Comment Anywhere

Pennwest California

CSC 490: Senior Project 1

Design Document

Dr. Chen

<DATE>

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 and Evaluation

TO DO LIST

* For todo items, see the Trello board: <https://trello.com/b/cULajlBV/design-document>
  + Feel free to add more cards to the board for other work you do
  + You can double click a card to add yourself to the card
  + Move something from “to-do” to “doing” when you are working on it
  + Move something to “review” when you are done working on it

# Table of Contents

[**Instructor Comments and Evaluation**](#_xbkix2sy7ya7) **3**

[**Table of Contents**](#_c4zhs0i6c92k) **4**

[**Abstract**](#_kbk9tzzii0om) **6**

[**Description of Document**](#_kwyvaym0rwfe) **7**

[**Project Block Diagram**](#_clf8f9tz7wt) **8**

[Figure X](#_ns1wdoy9bgus) 8

[**Design Details**](#_at6bmxgghdcg) **9**

[System Modules and Responsibilities](#_8kjoj65zxz34) 9

[Module Cohesion](#_ukryedijluqq) 10

[Module Coupling](#_jm57xmgs48ex) 11

[Design Analysis](#_tj72z1qldsp2) 12

[Design Organization](#_1k0r6hlnk926) 14

[Description of Classes](#_ua5caa8t9tkl) 14

[Package util](#_18n55e110z53) 14

[Package util classes](#_mwhqct5g60wv) 14

[Package server](#_hoi9sfwbejcp) 15

[Controller Interface Pattern Diagram](#_bkrfpxpd7bav) 16

[Package server Classes](#_rj76xqv993d6) 19

[Example, Server Code](#_h5b83x6cm7k2) 25

[Package database and database.generated](#_sn5eani0vyzl) 28

[Database Schema Diagram](#_glzj8b5wripm) 29

[Description of Package database Classes](#_lqbfzl55qcke) 29

[Example, SQLC Generated Code](#_dhtxs62h90tj) 42

[Package communication](#_co9t1ocqut8d) 44

[Client-Server Communication Entities](#_uisfarpctei2) 44

[Server-Client Communication Entities](#_i7unjmwyhsz8) 50

[Front End](#_rhw9qolz7dwz) 53

[Front End Architecture Diagram](#_r4cdjzidmqqk) 54

[Description of Front End Classes](#_xo5xakf5yv4f) 55

[Functional Descriptions](#_o29c8xylq2b9) 71

[Package util](#_grdcaonb4g9y) 71

[Package server](#_9uaud1myetag) 72

[Package database](#_yxvmhc3xpe5g) 84

[Front End](#_66eaqvx9ti5i) 94

[Narrative/PDL](#_qyae7sju5y0a) 95

[Implementation Tools](#_m2kopikyuhuj) 99

[Implementation Timeline](#_cb0fhcylp0q) 99

[Testing](#_uxfgx4p9hjou) 101

[**Appendix**](#_ed6zo1hifewf) **102**

[Appendix A: Team Details](#_x0ywrxecj0sv) 102

[Appendix B: Workflow Authentication](#_bovmkuttulgb) 103

[Appendix C: Writing Center Report](#_3ebbvepb2umf) 104

# Abstract

This document is a description of the design specifications for the Comment Anywhere software product, hereafter referred to generally as Comment Anywhere. The product allows Users to bring discussions to any web content through a Browser Extension. This document is for the developers of the software product. This document is provided to lay out information about the development process and manage guidelines and requirements for development. Architectural details provide a basis for understanding the product modules, the inter-module communication, and the tools of development.

# Description of Document

The Design serves to describe the Architecture and design of the software product, Comment Anywhere. This document lays out the Client-Server architecture and the modules that comprise each. The modules are then planned out with accompanying action methods, data types, and messaging protocols defined in the planned implementation languages. This document is to be used by the team to guide development.

The Design document synthesizes the information from the Specification document. This lays out an architecture for the product, categorizes the products required features into modules, and defines the modules with respect to the intended implementations.

This document is written with the development team as an intended audience. The team is expected to use this to guide development and limit feature creep or unintended additions. The Design document is intended to provide the developer with an understanding of the product architecture and how feature interaction happens.

# Project Block Diagram

## Figure X

# Design Details

## System Modules and Responsibilities

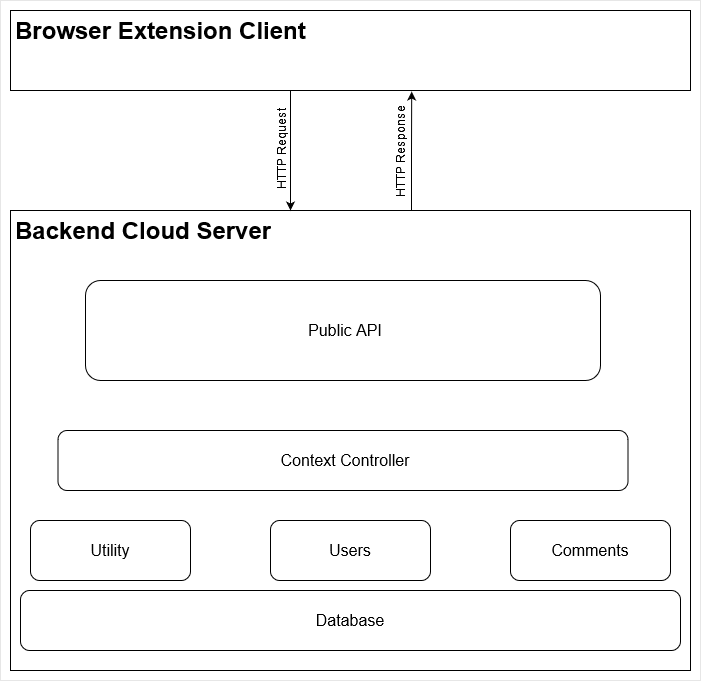


Figure 2: Architectural Diagram

Comment Anywhere is bifurcated into a Client-Server architecture, as depicted in the Architectural Diagram (Figure 2 above). The client is the Browser Extension, a platform specific graphical user interface that runs within a web browser. The client and server communicate through an Application Programmer Interface (API) using HTTP requests and responses to submit and retrieve comments, to register, login, and change user data, and access moderation functionality.

The server consists of several modules: the API, Context Controllers, Users, Comments, Utilities.

The API Module is responsible for accepting HTTP Requests at each exposed endpoint. It ensures valid incoming requests, and delegates to a Context Controller.

The Context Controllers interface with the User Manager to determine User Roles and Permissions and perform User Account related actions. Controllers also interface with the Comment Manager to retrieve, submit, or change content for the specific Context.

The Users module consists of User data retrieval and User account management, and directly interfaces with the Database for storage. Account management includes actions such as creating accounts and updating account information.

The Comments module handles the lifecycle of content, primarily the Comments that make up discussions on Comment Anywhere.

The Utilities module consists of utility functionality such as logging and data conversion or composition.

### Module Cohesion

Modular Cohesion is achieved through restricting module functionality to a category of Data that each Module is responsible for. For example, the Users module is the sole manager of all User related data and management, such as saving and loading User data from the Database. This is mirrored in the design of the Comments module as well.

### Module Coupling

Modules are designed to be as loosely coupled as possible. Context Controllers are the main source of inter-module interaction. This helps enforce that separate modules have no direct interaction. Each HTTP Request is assigned a new or extant Context Controller. Each API endpoint calls an appropriate method on the assigned Context Controller. The Controller then interfaces with the other modules depending on the action generated, such as retrieving User information and permissions from the User module or passing a designated filter to the Comments module to retrieve specific comments.

## 

## Design Analysis

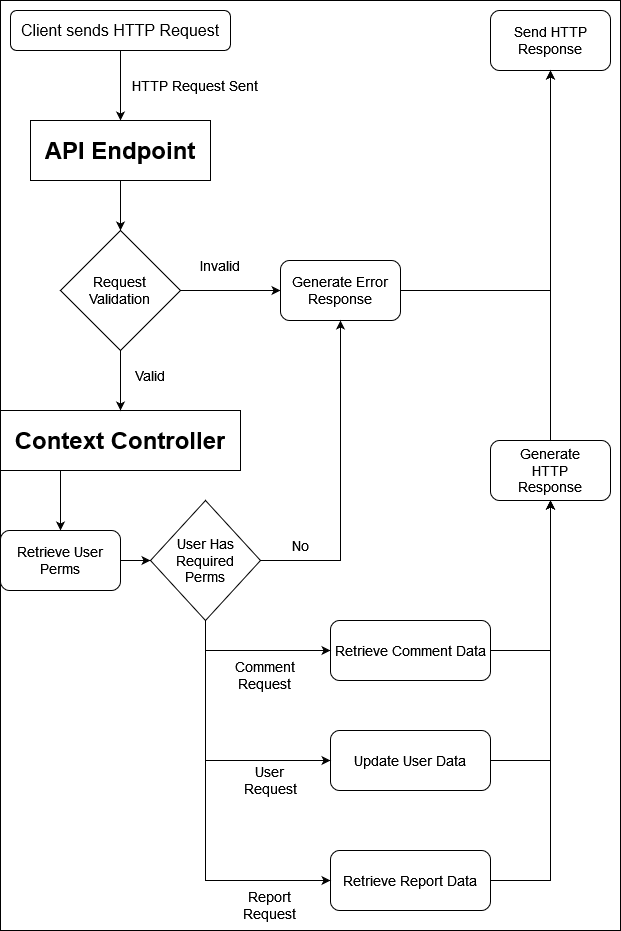


Figure 3: Data Flow Diagram

The Data Flow Diagram (Figure 3 above) shows a high-level view of the lifecycle of an HTTP Request sent from a Client to the Server. An incoming request is validated to be a properly formed request before the appropriate API Endpoint generates a Context Controller for the request. The Context Controller then retrieves User Permissions to verify if the User is able to make the request. Upon success, the Controller performs the request action based on the request type, and sends a successful payload response back to the client in the form of an HTTP Response. Failed request responses will be delivered with appropriate error messaging.

## Design Organization

### Description of Classes

#### Package util

The util package is primarily responsible for reading and validating an .env file to set up the server and database configuration. If future changes to the design require that stand-alone functions be written that are available to multiple parts of the back end, those functions will also be included in the util package.

*Note*: The capitalization of classes is not arbitrary. Go exports members which are capitalized, while members which are not are only available in that scope.

##### Package util classes

**dbCredentials**

**Description**: dbCredentials holds information the Server needs to connect to the Database, such as the Port, Password, and Database name.

**Data Members**: string Host, string Port, string User, string Password, string DBName

**Class Methods**: ConnectString(): string, loadDBEnv()

**serverConfig**

**Description**: serverConfig holds information the Server needs to run, such as the port it runs at and the cookie name that will be used.

**Data Members**: string Port, bool DoesLogAll, string JWTKey, string JWTCookieName

**Class Methods:** loadServerEnv()

**config**

**Description**: config holds both the Server and Database configurations and loads them from the .env file. It is exported as the singleton Config (with a capital C, for Go exports) and accessible around the program.

**Data Members**: dbCredentials DB, serverConfig Server

**Class Methods**: Load(string)

#### Package server

The Server package contains classes for performing the majority of the business logic on the back end, such as authentication and responding appropriately to an HTTP Request at a particular endpoint.

