the usage of the following extensions: Go, Typescript, Live Share.

Git

Git (https://git-scm.com/) is a distributed version control system designed to handle projects with speed and efficiency. Git is used to facilitate version control of product files.

Docker

Docker (https://www.docker.com/) is a platform as a service product that uses Virtualization to enable developers to quickly deliver products by utilizing containerization. Docker is used to deploy and test the application's components.

NodeJS

NodeJS (https://nodejs.org/en/) is a JavaScript runtime that enables developers to build products utilizing the entire ecosystem of JavaScript libraries, using an asynchronous event-driven architecture. Software dependencies are managed by the Node Package Manager (https://www.npmjs.com/). This product relies on the following packages: Typescript, Webpack. NodeJS is used to develop the Front End software for the product.

Go

Go (https://go.dev/) is a programming language developed by Google designed to support scalable, concurrent software that is fast, reliable, and efficient. Go is used to develop the backend server software.

SQLC

SQLC (https://sqlc.dev/) generates type-safe code from SQL queries.

Discord

Discord (https://discord.com/) is a social application and the chosen tool to facilitate discussions and collaboration between team members.

Front End Extension Requirements

Web Browser

A supported web browser will be required to run and test the Front End. The supported browsers initially are the following:

- Google Chrome (https://www.google.com/chrome/)
- Mozilla Firefox (https://www.mozilla.org/en-US/firefox/new/)

Back End Server Requirements

Docker

Docker (https://www.docker.com/) is a platform as a service product that uses Virtualization to enable developers to quickly deliver products by utilizing containerization. Docker is used to deploy and test the application's components.

GNU Make

GNU Make (https://www.gnu.org/software/make/) is a tool which controls the generation of executables and other non-source files of a program from the program's source files. This is used in combination with Docker for deployment and testing of the project.

PostgreSQL

PostgreSQL (https://www.postgresql.org/) is a relational database system. PostgreSQL will be the primary means of storage for the product’s data.

Appendix: Glossary

Administrator

An Administrator is a User with the highest administration privileges. Administrators can access reports, ban users, remove comments, and delegate those authorities by assigning Domain and Global Moderators.

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. (Christensson, 2020)

Browser

A program that accesses and displays information from the Internet.

Browser Extension

A program which has the Operating Environment of a Browser. It can be installed by users of that Browser to add functionality to the Browser.

Cloud Service Provider

Cloud Service providers are vendors which provide Information Technology (IT) as a service over the Internet.

Database

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a Database Management System, or DBMS. Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.Data within the most common types of databases in operation today is typically modeled in rows and columns in a series of tables to make processing and data querying efficient. The data can then be easily accessed, managed, modified, updated, controlled, and organized. Most databases use structured query language (SQL) for writing and querying data. (Oracle Corporation, n.d.)

Docker 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.

Data Flow Diagram

Data Flow diagrams graphically represent the processes which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. (“What is Data Flow Diagram?”)

Dynamic State Chart

State Charts model the dynamic behavior of an object (with multiple states of behavior) by showing the possible states that the object can be in.

Entity Class Diagrams

Entity Class Diagrams model the entities within the system..

Entity Classes

An entity is a long-lived, passive element that is responsible for some meaningful chunk of information. Entities perform behavior organized around some cohesive amount of data. (Armstrong Process Group)

Front End

The Front End of a software program is everything with which the user interacts. From a user standpoint, Front End is synonymous with the User Interface. The Front End of Comment Anywhere is the Browser Extension. (Christensson, 2020)

Graphical User Interface

A type of user interface which allows a user to interact with an application in ways other than text, for example, with menus and buttons.

Guest

A user of Comment Anywhere who is not logged in. They are capable of viewing some comments for a URL but not posting.

HTTP Request

An HTTP request is made by a client, to a named host, which is located on a server. The aim of the request is to access a resource on the server. (IBM)

HTTP Response

An HTTP response is made by a server to a client. The aim of the response is to provide the client with the resource it requested or inform the client that the action it requested has been carried out; or else to inform the client that an error occurred in processing its request. (IBM)

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). (MDN Contributors)

Localization

Localization refers to the process of conforming software and the digital user experience to the language and cultural norms of an end user in any geographic region. (Kent State University)

Member

A Member of Comment Anywhere is any User that has registered an account.

Moderator

A User who has been granted permission to take Moderation Actions. In the context of Comment Anywhere, Moderators may be Domain Moderators or Global Moderators.

Namespace

A Namespace is a set of signs or names that are used to identify and refer to objects of various kinds. A namespace ensures that all of a given set of objects have unique names so that they can be easily identified.

Register

The process of creating an account by providing necessary information, such as a username, email address, and password.

SQL

Structured Query Language, or SQL, is a programming language for storing and processing information in a relational database. (Amazon Web Services, Inc.)

SQL Injection

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. (PortSwigger Ltd.)

Server

A program which waits for a request then performs some service for the requester and which runs on a computer other than the one on which the requestor/client runs. (Raymond, n.d.)

User

Anyone who uses Comment Anywhere through a Front End. They may be a Guest, Member, Domain Moderator, Global Moderator, or Admin.

User Activity

Activity performed by a user of Comment Anywhere such as viewing comments, posting comments, and taking moderation actions.

Use Case Scenario

A Use Case Scenario is an example of how a User interacts with the product.

Website

A group of World Wide Web pages usually containing hyperlinks to each other and made available online by an individual, company, educational institution, government, or organization.

Web Service

A Web Service is an application running on a networked device servicing incoming requests and serving web documents.

Waterfall Model

The linear life cycle model with feedback loops. (Schach)

Appendix: Team Details

I, Karl Miller, attest that I executed the functions listed within the workflow authentication section of the document. I agree with all information stated within the Specifications document.