##### Controller Interface Pattern Diagram

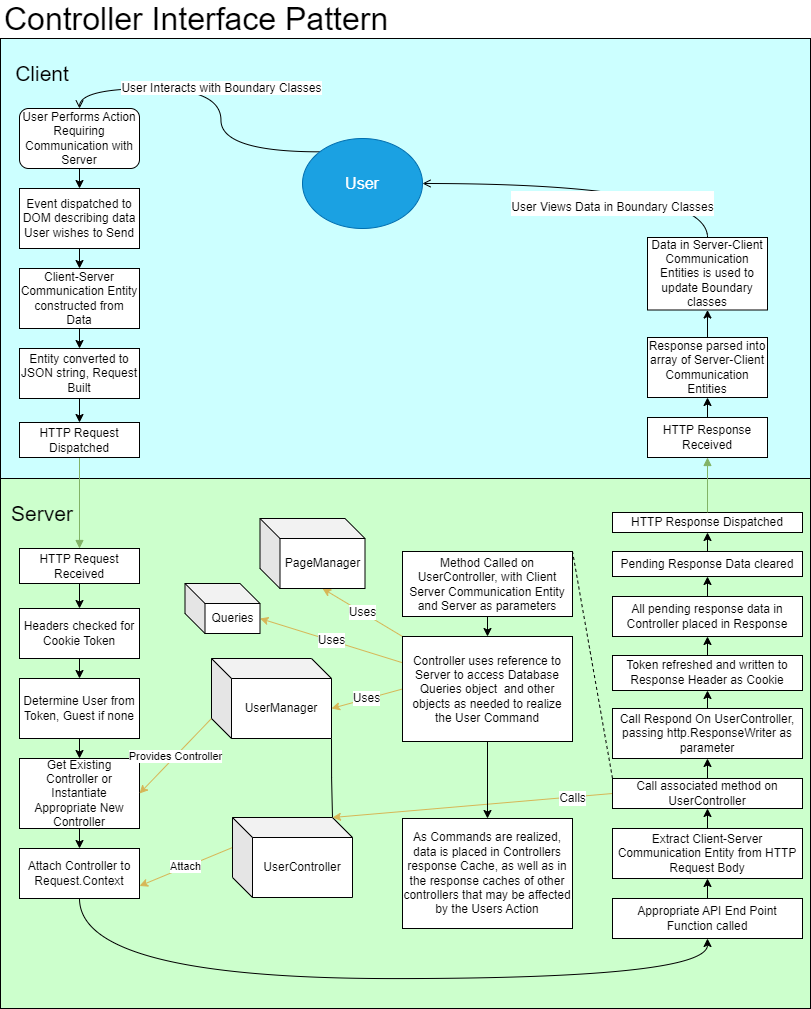


Figure 4: Controller Interface Pattern

The Dataflow Diagram (Figure 4 above), illustrates how user commands are associated with a particular user and realized. When a User performs some action on the Front End that requires server action, a Client-Server Communication Entity is constructed and an HTTP Request is dispatched to the appropriate API endpoint. Upon receiving the request, the Server first checks the Request Headers for a cookie containing an encrypted JWT Token and parses that to determine who, if anyone, the user is. If the token is valid and not-expired, UserManager retrieves the already-instantiated ControllerInterface for that User, or instantiates one appropriate to the User’s access level. Otherwise, it instantiates a GuestController. A reference to this controller is then attached to the HTTP Request Context and routing to the appropriate API endpoint function occurs.

At the API Endpoint function, Server parses the body of the HTTP Request to extract the Client-Server Communication Entity generated by the Front End. It then calls the associated method on the ControllerInterface that was attached to the HTTP Request, passing to that method the extracted communication entity as well as a reference to the Server so that the ControllerInterface may access other objects as needed to realize the command. As the ControllerInterface realizes the User’s command, it populates its own response data and may result in the population of response data for other Users. For example, when a User posts a new comment on a page, all ControllerInterfaces located on that page get information about the new comment. After the ControllerInterface method has been called, Server directs ControllerInterface to create a new authentication token and add the Cookie to the header. Finally, Server calls the Respond method of the ControllerInterface which causes it to populate an HTTP Response body with its saved data before clearing its saved data. It then dispatches the Response back to the client.

When the client receives the HTTP Response, it parses the body of the response to extract data sent by the Server. It uses this data to update and add boundary objects and display to the user the data they requested.

##### Package server Classes

**UserControllerInterface**

**Description**: UserControllerInterface provides method signatures which other UserController types implement. Controller references are attached to HTTP Request Contexts in the first middleware that a Request passes through. Those controller references are subsequently used by API endpoints to execute access-appropriate code associated with a particular user or guest. At the API endpoints, the Server is “blind”, and will tell whatever controller is attached to the Request to deal with the command extracted from the Request body, which necessitates the interface polymorphism. UserControllerInterface is also used to track which pages are currently being viewed by users, via maps on Pages.

**Data Members**: None; interface does not have data members.

**Class Methods**: HandleCommandBan(\*communication.Ban, \*Server), HandleCommandChangeEmail(\*communication.ChangeEmail, \*Server), HandeCommandChangeFeedback(\*communication.ChangeFeedback), HandleCommandChangePassword(\*communication.SetNewPass, \*Server),

HandleCommandChangeProfileBlurb(\*communication.ChangeProfileBlurb, \*Server), HandleCommandCommentReply(\*communication.CommentReply, \*Server), HandleCommandCommentVote(\*communication.CommentVote, \*Server), HandleCommandFeedback(\*communication.Feedback, \*Server), HandleCommandGetComments(\*communication.GetComments, \*Server), HandleCommandGetUserProfile(\*communication.GetUserProfile, \*Server), HandleCommandLogin(\*Server), HandleCommandLogout(\*Server), HandleCommandModerate(\*communication.Moderate, \*Server), HandleCommandPasswordResetCode(\*communication.PasswordResetCode, \*Server),

HandleCommandPasswordResetRequest(\*communication.PasswordResetRequest, \*Server), HandleCommandCommentReport(\*communication.PostCommentReport, \*Server), HandleCommandRequestVerification(\*communication.RequestVerification, \*Server), HandleCommandVerify(\*communication.Verify, \*Server), HandleCommandViewBans(\*communication.ViewBans, \*Server), HandleCommandViewCommentReports(\*communication.ViewCommentReports, \*Server), HandleCommandViewLogs(\*communication.ViewLogs, \*Server), HandleCommandViewModRecords(\*communication.ViewModRecords, \*Server), HandleCommandViewMods(\*communication.ViewMods, \*Server), Respond(r http.Request, w http.ResponseWriter), GetCurrentPage() :\*Page, dispatchResponse(r http.Request, w http.ResponseWriter)

**UserControllerBase**

**Class Description**: UserControllerBase provides data members for UserControllers. It does not implement UserControllerInterface fully. Other controllers are defined by extending this Base class and implementing the rest of the interface. Controllers also retain an array of messages that need to be sent to the client, which will be dispatched the next time a request from that user is received.

**Class Data Members**: generated.User User, time.Time lastTokenRefresh, \*Page OnPage, [][]byte nextResponse

**Class Methods**: dispatchResponse(http.Request, http.ResponseWriter), GetCurrentPage() \*Page

**MemberControllerBase**

**Class Description**: UserControllerBase provides data members for MemberControllers. It extends UserControllerBase, adding some fields necessary for validation and password reset tracking.

**Class Data Members**: boolean canResetPassword, *extends UserControllerBase*

**Class Methods**: *extends UserControllerBase*

**GuestController**

**Class Description**: This Controller is attached to an HTTP Request Context when a non-logged in user accesses Comment Anywhere.

**Class Data Members**: *extends UserControllerBase*

**Class Methods**: *Implements UserControllerInterface, extends UserControllerBase*

**MemberController**

**Class Description**: This Controller is attached to an HTTP Request Context when a regular member accesses Comment Anywhere.

**Class Data Members**: *extends MemberControllerBase*

**Class Methods**: *Implements UserControllerInterface, extends MemberControllerBase*

**DomainModeratorController**

**Class Description**: This Controller is attached to an HTTP Request Context when a domain moderator accesses Comment Anywhere.

**Class Data Members**: []string DomainsModerated, *extends MemberControllerBase*

**Class Methods**: *Implements UserControllerInterface, extends MemberControllerBase*

**GlobalModeratorController**

**Class Description**: This Controller is attached to an HTTP Request Context when a global moderator accesses Comment Anywhere.

**Class Data Members**: *extends MemberControllerBase*

**Class Methods**: *Implements UserControllerInterface, extends MemberControllerBase*

**AdminController**

**Class Description**: This Controller is attached to an HTTP Request Context when an administrator accesses Comment Anywhere.

**Class Data Members**: *extends MemberControllerBase*

**Class Methods**: *Implements UserControllerInterface, extends MemberControllerBase*

**UserManager**

**Class Description**: UserManager maintains a map of all instantiated controllers for logged-in users and a map of all instantiated controllers for guests. UserManager is responsible for retrieving controllers associated with a userID or temporary guest ID, either by instantiating a new controller, querying the database if necessary, or by supplying an existing controller if one has already been instantiated for that ID.

**Class Data Members**: map[int64]UserControllerInterface members, map[int64]UserControllerInterface guests

**Class Methods**: Ban(\*communication.Ban, \*Server server), Login(\*communication.Login, \*UserControllerInterface, \*Server server): \*UserControllerInterface, Logout(\*UserControllerInterface, \*Server server) : \*GuestControllerInterface, Register(\*UserControllerInterface, \*Server server), GetMemberController(int64 id): \*UserControllerInterface, GetGuestController(int64 id): \*UserControllerInterface, DispatchPasswordResetEmail(\*UserControllerInterface, \*Server server)

**PageManager**

**Class Description**: PageManager maintains a map of all instantiated Pages that are currently being viewed by some user or guest. It is responsible for ‘placing’ and ‘removing’ users from pages.

**Class Data Members**: map[string]Page

**Class Methods**: MoveMemberToPage(\*UserControllerInterface user, string pagePath, \*Server server), MoveGuestToPage(\*UserControllerInterface user, string pagePath, \*Server server), UnloadEmptyPages(\*Server server), loadPage(string path, \*Server server)

**Page**

**Class Description**: Page contains cached data for a page, which is a discrete set of comments associated with a particular URL. It also contains a map of all users and guests on the current page.

**Class Data Members**: string fullPath, map[int64]CachedComment comments, map[int64]UserControllerInterface membersOnPage, map[int64]UserControllerInterface guestsOnPage

**Class Methods**: GetComments(\*UserInterface, string sortedBy, bool ascending): []communication.Comment, addMemberToPage(\*UserControllerInterface user), removeMemberFromPage(\*UserControllerInterface user), addGuestToPage(\*UserControllerInterface user), removeGuestFromPage(\*UserControllerInterface user), Moderate(\*communication.Moderate, \*Server), CreateComment(\*communication.CommentReply, \*Server server), VoteComment(\*communication.CommentVote, \*Server server)

**CachedComment**

**Class Description**: CachedComment contains data for a single comment which has been loaded from the database.

**Class Data Members**: int64 id, string content, int64 userID, int64 parent, string username, []CachedVote votes, int64 createdAt, bool hidden, bool removed

**Class Methods**: Vote(\*communication.CommentVote, \*Server server), dataForUser(int64 userId) : communication.Comment

**CachedVote**

**Class Description**: CachedComment contains data for a single comment vote which has been loaded from the database.

**Class Data Members**: int64 userId, string username, string, category, int8 value

**Server**

**Class Description**: Server holds references to core data structures, such as UserManager, PageManager, database.Store, and Router. It has a method for each API end point. At each end point, it extracts the communication entity the User sent and calls the command handler on the Controller which has been attached the HTTP Request with the extracted data. It generally passes a reference to itself to Controller method calls so that the Controller can access components such as the database and page manager. It is, essentially, the “highway” of the Back End.

**Class Data Members**: \*mux.Router router, database.Store DB, ControllerManager ControllerManager, PageManager PageManager

**Class Methods**: New(): \*Server, setupRouter(), Start(), MiddlewareAttachController(handler http.Handler): http.Handler, postAssignDomainModerator(\*http.Request, http.ResponseWriter), postAssignGlobalModerator(\*http.Request, http.ResponseWriter), postBan(\*http.Request, http.ResponseWriter), postChangeEmail(\*http.Request, http.ResponseWriter), postChangeFeedback(\*http.Request, http.ResponseWriter), postChangeProfileBlurb(\*http.Request, http.ResponseWriter), postCommentReply(\*http.Request, http.ResponseWriter), postCommentVote(http.Request, http.ResponseWriter), postFeedback(\*http.Request, http.ResponseWriter), GetComments(\*http.Request, http.ResponseWriter), getUserProfile(\*http.Request, http.ResponseWriter), postLogin(\*http.Request, http.ResponseWriter), postLogout(\*http.Request, http.ResponseWriter), postModerate(http.Request, http.ResponseWriter), postPasswordResetCode(\*http.Request, http.ResponseWriter), postPasswordResetRequest(\*http.Request, http.ResponseWriter), postCommentReport(\*http.Request, http.ResponseWriter), postRegister(\*http.Request, http.ResponseWriter), postRequestVerification(\*http.Request, http.ResponseWriter), putSetNewPass(\*http.Request, http.ResponseWriter), postVerify(\*http.Request, http.ResponseWriter), getBans(\*http.Request, http.ResponseWriter), getCommentReports(\*http.Request, http.ResponseWriter), getDomainReport(\*http.Request, http.ResponseWriter), getUsersReport(\*http.Request, http.ResponseWriter), getFeedback(\*http.Request, http.ResponseWriter), getLogs(\*http.Request, http.ResponseWriter), getModRecords(\*http.Request, http.ResponseWriter), getMods(\*http.Request, http.ResponseWriter), getLoginStatus(\*http.Request, http.ResponseWriter)

##### Example, Server Code

// src/server/postCommentReport.go

package server

import (

"context"

"database/sql"

"encoding/json"

"net/http"

"github.com/comment-anything/prototype1/communication"

"github.com/comment-anything/prototype1/database/generated"

)

// API Endpoint for https://commentanywhere.net/newReport

func (server \*Server) postCommentReport(request \*http.Request, writer http.ResponseWriter) {

// instantiate a new empty report

report := communication.PostCommentReport{}

// attempt to read the body of the request to the report

err := json.NewDecoder(request.Body).Decode(&report)

if err != nil {

writer.WriteHeader(http.StatusBadRequest)

} else {

controller := getControllerInterfaceFromContext(request.Context())

controller.HandleCommandCommentReport(&report, server)

controller.Respond(request, writer)

}

}

// What occurs when a Guest attempts to report a comment.

func (c \*GuestController) HandleCommandCommentReport(msg \*communication.PostCommentReport, server \*Server) {

// create an error message for transmission to the client

message := communication.Message{

Success: false, Text: "You must be logged in to report a comment.",

}

// convert that message into a packet for front-end parsing

bytes, err := communication.CreatePacket(message, communication.ServerMessage)

if err != nil {

// append the message to the responses the client is waiting on

\_ = append(c.nextResponse, bytes)

}

}

// What occurs when a logged-in user attempts to report a comment; a record is inserted into the database.

func (c \*MemberController) HandleCommandCommentReport(msg \*communication.PostCommentReport, server \*Server) {

// create the comment report in the database

server.DB.Queries.CreateCommentReport(context.Background(), generated.CreateCommentReportParams{

ReportingUser: c.User.ID,

Comment: msg.CommentId,

Reason: sql.NullString{String: msg.Reason},

})

// create a response message

message := communication.Message{

Success: true, Text: "Comment Report submitted.",

}

bytes, err := communication.CreatePacket(message, communication.ServerMessage)

if err != nil {

// append the message to the responses the client is waiting on.

\_ = append(c.nextResponse, bytes)

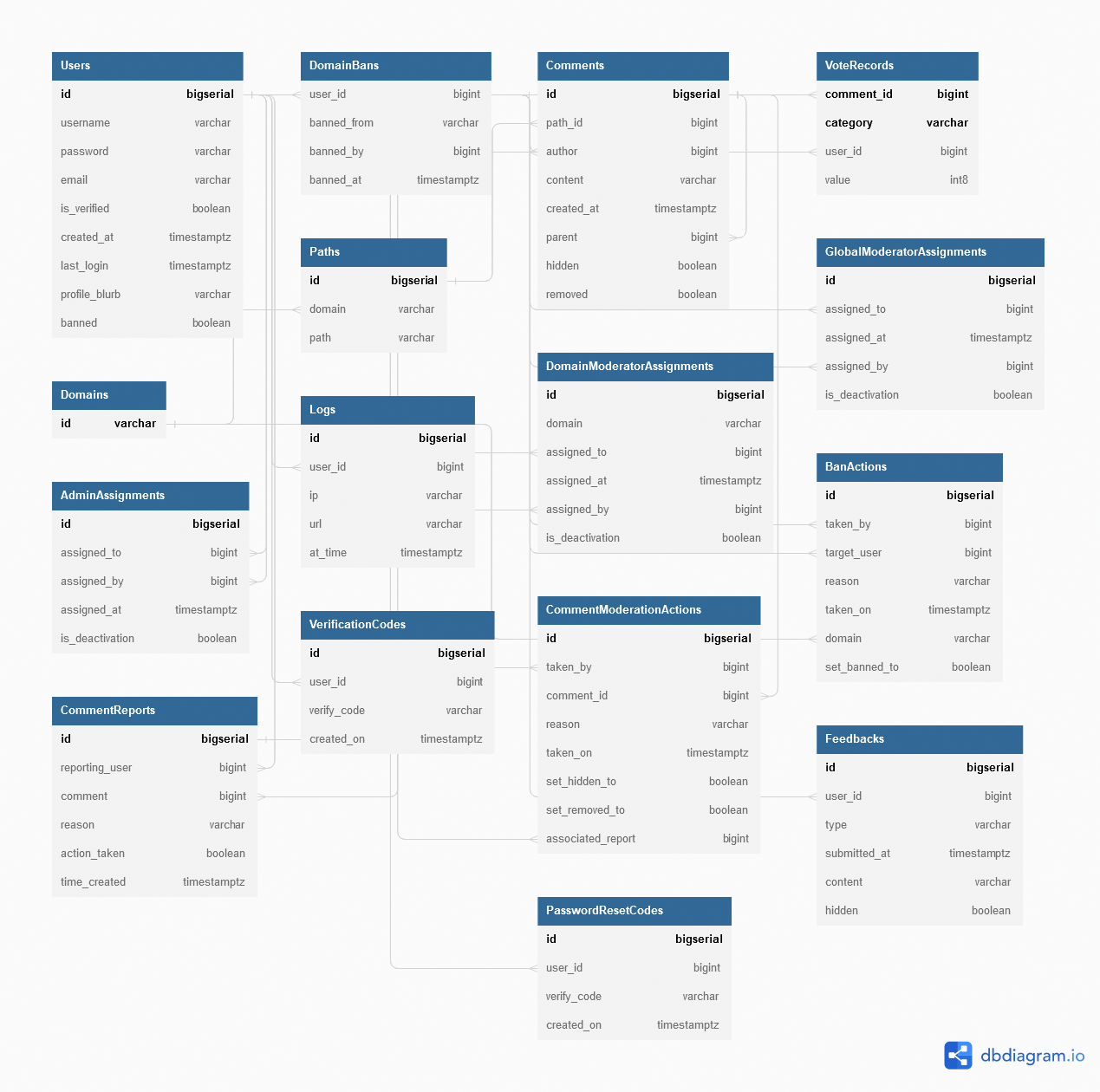
}

}

#### **Package database and database.generated**

Package database contains classes the Server needs for interfacing with the database. It provides an access point between the server and the generated packages. The database.generated package contains code generated by sqlc which is created from the schema of the database and queries for that database.

##### Database Schema Diagram



##### Description of Package database Classes

**Store**

**Class Description**: Store handles connecting with the database and provides an instance of Queries for utilizing the sqlc generated code for parameterized queries. It is the only class in this section which is not in the database.generated package. Its primary purpose is to wrap the Queries object and connect to the Postgres instance.

**Class Data Members**: \*sql.DB DB, \*generated.Queries Queries

**Class Methods**: New(): Store, Connect(), Disconnect()

**AdminAssignment**

**Class Description**: A model of an entry in the AdminAssignments table.

**Class Data Members**: int64 ID, int64 AssignedTo, int64 AssignedBy, bool isDeactivation

**BanAction**

**Class Description**: A model of an entry in the BanActions table.

**Class Data Members**: int64 ID, int64 TakenBy, int64 TargetUser, string Reason, int64 TakenOn, string Domain, bool SetBannedTo

**Comment**

**Class Description**: A model of an entry in the Comments table.

**Class Data Members**: int64 ID, int64 PathId, string Author, string Content, time.Time CreatedAt, int64 Parent, bool Hidden, bool Removed

**CommentModerationAction**

**Class Description**: A model of an entry in the CommentModerationActions table.

**Class Data Members**: int64 ID, int64 TakenBy, int64 CommentId, string Reason, time.Time TakenOn, bool SetHiddenTo, bool SetRemovedTo, int64 AssociatedReport

**CommentReport**

**Class Description**: A model of an entry in the Comments table.

**Class Data Members**: int64 ID, int64 ReportingUser, int64 Comment, string Reason, bool ActionTaken, time.Time TimeCreated

**Domain**

**Class Description**: A model of an entry in the Domains table.

**Class Data Members**: string ID

**DomainBan**

**Class Description**: A model of an entry in the DomainBans table.

**Class Data Members:** int64 UserID, string BannedFrom, int64 BannedBy, time.Time BannedAt

**DomainModeratorAssignment**

**Class Description**: A model of an entry in the DomainModeratorAssignments table.

**Class Data Members:** int64 ID, string Domain, int64 AssignedTo, int64 AssingedBy, time.Time AssignedAt, boolean isDeactivation

**Feedback**

**Class Description**: A model of an entry in the DomainBans table.

**Class Data Members:** int64 ID, int64 UserID, string Type, time.Time SubmittedAt, string Content, boolean Hidden

**GlobalModeratorAssignment**

**Class Description**: A model of an entry in the GlobalModeratorAssignments table.

**Class Data Members:** int64 ID, int64 AssignedTo, time.Time GrantedAt, int64 AssignedBy, boolean IsDeactivation

**Log**

**Class Description**: A model of an entry in the Logs table.

**Class Data Members:** int64 ID, int64 User, string IP, string URL

**PasswordResetCode**

**Class Description**: A model of an entry in the PasswordResetCodes table.

**Class Data Members:** int64 ID, int64 UserID, string VerifyCode, time.Time CreatedOn

**Path**

**Class Description**: A model of an entry in the Paths table.

**Class Data Members:** int64 ID, string Domain, string Path

**Report**

**Class Description**: A model of an entry in the Reports table.

**Class Data Members:** int64 ID, int64 ReportingUser, int64 Comment, string Reason, bool ActionTaken

**User**

**Class Description**: A model of an entry in the Users table.

**Class Data Members:** int64 ID, string Username, string Password, string Email, time.Time CreatedAt, time.Time LastLogin, string ProfileBlurb, boolean Banned, boolean isVeirified

**ValidationCode**

**Class Description**: A model of an entry in the ValidationCodes table.

**Class Data Members:** int64 ID, int64 UserID, string VerifyCode, time.Time CreatedOn

**VoteRecord**

**Class Description**: A model of an entry in the VoteRecords table.

**Class Data Members:** int64 CommentId, string Category, int64 UserId, int8 Value, int64 CommentId, string Category, int64 UserId, int8 Value

**Queries**

**Class Description**: A generated struct which provides methods and parameter structs for every database query used by Comment Anywhere.

**Class Methods:** CreateDomainModeratorAssignment(context.Context, CreateDomainModeratorAssignmentParams), CreateGlobalModeratorAssignment(context Context, CreateGlobalModeratorAssignmentParams), CreateDomainBanRecord(context Context, CreateDomainBanRecordParams), UpdateUserBanStatus(ctx context.Context, UpdateUserBanStatusParams), UpdateUserEmail(ctx context.Context, UpdateUserEmailParams), UpdateFeedbackHidden(context.Context, UpdateFeedbackHiddenParams), UpdateUserBlurb(context.Context, UpdateUserBlurbParams), CreateComment(context.Context, CreateCommentParams), CreateCommentVote(context.Context, CreateCommentVoteParams), UpdateCommentVote(context.Context, UpdateCommentVoteParams), DeleteCommentVote(context.Context, DeleteCommentVoteParams), CreateFeedback(context.Context, CreateFeedbackParams), GetCommentsAtPath(context.Context, int64 pathID): []GetCommentsAtPathRow, GetCommentVotes(context.Context, int64 commentID): []GetCommentVotesRow, GetUserByUserID(context.Context, int64 id): User,

CreateModerationRecord(context.Context, CreateModerationRecordParams), UpdateCommentHidden(context.Context, UpdateCommentHiddenParams), UpdateCommentRemove(context.Context, UpdateCommentRemoveParams), CreateLog(context.Context, CreateLogParams), GetDomainModeratorAssignments(context.Context, int64 id), GetGlobalModeratorAssignments(context.Context, int64 id), GetAdminAssignments(context.Context, int64 id), GetUserByUsername(context.Context, string username) : User, UpdateUserLastLogin(context.Context, int64 id), CreateCommentReport(context.Context, CreateCommentReportParams), CreateUser(context.Context, CreateUserParams), DeleteUser(context.Context, int64 id), GetUserByUsername(context.Context, string username): User, GetUserByEmail(context.Context, string email): User, UpdateUserPassword(context.Context, UpdateUserPasswordParams), UpdateUserVerification(context.Context, UpdateUserVerificationParams), CreateVerificationRecord(context.Context, CreateVerificationRecordParams) GetVerificationRecord(context.Context, int64 userID): []VerificationCode, DeleteVerificationRecords(context.Context, int64 userID), GetPWResetRecord(context.Context, int64 userID): []PasswordResetCode, CreatePWResetRecord(context.Context, CreatePWResetRecordParams), DeletePWResetRecords(context.Context, int64 userID), GetBanRecords(context.Context): []GetBanRecordsRow, GetCommentReports(context.Context, Boolean actionTaken): []GetCommentReportsRow, GetFeedback(context.Context, Boolean hidden) []GetFeedbackRow, GetLogsForDateRange(context.Context, GetLogsForDateRangeParams): []GetLogsForDateRangeRow, GetModRecordsForModerator(context.Context, int64 id): []GetModRecordsForModeratorRow, GetDomainModerators(context.Context, string domain): []GetDomainModeratorsRow, GetGlobalModerators(context.Context): []GetGlobalModeratorsRow, GetAdmins(context.Context): []GetAdminsRow, GetNewestUser(context.Context): User, GetUserCount(context.Context): int64, GetLogsForIP(context.Context, string IP): []GetLogsForIPRow, GetLogsForUser(context.Context, int64 userID): []GetLogsForUserRow

**GetLogsForUserRow**

**Class Description:** The result of executing the GetLogsForUser method of the Queries object.

**Class Data Members:** int64 ID, string ID, string Url, time.Time AtTime

**GetLogsForIP**

**Class Description:** The result of executing the GetLogsForUser method of the Queries object.

**Class Data Members:** int64 ID, int64 UserID, string Username, string Url, time.Time AtTime

**GetAdminsRow**

**Class Description:** The result of executing the GetGlobalModerators method of the Queries object.

**Class Data Members:** int64 ID, int64 AssignedTo, string AssignedToUsername, time.Time AssignedAt, int64 AssignedBy, string AssignedByUsername