Karl Miller	Karl L. Miller	11/14/2022
Printed Name	Signature	Date

I, Frank Bedekovich, attest that I executed the functions listed within the workflow authentication section of the document. I agree with all information stated within the Specifications document.

Frank Bedekovich	Frank J Bedekovich	11/14/2022
Printed Name	Signature	Date

I, Robert Krency, attest that I executed the functions listed within the workflow authentication section of the document. I agree with all information stated within the Specifications document.

Robert Krency	Robert Krency	11/14/2022
Printed Name	Signature	Date

I, Luke Bates, attest that I executed the functions listed within the workflow authentication section of the document. I agree with all information stated within the Specifications document.

Luke Bates	Luke Bates	11/14/2022
Printed Name	Signature	Date

Appendix: Writing Center Report

Client: Comment Anywhere Team

Staff or Resource: Caedon Vogel

Date: 11/09/2022 12:00 pm - 1:00 pm

Did the student request that the instructor receive a visit report? **Yes.**

What course was serviced by this visit? **ACSC-490 Senior Project**

What goals were established for this tutoring session? **Reviewing the document, commenting on things the tutor did not understand, and fixing grammatical errors as well as format.**

How did the process of this consulting session address the established goals? **The tutor reviewed the document and went over revisions and comments with the clients.**

Please provide any additional comments relevant to this session?

Format Comments

- Not much needed to be changed! I removed some extra spaces you had inserted between titles and paragraphs and such. With double spacing, you generally don't need to add more space because things are already so spaced out.
- The format is consistent and looks good. I can tell what sections are what based on the headings and the paragraph format.

Grammar Comments

- There were quite literally zero issues that I found. Well done!

Others

- There were multiple areas in which I did not understand what was being said, but that is most likely due to my computer engineering incompetence! I would just add a brief explanation for those things in parentheses right next to them, especially because this

document should be understandable by most. If you can't really do that, no worries, it might just be my small brain making things difficult for me to understand.

- Make sure you finish the glossary! For people like me who have limited knowledge on the subject, it will be helpful!

Appendix: Workflow Authentication

Karl Miller created the database schema and prototype. He created, contributed to, or re-wrote the Abstract, User Interaction, Cost Constraints, Hardware Constraints, and Software Constraints sections. He created the Dynamic State Charts and Registration Data Flow Diagram. He went through several iterations to develop the necessary Entities and their diagrams.

Frank Bedekovich wrote several sections of the Analysis document including the purpose and use, intended audience, time constraints, cost constraints, acceptance test criteria, criteria for user acceptance, integration of separate parts and installation, and system modeling. Frank also wrote the drafts for the other constraints section, as well as the figures and run scenarios in the system modeling section. He later went on to fill out and cite sections of the glossary and revise and edit throughout the document.

Luke Bates helped describe and adjust the database schema. Helped complete Client-Server Communication Entities. He updated the use case diagrams. He wrote the Criteria for User Acceptance.

Robert Krency developed the Activation Dataflow and diagram, contributed to the glossary and references, and handled gathering information on required components and tools.

All group members were present for proofreading and editing suggestions.

References

Amazon Web Services, Inc. “What is SQL? - SQL.” *AWS*, https://aws.amazon.com/what-is/sql/. Accessed 13 November 2022.

Armstrong Process Group. “Guideline: Entity-Control-Boundary Pattern.”

Avi. “10 Best Docker Hosting Platforms for your Containers.” *Geekflare*, 22 September 2022, https://geekflare.com/docker-hosting-platforms/. Accessed 28 October 2022.

“Cloud Run: Container to production in seconds.” *Google Cloud*, Alphabet, Inc., https://cloud.google.com/run#section-13. Accessed 6 November 2022.

Conroy, Kyle. “sqlc Documentation.” *sqlc Documentation — sqlc 1.15.0 documentation*, 2021, https://docs.sqlc.dev/en/latest/index.html. Accessed 28 October 2022.

HostGator. “Cloud Hosting Plans - Secure & Scalable Services.” *HostGator*, https://www.hostgator.com/cloud-hosting. Accessed 6 November 2022.

IBM. “HTTP requests.” *HTTP requests - IBM Documentation*, IBM, 13 10 2021, https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests. Accessed 13 November 2022.

IBM. “HTTP responses.” *HTTP responses - IBM Documentation*, 18 December 2020, https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-responses. Accessed 13 November 2022.

Kent State University. “What Is Software Localization? | Computer Software Localization.” *Kent State*, https://www.kent.edu/appling/matranslationonline/blog/software-localization. Accessed 13 November 2022.

Luenendonk, Martin. “Cloud Hosting Cost Comparison In 2022.” *FounderJar*, 28 September 2022, https://www.founderjar.com/cloud-hosting-costs/. Accessed 6 November 2022.

MDN Contributors. “Working with JSON - Learn web development | MDN.” *MDN Web Docs*, 24 September 2022, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON. Accessed 13 November 2022.

PortSwigger Ltd. “What is SQL Injection? Tutorial & Examples | Web Security Academy.” *PortSwigger*, https://portswigger.net/web-security/sql-injection. Accessed 13 November 2022.

Schach, Stephen R. *Object-Oriented and Classical Software Engineering*. Edited by Weifeng Chen. 8th ed., 2011. *Chapter 2 - Fall 2022 Senior Project 1*, WCB/McGraw-Hill, https://pennwest.brightspace.com/d2l/le/content/3470075/viewContent/34182812/View. Accessed 13 11 2022.

“What is Data Flow Diagram?” *Visual Paradigm*, https://www.visual-paradigm.com/guide/data-flow-diagram/what-is-data-flow-diagram/. Accessed 13 November 2022.