**GetGlobalModeratorsRow**

**Class Description:** The result of executing the GetGlobalModerators method of the Queries object.

**Class Data Members:** int64 ID, int64 AssignedTo, string AssignedToUsername, time.Time AssignedAt, int64 AssignedBy, string AssignedByUsername

**GetDomainModeratorsRow**

**Class Description:** The result of executing the GetDomainModerators method of the Queries object.

**Class Data Members:** int64 ID, int64 AssignedTo, string AssignedToUsername, time.Time AssignedAt, int64 AssignedBy, string AssignedByUsername

**GetModRecordsForModeratorRow**

**Class Description:** The result of executing the GetLogsForDateRange method of the Queries object.

**Class Data Members:** int64 ID, int64 TakenBy, string Username, int64 CommentID, string Reason, time.Time TakenOn, Boolean SetHiddenTo, Boolean SetRemovedTo, int64 AssociatedReport

**GetLogsForDateRangeRow**

**Class Description:** The result of executing the GetLogsForDateRange method of the Queries object.

**Class Data Members:** int64 ID, int64 UserID, string Username, string IP, string URL

**GetFeedbackRow**

**Class Description:** The result of executing the GetFeedback method of the Queries object.

**Class Data Members:** int64 ID, int64 UserID, string Username, string Type, time.Time SubmittedAt, string Content, Boolean Hidden

**GetCommentReportsRow**

**Class Description:** The result of executing the GetCommentReports method of the Queries object.

**Class Data Members:** int64 ID, int64 ReportingUser, string Username, int64 Comment, string Reason, bool ActionTaken, time.Time TimeCreated

**GetBanRecordsRow**

**Class Description:** The result of executing the GetBanRecords method of the Queries object.

**Class Data Members:** int64 ID, int64 TakenBy, string TakenByUsername, int64 TargetUser, string TargetUsername, string Reason, time.Time takenOn, string Domain, bool SetBannedTo

**CreatePWResetRecordParams**

**Class Description:** Parameters used for executing the CreatePWResetRecord method of the Queries object.

**Class Data Members:** in64 UserID, stirng VerifyCode

**CreateVerificationRecord Params**

**Class Description:** Parameters used for executing the CreateVerificationRecord method of the Queries object.

**Class Data Members**: int64 UserID, string VerifyCode

**UpdateUserVerificationParams**

**Class Description:** Parameters used for executing the UpdateUserVerification method of the Queries object.

**Class Data Members**: int64 ID, Boolean IsVerified

**UpdateUserPasswordParams**

**Class Description:** Parameters used for executing the UpdateUserPassword method of the Queries object.

**Class Data Members:** int64 ID, string Password

**CreateDomainModeratorAssignmentParams**

**Class Description**: Parameters used for executing the CreateDomainModeratorAssignment method of the Queries object.

**Class Data Members**: string Domain, int64 AssignedTo, int64 AssignedBy, bool isDeactivation

**CreateGlobalModeratorAssignmentParams**

**Class Description**: Parameters used for executing the CreateGlobalModeratorAssignment method of the Queries object.

**Class Data Members**: int64 AssignedTo, int64 AssignedBy, Boolean isDeactivation

**CreateDomainBanRecordParams**

**Class Description**: Parameters used for executing the CreateGlobalModeratorAssignment method of the Queries object.

**Class Data Members**: int64 TakenBy, int64 TargetUser, string Reason, string Domain, Boolean SetBannedTo

**UpdateUserBanStatusParams**

**Class Description:** Parameters used for executing the UpdateUserBanStatus method of the Queries object.

**Class Data Members:** int64 ID, Boolean Banned

**UpdateUserEmailParams**

**Class Description:** Parameters used for executing the UpdateUserEmail method of the Queries object.

**Class Data Members:** int64 ID, string Email

**UpdateFeedbackHiddenParams**

**Class Description:** Parameters used for executing the UpdateFeedbackHidden method of the Queries object.

**Class Data Members:** int64 ID, bool Hidden

**UpdateUserBlurbParams**

**Class Description:** Parameters used for executing the UpdateUserBlurb method of the Queries object.

**Class Data Members:** in64 ID, string ProfileBlurb

**CreateCommentParams**

**Class Description:** Parameters used for executing the CreateComment method of the Queries object.

**Class Data Members:** int64 PathID, int64 Author, string Content, int64 Parent

**CreateCommentVoteParams**

**Class Description:** Parameters used for executing the CreateCommentVote method of the Queries object.

**Class Data Members:** int64 CommentID, string Category, int64 UserID, int8 Value

**UpdateCommentVoteParams**

**Class Description:** Parameters used for executing the UpdateCommentVote method of the Queries object.

**Class Data Members:** int64 UserID, int64 CommentID, int8 Value

**DeleteCommentVoteParams**

**Class Description:** Parameters used for executing the DeleteCommentVote method of the Queries object.

**Class Data Members:** int64 UserID, int64 CommentID

**CreateFeedbackParams**

**Class Description:** Parameters used for executing the CreateFeedback method of the Queries object.

**Class Data Members:** int64 UserID, string Type, string Content

**GetCommentsAtPathRow**

**Class Description:** The result of executing the GetCommentsAtPath method of the Queries object.

**Class Data Members:** int64 ID, int64 Author, string Content, time.Time CreatedAt, int64 Parent, Boolean Hidden, Boolean Removed, string Username

**GetCommentVotesRow**

**Class Description:** The result of executing the GetCommentVotes method of the Queries object.

**Class Data Members:** int64 UserID, string Category, int8 Value

**CreateModerationRecordParams**

**Class Description:** Parameters used for executing the CreateModerationRecord method of the Queries object.

**Class Data Members:** int64 TakenBy, int64 CommentID, string Reason, Boolean SetHiddenTo, Boolean SetRemovedTo, int64 AssociatedReport

**UpdateCommentHiddenParams**

**Class Description:** Parameters used for executing the UpdateCommentHidden method of the Queries object.

**Class Data Members:** int64 ID, Boolean Hidden

**UpdateCommentRemoveParams**

**Class Description:** Parameters used for executing the UpdateCommentRemove method of the Queries object.

**Class Data Members:** int64 ID, Boolean Removed

**CreateLogParams**

**Class Description:** Parameters used for executing the CreateLog method of the Queries object.

**Class Data Members:** int64 UserID, string IP, string URL

**GetUserAssignmentRow**

**Class Description:** The result of executing the GetUserAdminAssignments method of the Queries object.

**Class Data Members:** int64 ID, time.Time AssignedAt

**GetUserDomainModeratorAssignmentsRow**

**Class Description:** The result of executing the GetUserDomainModeratorAssignments method of the Queries object.

**Class Data Members:** int64 ID, time.Time AssignedAt, string Domain

**GetUserGlobalModeratorAssignmentsRow**

**Class Description:** The result of executing the GetUserGlobalModeratorAssignments method of the Queries object.

**Class Data Members:** int64 ID, time.Time AssignedAt

**CreateCommentReportParams**

**Class Description:** Parameters used for executing the CreateCommentReport method of the Queries object.

**Class Data Members:** int64 ReportingUser, int64 Comment, string Reason

**CreateUserParams**

**Class Description:** Parameters used for executing the CreateUser method of the Queries object.

**Class Data Members:** string Username, string Password, string Email

##### Example, SQLC Generated Code

// Code generated by sqlc. DO NOT EDIT.

// versions:

// sqlc v1.15.0

// source: commentReports.sql

package generated

import (

"context"

"database/sql"

)

const createCommentReport = `-- name: CreateCommentReport :exec

INSERT INTO "CommentReports" (

reporting\_user,

comment,

reason

) VALUES ($1, $2, $3)

`

type CreateCommentReportParams struct {

ReportingUser *int64* `json:"reporting\_user"`

Comment *int64* `json:"comment"`

Reason sql.NullString `json:"reason"`

}

func (q \*Queries) CreateCommentReport(ctx context.Context, arg CreateCommentReportParams) *error* {

\_, err := q.db.ExecContext(ctx, createCommentReport, arg.ReportingUser, arg.Comment, arg.Reason)

return err

}

#### Package communication

Package communication describes the client-server communication entities and the server-client communication entities for use with the back end. These were previously described in the Specifications document with some minor changes as described in the Ties to the Specifications Document section of this document. The same communication classes described herein are realized with typescript type definitions on the front end so that the front end and back-end share communication paths.

##### Client-Server Communication Entities

**AssignDomainModerator**

**Description**: AssignDomainModerator is dispatched when a global moderator or administrator assigns a new domain moderator.

**Data Members**: int64 AssignTo, string Domain

**AssignGlobalModerator**

**Description**: AssignGlobalModerator is dispatched when an administrator assigns a new global moderator.

**Data Members**: int64 AssignTo

**Ban**

**Description**: Ban is dispatched when a moderator or administrator bans a user.

**Data Members**: int64 UserId, string Reason, string Domain

**ChangeEmail**

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

**Data Members**: string NewEmail, string Password

**ChangeFeedback**

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

**Data Members**: bool Delete, int64 FeedbackId, bool SetHiddenTo

**ChangeProfileBlurb**

**Description**: ChangeProfileBlurb is dispatched to the server when a client updates their profile blurb.

**Data Members**: string NewBlurb

**CommentReply**

**Description**: 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.

**Data Members**: int64 ReplyingTo, string Reply

**CommentVote**

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

**Data Members**: int64 VotingOn, string VoteType, int8 Value

**Feedback**

**Description**: Feedback is dispatched to the Server when a user submits feedback on Comment

**Data Members**: string FeedbackType, string Content

**GetComments**

**Description**: 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.

**Data Members**: string Url, string SortedBy, bool SortAscending

**GetUserProfile**

**Description**: GetUserProfile is dispatched to the server when the user needs to see a user’s profile.

**Data Members**: int64 UserId

**Login**

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

**Data Members**: string Username, string Password

**Logout**

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

**Data Members**: None

**Moderate**

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

**Data Members**: int64 CommentId, int64 AssociatedReport, bool SetHiddenTo, bool SetRemovedTo, string Reason

**PasswordResetCode**

**Description**: PasswordResetCode is dispatched by a user when they enter a password reset code. After a user clicks “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.

**Data Members**: int32 Code

**PasswordReset**

**Description**: PasswordReset is dispatched to the server when a password reset is requested. The client supplies the email associated with their account.

**Data Members**: string Email

**PostCommentReport**

**Description**: PostCommentReport is dispatched to the server when the user reports a comment.

**Data Members**: int64 CommentId, string Reason

**Register**

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

**Data Members**: string Username, string Password, string RetypePassword, string Email, bool AgreedToTerms

**RequestVerification**

**Description**: RequestVerification is dispatched to the server when the client wants a new validation code. 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.

**SetNewPass**

**Description**: SetNewPass is dispatched to the Server when the user changes their password. 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.

**Data Members**: string Password, string RetypePassword

**Verify**

**Description**: Verify is dispatched to the server when the client inputs a validation code they received in an email to verify their account.

**Data Members**: Code int32

**ViewBans**

**Description**: ViewBans is dispatched to the server when an admin requests records of banned users.

**Data Members**: []string ForDomains

**ViewCommentReports**

**Description**: 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.

**ViewDomainReport**

**Description**: ViewDomainReport is dispatched to the server when an admin requests a report on a domain.

**Data Members**: string domain

**ViewUsersReport**

**Description**: ViewUsersReport is dispatched to the server when an admin requests a report on the overall users of comment anywhere.

**Data Members**:

**ViewFeedback**

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

**Data Members**: int64 From, int64 To, string FeedbackType

**ViewLogs**

**Description**: ViewLogs is dispatched to the server when an admin requests access logs.

**Data Members**: int64 ForUserId, string ForIp, string ForDomain, int64 StartingAt, int64 EndingAt

**ViewModRecords**

**Description**: ViewModRecords 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.

**ViewMods**

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

**Data Members**: []string ForDomains

##### Server-Client Communication Entities

**AdminAccessLog**

**Description**: AdminAccessLog contains data needed by Admins to see an access log.

**Data Members**: string Ip, int64 LogId, string Url, string UserId, string Username

**AdminDomainReport**

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

**Data Members**: int32 CommentCount, string Domain

**AdminUsersReport**

**Description**: AdminUsersReport is dispatched when an Admin requests the Users report.

**Data Members**: int32 LoggedInUserCount, int64 NewestUserId, string NewestUsername, int64 UserCount

**BanRecord**

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

**Data Members**: string BannedFrom, int64 BanRecordId, int64 BannedUserId, string BannedUsername, int64 BannedByUserID, string BannedByUsername, bool SetBannedTo, int64 BannedAt, string Reason

**Comment**

**Description**: Comment provides the data the Front End needs to render a comment.

**Data Members**: string UserId, int64 CommentId, string Content, CommentVoteDimension Factual, CommentVoteDimension Funny, CommentVoteDimension Agree, bool Hidden, int64 Parent, bool Removed, int64 TimePosted, string Username

**CommentReport**

**Description**: 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.Data Members: bool ActionTaken, Comment CommentData, string ReasonReported, int64 ReportId, int64 ReportingUserId, string ReportingUsername, int64 TimeReported

**CommentVoteDimension**

**Description**: CommentVoteRecord contains data for the number of votes on a comment.

**Data Members**: int8 AlreadyVoted, int64 Downs, int64 Ups

**DomainModeratorRecord**

**Description**: DomainModeratorRecord contains data needed by Admins to see information about DomainModerator assignments.

**Data Members**: int64 GrantedAt, int64 GrantedBy, string GrantedByUsername, int64 GrantedTo, string GrantedToUsername, int64 RecordId

**FeedbackRecord**

**Description**: 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.

**Data Members**: string Content, bool Hide, int64 Id, int64 SubmittedAt, string FeedbackType, int64 Userid, string Username

**GlobalModerator**

**Description**: GlobalModerator record contains data needed by Admins to see information about GlobalModerator assignments.

**Data Members**: int64 GrantedAt, int64 GrantedBy, string GrantedByUsername, int64 GrantedTo, string GrantedToUsername, int64 RecordId

**LoginResponse**

**Description**: LoginResponse is sent to the client when they successfully log in.

**Data Members**: UserProfile LoggedInAs

**Message**

**Description**: Message is a general communication entity used to provide feedback to a client that some action has completed (or not completed) on requests where the client has not asked for any particular data

**Data Members**: bool Success, string Text

**ModerationRecord**

**Description**: 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.

**Data Members**: CommentReport AssociatedReport, int64 ModerationRecordId, int64 ModeratorUserId, string ModeratorUsername, string Reason, bool SetHiddenTo, bool SetRemovedTo, int64 TimeModerated

**UserProfile**

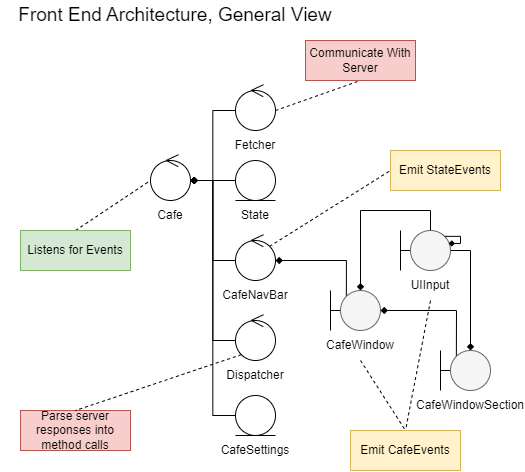
**Description**: UserProfile contains data needed by the Front End to display a profile for a user.

**Data Members**: int64 CreatedOn, []string DomainsModerating, bool IsAdmin, bool IsDomainModerator, bool IsGlobalModerator, string ProfileBlurb, int64 UserId, string Username

#### Front End

The Comment Anywhere front end consists of classes written in TypeScript/JavaScript. It is responsible for displaying the data to the user, capturing user interaction, converting those actions into Client-Server communication entities, and dispatching them to the server. It runs in the browser engine.

##### Front End Architecture Diagram



The above diagram is a general overview of the Front End Classes and how they interact. The subclasses that inherit from CafeWindow, UIInput, and CafeWindowSection are compressed into their base classes in the diagram, and the possible relationships between the three are shown. That is, a CafeWindow may be composed of some number of UIInputs and some number of CafeWindowSections while a CafeWindowSection may be composed of some number of UIInputs. UIInputs may be composed of additional UIInputs. The actual composition of a given CafeWindow or CafeWindowSection varies.

UIInputs display a Server-Client Communication Entity to the User, such as a Comment. They may provide controls in the form of HTMLElements for the user to interact with that entity. If that interaction needs to be sent to the Server, the UIInput builds the associated Client-Server communication entity from the User inputs and emits it on the global “document” object of the Document-Object-Model. (DOM).

Cafe, the top level instance of the program, listens for CafeEvents on the “document” object. When it detects one, it passes it along to Fetcher, along with a callback function, and fetcher dispatches that data to the server. When the server responds, Fetcher passes the response to callback provided by Cafe. In this callback, Cafe provides the server response to Dispatcher along with a reference to itself. Dispatcher calls a distinct method for each type of server response. Each of these methods are able to send the data provided by the Server to where it belongs in the front-end.

Another type of event that can be emitted is a StateEvent, which is emitted by NavBar when the client clicks one of the navigation buttons. Rather than directly update the UI, NavBar instead emits a StateEvent so the primary State object can be updated with the change. Cafe listens for this event too, updates the state accordingly, and may make method calls depending on the state the user wants to transition to.

##### Description of Front End Classes

**Cafe**

**Class Description**: Cafe stands for "Comment Anywhere Front End". Cafe is the base class that is composed of other major classes used in the Front end. It is responsible for updating the State and listening for user input events on the DOM, and transmitting them to the fetcher when appropriate.

**Class Data Members**: Fetcher fetcher, State state, CafeNavbar navbar, Settings settings, Dispatcher dispatcher, string currentUrl

**Class Methods:** userChange(Server.UserProfile | undefined), pageChange(string), changeState(string), checkForResponses(), stateChangeEventReceived(), clientEventReceived(), settingEventReceived(Settings)

**Fetcher**

**Class Description**: Fetcher is responsible for dispatching requests to the server at the appropriate API endpoints and populating its responses object with the server responses.

**Class Data Members:** responses ServerResponse[]

**Class Methods:** fetch(string, client-server Communication Entity), getAndClearResponses(

**State**

**Class Description:** State holds the current state of the front end, including who is logged in and what window is being viewed. Cafe passes State to NavBar to realize a state change.

**Class Data Members:** string viewing, UserProfile ownProfile

**Class Methods:** loadProfile(UserProfile | undefined), setViewingTo(string)

**CafeNavbar**

**Class Description:** CafeNavBar displays navigation buttons for the user to move between states, holds the active window, and holds a general message display object.

**Class Data Members:** CafeCommentsWindow commentsWindow, CafeSettingsWindow settingsWindow, CafeModerationWindow modWindow, CafeAdminWindow adminWindow, CafeLoginWindow loginWindow, CafeRegisterWindow registerWindow, CafePWResetCodeWindow pwResetWindow, CafeNewPasswordWindow newPwWindow, CafeMessageDisplay globalMessage, CafeUserDisplay userHoverDisplay, CafeWindow currentlyViewing, HTMLButtonElement commentsButton, HTMLButtonElement settingsButton, HTMLButtonElement modButton, HTMLButtonElement adminButton, HTMLButtonElement loginButton, HTMLButtonElement registerButton, HTMLButtonElement logoutButton

**Class Methods:** commentsButtonClicked(), settingsButtonClicked(), modButtonClicked(), adminButtonClicked(), loginButtonClicked(), registerButtonClicked(), logoutButtonClicked(), setFromState(state:State), displayMessage(data:Server.Message)

**CafeSettings**

**Class Description:** CafeSettings holds comment sorting settings configured by the user.

**Class Data Members:** Boolean viewHidden, string sortBy, boolean sortAscending

**Dispatcher**

**Class Description:** Dispatcher is responsible for parsing an array of server responses and dispatching them to the appropriate objects around the front end for rendering to the user.

**Class Methods:** dispatch(serverResponse[], Cafe), dispatchMessage(Message, CafeNavbar), dispatchUserUpdate(UserProfile, Cafe), dispatchFeedbackUpdate(FeedbackRecord, FeedbackReportSection) , dispatchCommentUpdate(Comment, CafeCommentsWindow) , dispatchComments(Comment[], CafeCommentsWindow), dispatchUserProfile(UserProfile, CafeUserDisplay), dispatchLoginResponse(LoginResponse, Cafe), dispatchLogoutResponse(LogoutResponse, Cafe), dispatchPwResetCodeResponse(any, Cafe), dispatchPwResetReqResponse(any, Cafe), dispatchReqVerificationResponse(any, CafeMessageDisplay), dispatchNewPassResponse(any, CafeMessageDisplay) , dispatchVerifyResponse(any, CafeMessageDisplay), dispatchBanRecords(BanRecord[], BanRecordsSection) , dispatchCommentReports(CommentReport[], CommentReportsSection) , dispatchCommentReportUpdate(CommentReport, CommentReportsSection) , dispatchFeedbacks(FeedbackRecord[], FeedbackReportSection) , dispatchLogs(AdminAccessLog[], LogsSection) , dispatchModRecords(ModerationRecord[], ModActionsReportSection) , dispatchGlobalMods(GlobalModeratorRecord[], ModeratorsReportSection) , dispatchDomainMods(DomainModeratorRecord[], ModeratorsReportSection) , dispatchDomainReport(AdminDomainReport, CafeAdminWindow) , dispatchUsersReport(AdminUsersReport, CafeAdminWindow)

**CafeWindow**

**Class Description**: CafeWindow is the base class for all CafeWindows. CafeWindows hold all the content of CommentAnywhere, except for the navbar buttons and message. They correspond to the frond end states. Only one CafeWindow is visible at any given time, and is displayed and hidden by the CafeNavBar instance.

**Class Data Members**: HTMLDivElement el

**Class Methods**:show(), hide()

**CafeCommentsWindow**

**Extends:** CafeWindow

**Class Description**: CafeCommentsWindow displays all comments for the current page. It is responsible for repopulating with new comments when comments for a new page are retrieved and updating comments as new ones are added and voted on.

**Class Data Members**: Comment[] data, lastSettings?: CafeSettings, Map<number, CafeComment> displayedComments,CafeCommentSortDisplay commentSortSettings

**Class** **Methods**:populateNewComments(Comment[]), updateComment(Comment), resortComments(), updateFromSettings(CafeSettings)

**CafeSettingsWindow**

**Extends:** CafeWindow

**Class Description**: CafeSettingsWindow displays user settings and provides options for the user to reset their password or verify their email.

**Class Data Members**: CafeCommentSortDisplay commentSortSettings, HTMLButtonElement passwordResetButton, HTMLButtonElement requestEmailValidationButton, HTMLButtonElement verifyEmailButton, HTMLInputElement verifyCodeInput, HTMLInputElement verifyCodeSubmit

**Class** **Methods**: updateFromSettings(CafeSettings), verifyEmailClicked(), verifyCodeSubmitClicked(), passwordResetClicked()

**CafeModerationWindow**

**Extends:** CafeWindow

**Class** **Description**: CafeModerationWindow allows moderator interactions with Comment Anywhere. It is shown when a user clicks the moderation button on the Navbar, transitioning them to that state.

**Class** **Data** **Members**: ModeratorsReportSection moderators, ModActionsReportSection modActions, BanRecordsSection banRecords, CommentReportsSection reports

**CafeAdminWindow**

**Extends:** CafeWindow

**Class** **Description**: CafeModerationWindow allows admin interactions with Comment Anywhere. It is shown when a user clicks the admin button on the Navbar, transitioning them to that state.

**Class** **Data** **Members**: FeedbackReportSection feedbackReport, LogsSection logs, domainReports CafeDomainReportDisplay[]

**Class** **Methods**: CafeUsersReportDisplay userReport, HTMLInputElement domainInput, HTMLButtonElement domainRequestButton, HTMLButtonElement usersReportButton,domainRequestButtonClicked(), showDomainReport(AdminDomainReport), usersReportRequestButtonClicked(), showUsersReport(AdminUsersReport),

**CafeLoginWindow**

**Extends:** CafeWindow

**Class** **Description**: CafeLoginWindow is displayed when the user clicks the "Login" button in the navbar and transitions to the login state.

**Class** **Data** **Members**: HTMLInputElement username, HTMLInputElement password, HTMLButtonElement submitLoginButton, HTMLButtonElement forgotPasswordButton,

**Class** **Methods**: submitLoginButtonClicked(), forgotPasswordButtonClicked()

**CafeRegisterWindow**

**Extends:** CafeWindow

**Class** **Description**: CafeRegisterWindow is displayed when the user clicks the "Register" button in the navbar and transitions to the register state.

**Class** **Data** **Members**: HTMLInputElement username, HTMLInputElement password, HTMLInputElement retypePassword, HTMLInputElement email, HTMLInputElement agreedToTerms, HTMLButtonElement submitRegister,

**Class** **Methods**:submitRegisterClicked()

**CafePwResetRequestWindow**

**Extends:** CafeWindow

**Class** **Description**: CafePWResetRequestWindow is displayed when the user transitions to the request password reset form state.

**Class** **Data** **Members**: HTMLInputElement email, HTMLButtonElement submitPWResetRequestButton,

**Class** **Methods**:submitButtonClicked()

**CafePWResetCodeWindow**

**Extends:** CafeWindow

**Class** **Description**: CafePWResetCodeWindow is displayed when the user transitions to the input password reset code state.

**Class** **Data** **Members**: HTMLInputElement code, HTMLButtonElement submitCodeButton,

**Class** **Methods**:submitButtonClicked()

**CafeNewPasswordWindow**

**Extends:** CafeWindow

**Class** **Description**: CafeNewPasswordWindow is displayed when the user transitions to the input new password state.

**Class** **Data** **Members**: HTMLInputElement password, HTMLInputElement newPassword, HTMLButtonElement submitButton,

**Class** **Methods**:submitButtonClicked()

**CafeWindowSection**

**Class** **Description**: CafeWindowSection represents an area of a window that performs some particular task or shows some collection of a data. CafeWindows may be composed of several CafeWindowSections.

**Class** **Data** **Members**: HTMLElement el,

**FeedbackReportSection**

**Extends:** CafeWindowSection

**Class** **Description**: FeedbackReportSection is part of the CafeAdminWindow and displays information about user-submitted feedback.

**Class** **Data** **Members**: Map<number, CafeFeedbackDisplay> feedbacks,

**Class** **Methods**:HTMLInputElement from, HTMLInputElement to, HTMLSelectElement feedbackType, HTMLButtonElement requestFeedbackButton,updateFeedback(FeedbackRecord), requestFeedbackClicked(), populateFeedback(FeedbackRecord[]), clearFeedback(),

**ModeratorsReportSection**

**Extends:** CafeWindowSection

**Class** **Description**: ModeratorsReportSection is part of the CafeModerationWindow and allows viewing of moderator assignment records.

**Class** **Data** **Members**: HTMLInputElement forDomainInput, HTMLButtonElement requestDomainModeratorsButton, HTMLButtonElement requestGlobalModeratorsButton, Map<number, CafeDomainModDisplay> domainModRecords, Map<number, CafeGlobalModDisplay> globalModRecords

**Class** **Methods**: HTMLButtonElement clearModRecordsButton,requestDomainModsClicked(), requestGlobalModsClicked(), populateDomainModerators(DomainModeratorRecord[]), populateGlobalModerators(GlobalModeratorRecord[]), clearModsClicked(),

**ModActionsReportSection**

**Extends:** CafeWindowSection

**Class** **Description**: ModActionsReportSection is part of CafeModerationWindow and provides functionality for viewing moderation actions taken on comments.

**Class** **Data** **Members**: Map<number, CafeModRecordDisplay> modRecords, HTMLButtonElement requestModRecordsButton, HTMLButtonElement clearModActionsButton

**Class** **Methods**: requestModRecordsButtonClicked(), populateModActions(ModerationRecord[]), updateModAction(ModerationRecord), clearModActionsClicked()

**BanRecordsSection**

**Extends:** CafeWindowSection

**Class** **Description**: BanRecordsSection is part of CafeModerationWindow. It is used to display records of bannings to other moderators.

**Class** **D**ata **Members**: banRecords: Map<number, CafeBanRecord>, HTMLButtonElement requestGlobalBansButton, HTMLInputElement forDomain, HTMLButtonElement requestDomainBansButton

**Class Methods**: populateBanRecords(BanRecord[]), requestGlobalBansClicked(), requestDomainBansClicked(), clearBanRecords()

**CommentReportsSection**

**Extends:** CafeWindowSection

**Class** **Description**: CommentReportsSection is part of CafeModerationWindow. It is responsible for displaying comment reports to moderators.

**Class** **Data** **Members**: Map<number, CafeCommentReportDisplay> reports, HTMLButtonElement viewReportsButton

**Class** **Methods**: populateCommentReports(CommentReport[]), viewReportsClicked(), updateCommentReport(CommentReport), clearCommentReports()

**LogsSection**

**Extends:** CafeWindowSection

**Class** **Description**: LogsSection is part of CafeAdminWindow. It is responsible for populating log entries.

**Class Data** **Members**: HTMLInputElement forUser, HTMLInputElement forIP, HTMLInputElement forDomain, HTMLInputElement startingAt, HTMLInputElement endingAt, HTMLButtonElement submitLogsRequestButton, Map<number, CafeLogDisplay> logs

**Class** **Methods**: submitLogsRequestClicked(), populateLogs(AdminAccessLog[]), clearLogs()

**UIInput<Type>**

**Generic** **Description**: Type represents the underlying data that UIInput displays.

**Class** **Description**: UIInput is a base class for all boundary classes representing a server-client communication entity. It provides methods for gracefully deleting its instance, binding listeners, disabling, and re-enabling the UIInput.

**Class** **Data** **Members**: Type data, HTMLElement el, HTMLElement blocker, [HTMLElement, Function, string][] listeners

**Class** **Methods**: clickListen(HTMLElement, Function[], Boolean), disable(), enable(), destroy()

**CafeComment**

**Class** **Description**: CafeComment is responsible for rending a Comment communication entity.

**Extends**: UIInput<Comment>

**Class** **Data** **Members**: HTMLDivElement Username, HTMLDivElement Content, CafeCommentVoter funny, CafeCommentVoter factual, CafeCommentVoter agree, HTMLTextArea replyText, HTMLTextArea reportText, HTMLButtonElement submitReplyButton, HTMLButtonElement submitReportButton

**Class** **Methods**: replyClicked(), submitReplyClicked(), reportClicked(), submitReportClicked()

**CafeCommentVote**

**Class** **Description**: CafeCommentVote is responsible for rendering a CommentVoteDimension communication entity.

**Extends**: UIInput<CommentVoteDimension>

**Class** **Data** **Members**: number commentId, string voteType, HTMLDivElement voteLabel, HTMLButtonElement up, HTMLButtonElement down, HTMLDivElement total

**Class** **Methods**: upVoteClicked(), downVoteClicked()

**CafeMessageDisplay**

**Class** **Description**: CafeMessageDisplay is responsible for rendering a Message communication entity.

**Extends**: UIInput<Message>

**Class** **Data** **Members**: HTMLDivElement displayedText

**Class** **Methods**: updateMessage(Message), clearMessage()

**CafeUserDisplay**

**Class** **Description**: CafeUserDisplay displays UserProfile communication entity data with HTML Elements.

**Extends**: UIInput<UserProfile>

**Class** **Data** **Members**: HTMLDivElement Username, HTMLDivElement CreatedOn,HTMLDivElement DomainsModerating, HTMLDivElement IsAdmin, HTMLDivElement IsDomainModerator, HTMLDivElement IsGlobalModerator, HTMLDivElement ProfileBlurb

**Class** **Methods**: changeProfile(UserProfile), hide(), show()

**CafeFeedbackDisplay**

**Class** **Description**: CafeFeedbackDisplay displays a feedback record for an admin to view.

**Extends**: UIInput<FeedbackRecord>

**Class** **Data** **Members**: HTMLDivElement Content, HTMLDivElement Hide, HTMLDivElement SubmittedAt, HTMLDivElement FeedbackType, HTMLDivElement Username, HTMLButtonElement ToggleHidden

**Class** **Methods**: hideClicked()

**CafeOwnProfileDisplay**

**Class** **Description**: CafeOwnProfileDisplay is used to display a User's own profile to them.

**Extends**: UIInput<UserProfile>

**Class** **Data** **Members**: HTMLDivElement Username, HTMLDivElement CreatedOn,HTMLDivElement DomainsModerating, HTMLDivElement IsAdmin, HTMLDivElement IsDomainModerator, HTMLDivElement IsGlobalModerator, HTMLDivElement ProfileBlurb, HTMLButtonElement editProfileBlurbButton, HTMLTextAreaElement editProfileTextarea, HTMLButtonElement editProfileSubmitButton, HTMLButtonElement changePasswordButton

**Class** **Methods**: updateProfile(UserProfile), editProfileBlurbClicked(), editProfileSubmitClicked()

**CafeBanRecordDisplay**

**Class** **Description**: CafeBanRecordDisplay displays an individual ban or unban record.

**Extends**: UIInput<BanRecord>

**Class** **Data** **Members**: HTMLDivElement BannedFrom, HTMLDivElement BannedUsername, HTMLDivElement BannedByUsername, HTMLDivElement SetBannedTo, HTMLDivElement BannedAt, HTMLDivElement Reason

**Class** **Methods**: bannedUserClicked(MouseEvent) ,bannedByClicked(MouseEvent)

**CafeCommentReportDisplay**

**Class** **Description**: CafeCommentReportDisplay displays data for a single CommentReport.

**Extends**: UIInput<CommentReport>

**Class** **Data** **Members**: HTMLDivElement ActionTaken, CafeComment CommentData, HTMLDivElement ReasonReported, HTMLDivElement ReportingUsername, HTMLDivElement TimeReported, HTMLButtonElement moderateButton, HTMLInputElement setHiddenTo, HTMLInputElement setRemovedTo, HTMLTextAreaElement reason, HTMLButtonElement submitModerationButton

**Class** **Methods**: reportingUserClicked(MouseEvent) ,moderateButtonClicked(), submitModerationButtonClicked()

**CafeLogDisplay**

**Class** **Description**: CafeLogDisplay displays data from a single AdminAccessLog

**Extends**: UIInput<AdminAccessLog>

**Class** **Data** **Members**: HTMLDivElement Ip, HTMLDivElement Url, HTMLDivElement Username

**Class** **Methods**: usernameClicked(MouseEvent)

**CafeModRecordDisplay**

**Class** **Description**: CafeModRecordDisplay displays data from a ModerationRecord.

**Extends**: UIInput<ModerationRecord>

**Class** **Data** **Members**: CafeCommentReportDisplay AssociatedReport?, HTMLDivElement ModeratorUsername, HTMLDivElement Reason, HTMLDivElement SetHiddenTo, HTMLDivElement SetRemovedTo, HTMLDivElement TimeModerated

**Class** **Methods**: moderatorUsernameClicked(MouseEvent)

**CafeGlobalModDisplay**

**Class** **Description**: CafeGlobalModDisplay renders data from a GlobalModeratorRecord.

**Extends**: UIInput<GlobalModeratorRecord>

**Class** **Data** **Members**: HTMLDivElement GrantedAt, HTMLDivElement GrantedByUsername, HTMLDivElement GrantedToUsername

**Class** **Methods**: grantedByUsernameClicked(MouseEvent), grantedToUsernameClicked(MouseEvent)

**CafeDomainModDisplay**

**Class** **Description**: CafeDomainModDisplay displays data for a DomainModeratorRecord.

**Extends**: UIInput<DomainModeratorRecord>

**Class** **Data** **Members**: HTMLDivElement Domain, HTMLDivElement GrantedAt, HTMLDivElement GrantedByUsername, HTMLDivElement GrantedToUsername

**Class** **Methods**: grantedByUsernameClicked(MouseEvent), grantedToUsernameClicked(MouseEvent)

**CafeDomainReportDisplay**

**Class** **Description**: CafeDomainReportDisplay displays data from an AdminDomainReport object.

**Extends**: UIInput<AdminDomainReport>

**Class** **Data** **Members**: HTMLDivElement Domain, HTMLDivElement CommentCount, HTMLButtonElement xButton

**Class** **Methods**: xButtonClicked()

**CafeUsersReportDisplay**

**Class** **Description**: CafeUsersReportsDisplay displays data from an AdminUsersReport object.

**Extends**: UIInput<AdminUsersReport>

**Class** **Data** **Members**: HTMLDivElement LoggedInUserCount, HTMLDivElement NewestUsername, HTMLDivElement UserCount

**Class** **Methods**: newestUserClicked(MouseEvent), update(AdminUsersReport)

**CafeCommentSortDisplay**

**Class** **Description**: CafeCommentSortDisplay shows the users comment viewing settings and allows the user to change them.

**Extends**: UIInput<CafeSettings>

**Class** **Data** **Members**: HTMLSelectElement sortBy,HTMLInputElement viewHidden,HTMLInputElement sortAscending

**Class** **Methods**: settingChange(), updateFromSettings(CafeSettings)

### Functional Descriptions

#### Package util

**dbCredentials**

**ConnectString**() : string

*Output*: ConnectString forms and returns a Postgres connection string from its members, in the form “host=%s, port=%s, user=%s, password=%s, dbname=%s, sslmode=disable”.

*Returns*: A string which can be used by the SQL package to connect to a Postgres instance.

**loadDBEnv**(),

*Output*: loadDbEnv makes calls to os.GetEnv for each .env variable needed to populate dbCredential’s members. It validates that each exists and loads their values into dbCredential’s members.

*Error*: If loadDBEnv fails to find any env variables it needs, an error message is printed to the console and the process shuts down. The server cannot connect to the database, and therefore cannot run, without being able to connect to the Database.

**config**

**Load**(string)

*Input*: A path to the .env file.

*Output*: The imported godotenv package function Load() is called with the path as a parameter. This makes the environment variables available. Then dbConfig.loadDBEnv() and serverConfig.loadServerEnv() are called to validate the configuration.

*Error*: If gotdotenv fails to find an env file an error message is printed to the console and the process shuts down. The server and database cannot run without a configuration.

**serverConfig**

**loadServerEnv**()

*Output*: loadServerEnv makes calls to os.GetEnv for each .env variable needed to populate serverConfig’s members. It validates that each exists and loads their values into serverConfig’s members.

*Error*: If loadServerEnv fails to find an env variable, an error message is printed to the console and the process shuts down. The server cannot run without a proper configuration.

#### Package server

**UserControllerInterface**

*Note*: While UserControllerInterface is abstract, the implementing Controllers realize these functions in similar manners. For the sake of brevity, only the interface methods are described and the different realizations are hinted at in the Output description.

**HandleCommandBan**(\*communication.Ban, \*Server)

*Input:* A pointer to communication.Ban and a pointer to Server.

*Output:* If the controller is an Admin or Moderator Controller, Server.UserManager.Ban is called.

**HandleCommandChangeEmail**(\*communication.ChangeEmail, \*Server)

*Input:* A pointer to communication.ChangeEmail and a pointer to Server.

*Output:* If the controller is a member Controller, the database record for the User is updated with the new email and the is\_verified field is set to false until the new email is verified.

**HandleCommandChangeFeedback**(\*communication.ChangeFeedback),

*Input*: A pointer to communication.ChangeFeedback and a pointer to Server.

*Output*: If the controller is an Admin controller, the database record for the Feedback is updated to set hidden to true or false, indicating that the feedback has been reviewed.

**HandleCommandChangePassword**(\*communication.SetNewPass, \*Server)

*Input*: A pointer to communication.SetNewPass and a pointer to Server.

*Output*: If the controller is a Member controller, the user’s password is updated in the database.

**HandleCommandChangeProfileBlurb**(\*communication.ChangeProfileBlurb, \*Server)

*Input*: A pointer to communication.ChangeProfileBlur and a pointer to Server.

*Output*: If the controller is a Member controller, the database record for the User’s profile blurb is changed and the profile blurb is updated in the cache memory associated with the User.

**HandleCommandCommentReply**(\*communication.CommentReply, \*Server)

*Input*: A pointer to communication.CommentReply and a pointer to Server.

*Output*: If the controller is a Member controller, Page.CreateComment is called for the page the controller is on.

**HandleCommandCommentVote**(\*communication.CommentVote, \*Server)

*Input*: A pointer to communication.CommentVote and a pointer to Server.

*Output*: If the controller is a Member controller, Page.VoteComment is called for the page the controller is on.

**HandleCommandFeedback**(\*communication.Feedback, \*Server)

*Input*: A pointer to communication.Feedback and a pointer to Server.

*Output*: If the controller is a Member controller, a new Feedback entry is inserted into the Feedbacks table.

**HandleCommandGetComments**(\*communication.GetComments, \*Server)

*Input*: A pointer to communication.GetComments and a pointer to Server.

*Output*: A new Page is instantiated if one does not already exist for the page that the user wants comments for. GetComments is called for that page, and the returned data is added to the nextResponse field for the controller.

**HandleCommandGetUserProfile**(\*communication.GetUserProfile, \*Server)

*Input*: A pointer to communication.GetUserProfile and a pointer to Server.

*Output*: The database is queried and Server-Client communication Entity, UserProfile, is instantiated and added to the nextResponse field for the controller.

**HandleCommandLogin**(\*communication.Login, \*Server),

*Input*: A pointer to communication. Login and a pointer to Server.

*Output*: If the controller is a GuestController, it calls UserManager.Login.

**HandleCommandLogout**(\*Server)

*Input*: A pointer to the server.

*Output*: If the controller is a Member Controller, it calls UserManager.Logout.

**HandleCommandModerate**(\*communication.Moderate, \*Server)

*Input*: A pointer to communication.Moderate and a pointer to Server.

*Output*: If the controller is at least a Moderator controller, and they have permission to moderate that domain, PageManager.Moderate is called.

**HandleCommandPasswordResetCode**(\*communication.PasswordResetCode, \*Server)

*Input*: A pointer to communication.PasswordResetCode, and a pointer to Server.

*Output*: If the client input a correct request, the database field for that password reset is updated to reflect that a valid code has been submitted and the client may enter a new password.

**HandleCommandPasswordResetRequest**(\*communication.PasswordResetRequest, \*Server)

*Input*: A pointer to communication.PasswordResetRequest containing the clients email, and a pointer to Server.

*Output*: If an email associated with a user is submitted, a new record is inserted into the PasswordResetCodes table and an email is dispatched containing the code.

**HandleCommandCommentReport**(\*communication.PostCommentReport, \*Server)

*Input*: A pointer to communication.PostCommentReport and a pointer to Server.

*Output*: If the controller has appropriate access, a new record is inserted into CommentReports table.

**HandleCommandRequestVerification**(\*communication.RequestVerification, \*Server)

*Input*: A pointer to communication.RequestVerification and a pointer to Server.

*Output*: A new record is inserted into the VerificationCodes table and an email is dispatched containing the verification code.

**HandleCommandVerify**(\*communication.Verify, \*Server)

*Input*: A pointer to communication.Verify and a pointer to Server.

*Output*: If the code matches the data in the VerificationCodes table, the is\_verified field of the User record is changed to true, indicating that the client has verified their email.

**HandleCommandViewBans**(\*communication.ViewBans, \*Server)

*Input*: A pointer to communication.ViewBans and a pointer to Server.

*Output*: If the controller is one with appropriate access, an array of the server-client communication entity BanRecords is created from data in the BanRecords table, converted to a packet, and added to the nextResponse field of the Controller.

**HandleCommandViewCommentReports**(\*communication.ViewCommentReports, \*Server)

*Input*: A pointer to communication.ViewCommentReports and a pointer to Server.

*Output*: If the controller is one with appropriate access, an an array of the server-client communication entity CommentReport is created from data in the CommentReports table, converted to a packet, and added to the nextResponse field of the Controller.

**HandleCommandViewLogs**(\*communication.ViewLogs, \*Server)

*Input*: A pointer to communication.ViewLogs and a pointer to Server.

*Output*: If the controller is one with appropriate access, an array of the server-client communication entity AdminAccessLog is created from the data in the Logs table, converted to a packet, and added to the nextResponse field of the Controller.

**HandleCommandViewModRecords**(\*communication.ViewModRecords, \*Server)

*Input*: A pointer to communication.ViewModRecords and a pointer to Server.

*Output*: If the controller is one with appropriate access, an array of the server-client communication entity ModerationRecord is created from data in the ModerationRecords table,

**HandleCommandViewMods**(\*communication.ViewMods, \*Server)

*Input*: A pointer to communication.ViewMods, and a pointer to Server.

*Output*: If the controller is one with appropriate access, an array of the server-client communication entities DomainModeratorRecords or an array of GlobalModeratorRecords, depending on the client request, is converted to a packet and added to the nextResponse field of the controller.

**Respond**(r http.Request, w http.ResponseWriter), GetCurrentPage() :\*Page, dispatchResponse(r http.Request, w http.ResponseWriter)

*Input*: A pointer to the http.Request and an http.ResponseWriter

*Output*: The controller responds with all packets saved in its nextResponse field by writing them to the body of the http.ResponseWriter

**UserManager**

**Ban**(\*communication.Ban, \*Server server)

*Input*: A pointer to a communication.Ban entity and a pointer to Server.

*Output*: If the User is bannable, a new record is created in the database and the target user’s banned field may be changed. If a controller instance is active for that user, it is deleted.

**Login**(\*UserControllerInterface, \*communication.Login, \*Server server): \*UserControllerInterface

*Input*: A pointer to a communication.Login and a pointer to Server.

*Output*: If the username and password are valid, A new UserControllerInterface is instantiated an added to the UserManager.members map or retrieved from the instantiated controllers. The current guest controller is deleted and removed from the guests map. The new controller is added to the page the guest was on, and the guest controller is removed from that map as well. A communication.LoginResponse entity is created an added to the nextResponse field of the controller.

*Returns*: A pointer the newly instantiated UserControllerInterface.

**Logout**(\*UserControllerInterface, \*Server server): \*GuestController

*Input*: None

*Output*: The controller is removed from the UserManager.members map and the map at the page the controller was on. A new GuestController is instantiated and placed in the UserManager.guests map and the guests map of the page the user was on. A communication.LogoutResponse entity is created and added to the nextResponse field of the guest controller.

*Returns*: A pointer to the newly instantiated GuestControllerInterface

**Register**(\*UserControllerInterface, \*communication.Register, \*Server server)

*Input*: A pointer to the GuestController previously associated with the user, a pointer to the communication.Register entity, and a pointer to the Server.

*Output*: If the user doesn’t already exist and has a valid name and password, a new user is created in the database and communication.Login entity is constructed and UserManger.Login is called.

**GetMemberController**(int64 id): \*UserControllerInterface

*Input*: A 64-bit integer representing a userID.

*Output*: If the controller does not exist in the UserManager.members map, a new controller is instantiated.

*Returns*: A pointer to the associated controller.

**GetGuestController**(int64 id): \*UserControllerInterface

*Input*: A 64-bit integer representing a temporary guest userID.

*Output*: If the controller does not exist in the UserManager.guests map, a new GuestController is instantiated.

*Returns*: A pointer to the associated controller.

**DispatchPasswordResetEmail**(\*UserControllerInterface, \*Server server)

*Input*: A pointer to a user controller interface and a pointer to Server.

*Output*: A new entry is created in the PasswordResetCodes table and an email with that code is dispatched to the email associated with the UserControllerInterface.

**PageManager**

**MoveMemberToPage**(\*UserControllerInterface user, string pagePath, \*Server server),

*Input*: A pointer to a UserControllerInterface, a string representing the path, and a pointer to the server.

*Output*: A new page is instantiated if necessary. The UserControllerInterface is removed from the members map on its current Page and added to the new Page.

**MoveGuestToPage**(\*UserControllerInterface user, string pagePath, \*Server server),

*Input*: A pointer to a UserControllerInterface, a string representing the path, and a pointer to the server.

*Output*: A new Page is instantiated if necessary. The UserControllerInterface is removed from the guest map on its current Page and added to the new Page.

**UnloadEmptyPages**(\*Server server)

*Input*: A pointer to Server.

*Output*: Every instantiated Page in PageManger.pages is iterated through. References to pages which have no controllers in their users or guests map are removed from PageManager.pages to allow the garabage collector to delete them.

**loadPage**(string path, \*Server server)

*Input*: A string representing a URL path.

*Output*: The database is queried for necessary data to instantiate a Page for that path. The Page is added to the pages map of PageManager.

**Page**

**GetComments**(string sortedBy, bool ascending): communication.Comment

*Input*: A string representing the field to sort by and a Boolean representing whether to sort ascending or descending.

*Returns*: The cachedComments on the Page are iterated through and sorted in the appropriate order. They are returned as an array.

**addMemberToPage**(\*UserControllerInterface user)

*Input*: A pointer to a UserControllerInterface

*Output*: The UserControllerInterface is added the members map on the page.

**removeMemberFromPage**(\*UserControllerInterface user)

*Input*: A pointer to a UserControllerInterface

*Output*: The UserControllerInterface is removed from the members map on the page.

**addGuestToPage**(\*UserControllerInterface user)

*Input*: A pointer to a UserControllerInterface

*Output*: The UserControllerInterface is added to the guests map on the page.

**removeGuestFromPage**(\*UserControllerInterface user)

*Input*: A pointer to a UserControllerInterface

*Output*: The UserControllerInterface is removed from the guests map on the page.

**Moderate**(\*communication.Moderate, \*Server)

*Input*: A pointer to a communication.Moderate entity and a pointer to Server.

*Output*: A new record is inserted into the ModerationActions table. The CachedComment on the page is updated. Comment changes are pushed to all users viewing that Page. The underyling comment data is updated in the database.

**CreateComment**(\*communication.CommentReply, \*Server server)

*Input*: A pointer to a communication.CommentReply entity and a pointer to Server.

*Output*: A new record is inserted into the Comments table. A new CachedComment is instantiated from that data and added the comments map of the page. Comment changes are pushed to the nextResponse field of all users viewing that page.

**VoteComment**(\*communication.CommentVote, \*Server server)

*Input*: A pointer to a communication.CommentVote entity and a pointer to Server.

*Output*: Vote is called on the CachedComment associated with the communication.CommentVote entity.

**CachedComment**

**Vote**(\*communication.CommentVote, \*Server server)

*Input*: A pointer to a communication.CommentVote entity and a pointer to Server.

*Output*: A new entry is inserted into the CommentVotes table and a new CachedCommentVote is instantiated and added to the votes map of the CachedComment.

**getDataForGuest**(): communication.Comment

*Input*: None

*Output*: The cached votes data is aggregated and converted into a communication.Comment.

*Returns*: A communication.Comment

**getDataForUser**(int64 userId) : communication.Comment

*Input*: A number representing a user ID

*Output*: The cached votes data is aggregated into communication.CommentVotes. The userID allows population of the “alreadyVoted” field.

*Returns*: A communication.Comment

**Server**

**New**(): \*Server

*Input*: None

*Output*: A new Server is instantiated.

**setupRouter**()

*Input*: None

*Output*: Routing is set up and middleware is attached.

**Start**()

*Input*: None

*Output*: The server begins listening on the port configured in the .env file.

**MiddlewareAttachController**(handler http.Handler): http.Handler

*Input*: A handler function, which is one that takes a pointer to an http.Request and an http.ResponseWriter as parameters.

*Output*: Middleware to extract the token, instantiate a controller, and attach it to the http.Request.Context wraps the parameter function. The new wrapping function is returned.

***<Server API endpoint functions>***(\*http.Request, http.ResponseWriter)

*Note*: The many endpoint functions are compressed here for the sake of brevity.

*Input*: A pointer to an http.Request and an http.ResponseWriter.

*Output*: The Server retrieves the controller from the http.Request. It extracts the relevant communication entity from the http.Request.body. It passes that entity and a reference to itself to the associated handler method of the extracted controller.\

#### Package database

**Store**

**New**(): Store

*Input*: None

*Output*: Instantiates a new Store and returns it.

**Connect**()

*Input*: None

*Output*: Uses environment variables configured in a secret .env file to connect to the Postgres server on another port.

**Disconnect**()

*Input*: None

*Output*: Disconnects from the Postgres instance.

**Queries**

**CreateDomainModeratorAssignment**(context.Context, CreateDomainModeratorAssignmentParams)

*Input*: A CreateDomainModeratorAssignmentParams object.

*Output*: Inserts a new record into the DomainModeratorAssignments table.

**CreateGlobalModeratorAssignment**(context Context, CreateGlobalModeratorAssignmentParams)

*Input*: A CreateGlobalModeratorAssignmentParams object.

*Output*: Inserts a new record into the GlobalModeratorAssignments table.

**CreateDomainBanRecord**(context Context, CreateDomainBanRecordParams)

*Input*: A CreateDomainBanRecordParams object.

*Output*: Inserts a new record into the BanRecords table.

**UpdateUserBanStatus**(ctx context.Context, UpdateUserBanStatusParams)

*Input*: A UpdateUserBanStatusParams object.

*Output*: Updates the users banned status in the Users table.

**UpdateUserEmail**(ctx context.Context, UpdateUserEmailParams)

*Input*: A UpdateUserEmailParams object.

*Output*: Updates the users email in the Users table.

**UpdateFeedbackHidden**(context.Context, UpdateFeedbackHiddenParams),

*Input*: An UpdateFeedbackHiddenParams object.

*Output*: Updates an entry in the Feedbacks table to set the field “hidden” to true or false.

**UpdateUserBlurb**(context.Context, UpdateUserBlurbParams),

*Input*: An UpdateUserBlurbParams object.

*Output*: Updates and set’s the user’s profile blurb to a specified value by the user.

**CreateComment**(context.Context, CreateCommentParams),

*Input*: A CreateCommentParams object.

*Output*: Inserts a comment into the Comments table.

**CreateCommentVote**(context.Context, CreateCommentVoteParams),

*Input*: A CreateCommentVoteParams object.

*Output*: Inserts a vote into the VoteRecords table.

**UpdateCommentVote**(context.Context, UpdateCommentVoteParams)

*Input*: An UpdateCommentVoteParams object.

*Output*: Updates a record in the VoteRecords table.

**DeleteCommentVote**(context.Context, DeleteCommentVoteParams)

*Input*: A DeleteCommentVoteParams object.

*Output*: Removes a record from the VoteRecords table.

**CreateFeedback**(context.Context, CreateFeedbackParams)

*Input*: A CreateFeedbackParams object

*Output*: Inserts a new record into the Feedbacks table.

**GetCommentsAtPath**(context.Context, int64 pathID): []GetCommentsAtPathRow

*Input*: An ID associated with an entry in the Paths table.

*Output*: Returns records from the Comment table.

*Returns*: An array of GetCommentsAtPathRow objects.

**GetCommentVotes**(context.Context, int64 commentID): []GetCommentVotesRow,

*Input*: An ID associated with an entry in the Comments table.

*Output*: Returns records from the VoteRecords table.

*Returns*: An array of GetCommentVotesRow objects.

**GetUserByUserID**(context.Context, int64 id): User

*Input*: An ID associated with an entry in the Users table.

*Output*: Returns records from the Users table.

*Returns*: An Entry of GetCommentVotesRow objects.

**CreateModerationRecord**(context.Context, CreateModerationRecordParams)

*Input*: An CreateModerationRecordParams object.

*Output*: Creates a record in the BanActions table.

**UpdateCommentHidden**(context.Context, UpdateCommentHiddenParams)

*Input*: An UpdateCommentHiddenParams object.

*Output*: Updates a record in the Comments table.

**UpdateCommentRemove**(context.Context, UpdateCommentRemoveParams)

*Input*: An UpdateCommentRemoveParams object.

*Output*: Updates a record in the Comments table.

**CreateLog**(context.Context, CreateLogParams)

*Input*: A CreateLogParams object.

*Output*: Creates a record in the Logs table.

**GetDomainModeratorAssignments**(context.Context, int64 id)

*Input*: An ID associated with an entry in the DomainModeratorAssignments table.

*Output*: Returns records from the DomainModeratorAssignments table.

**GetGlobalModeratorAssignments**(context.Context, int64 id)

*Input*: An ID associated with an entry in the GlobalModeratorAssignments table.

*Output*: Returns records from the GlobalModeratorAssignments table.

**GetAdminAssignments**(context.Context, int64 id)

*Input*: An ID associated with an entry in the AdminAssignments table.

*Output*: Returns records from the AdminAssignments table.

**GetUserByUsername**(context.Context, string username) : User

*Input*: An username associated with an entry in the Users table.

*Output*: Returns records from the Users table.

**UpdateUserLastLogin**(context.Context, int64 id)

*Input*: An ID associated with an entry in the Users table.

*Output*: Updates a record in the Users table.

**CreateCommentReport**(context.Context, CreateCommentReportParams)

*Input*: A CreateCommentReportParams object.

*Output*: Creates a record in the CommentReports table.

**CreateUser**(context.Context, CreateUserParams)

*Input*: A CreateUserParams object.

*Output*: Creates a record in the Users table.

**DeleteUser**(context.Context, int64 id)

*Input*: An ID associated with an entry in the Users table.

*Output*: Remove a record in the Users table.

**GetUserByUsername**(context.Context, string username): User

An Username associated with an entry in the Users table.

*Output*: Returns a record in the Users table.

**GetUserByEmail**(context.Context, string email): User

An Email associated with an entry in the Users table.

*Output*: Returns a record in the Users table.

**UpdateUserPassword**(context.Context, UpdateUserPasswordParams)

*Input*: A CreateUserParams object.

*Output*: Updates a record in the Users table.

**UpdateUserVerification**(context.Context, UpdateUserVerificationParams)

*Input*: A UpdateUserVerificationParams object.

*Output*: Updates a record in the Users table.

**CreateVerificationRecord**(context.Context, CreateVerificationRecordParams)

*Input*: A CreateVerificationRecordParams object.

*Output*: Creates a record in the VerificationCodes table.

**GetVerificationRecord**(context.Context, int64 userID): []VerificationCode

*Input*:An userID associated with an entry in the VerificationCodes table.

*Output*: Returns a record in the VerificationCodes table.

*Returns*: An Entry of VerificationCode objects.

**DeleteVerificationRecords**(context.Context, int64 userID)

*Input*: An userID associated with an entry in the VerificationCodes table.

*Output*: Removes a record in the VerificationCodes table.

**GetPWResetRecord**(context.Context, int64 userID): []PasswordResetCode

*Input*: An userID associated with an entry in the PasswordResetCodes table.

*Output*: Returns a record in the PasswordResetCodes table.

*Returns*: An Entry of PasswordResetCode objects.

**CreatePWResetRecord**(context.Context, CreatePWResetRecordParams)

*Input*: A CreatePWResetRecordParams object.

*Output*: Creates a record in the PasswordResetCodes table.

**DeletePWResetRecords**(context.Context, int64 userID)

*Input*: An userID associated with an entry in the PasswordResetCodes table.

*Output*: Creates a record in the VerificationCodes table.

**GetBanRecords**(context.Context): []GetBanRecordsRow

*Output*: Returns a record in the Users table.

**GetCommentReports**(context.Context, Boolean actionTaken): []GetCommentReportsRow

*Input*: An actionTaken associated with an entry in the CommentReports table.

*Output*: Returns a record in the CommentReports table.

*Returns*: An array of GetCommentReportsRow objects.

**GetFeedback**(context.Context, Boolean hidden) []GetFeedbackRow

*Input*: An actionTaken associated with an entry in the Feedbacks table.

*Output*: Returns a record in the Feedbacks table.

*Returns*: An array of GetFeedbackRow objects.

**GetLogsForDateRange**(context.Context, GetLogsForDateRangeParams): []GetLogsForDateRangeRow

*Input*: A GetLogsForDateRangeParams object.

*Output*: Returns a record in the Logs table.

*Returns*: An array of Logs objects.

**GetModRecordsForModerator**(context.Context, int64 id): []GetModRecordsForModeratorRow

*Input*: An ID associated with an entry in the CommentModeratorActions table.

*Output*: Returns a record in the CommentModeratorActions table.

**GetDomainModerators**(context.Context, string domain): []GetDomainModeratorsRow

*Input*: An domain associated with an entry in the DomainModeratorAssignments table.

*Output*: Returns a record of the DomainModeratorAssignments table.

*Returns*: An array of GetDomainModeratorsRow objects.

**GetGlobalModerators**(context.Context): []GetGlobalModeratorsRow

*Input*: An domain associated with an entry in the GlobalModeratorAssignments table.

*Output*: Returns a record of the GlobalModeratorAssignments table.

*Returns*: An array of GetGlobalModeratorsRow objects.

**GetAdmins**(context.Context): []GetAdminsRow

*Input*: An domain associated with an entry in the AdminAssignments table.

*Output*: Returns a record of the AdminAssignments table.

*Returns*: An array of GetAdminsRow objects.

**GetNewestUser**(context.Context): User

*Output*: Returns a record of the Users table.

*Returns*: An Entry of an Users object.

**GetUserCount**(context.Context): int64

*Output*: Returns a record of the Users table.

*Returns*: An Entry of a Users object.

**GetLogsForIP**(context.Context, string IP): []GetLogsForIPRow

*Input*: An IP associated with an entry in the Logs table.

*Output*: Returns a record of the Logs table.

*Returns*: An Entry of a Logs object.

**GetLogsForUser**(context.Context, int64 userID): []GetLogsForUserRow

*Input*: An IP associated with an entry in the Logs table.

*Output*: Returns a record of the Logs table.

*Returns*: An Entry of a Logs object.

#### Front End

### Narrative/PDL

1. Start screen
   1. Log in
   2. Create a account
2. Log in
   * 1. Enter username
     2. Enter password
     3. Submit. Are the username and password correct? Is the user a moderator account? Is the user an administrator account?
        1. Yes. go to main screen.
        2. No. start Log in again.
        3. Yes. go to Moderator main screen.
        4. No. go to next step.
        5. Yes. go to Administrator main screen.
        6. No. go to main screen.
   1. Reset password
3. Reset password
   * 1. Enter new username
     2. Enter new password
     3. Submit. Does the user confirm the reset is them ?
        1. Yes. go to Log in.
        2. No. go to reset password.
4. Create an account
   * 1. Enter an email.
     2. Enter a username.
     3. Enter a password.
     4. Submit. Is there an email and a password? Is the email valid?
        1. Yes. check if the email is valid.
        2. No. go back to create an account and prompt for email username and password.
        3. Yes. Create the account and go to the main screen.
        4. No. Prompt for a new email go to create an account.
5. Main screen
   1. Rate a comment
   2. Make a comment
   3. Report a comment
   4. Log out
6. Rate a comment
   * 1. Rate the comment. as funny? As factual? As informative?
        1. Yes. Add 1 to the total of funny ratings and go to next step
        2. No. go to next step.
        3. Yes. Add 1 to the total of factual ratings and go to next step
        4. No. go to next step.
        5. Yes. Add 1 to the total of informative ratings and go to next step
        6. No. do nothing go to Main screen
     2. Main screen. Is the user a moderator? Is the user an Administrator?
        1. Yes. go to Moderator main screen
        2. No. go to next step
        3. Yes. go to Administrator main screen.
        4. No. go to Main screen.
7. Make a comment
   * 1. Type in the comment.
     2. Submit the comment. Does the comment contain disallowed or spam content?
        1. Yes. Do not submit the comment and go to Make a comment.
        2. No. Submit the comment to the database.
     3. Main screen. Is the user a moderator? Is the user an Administrator?
        1. Yes. go to Moderator main screen
        2. No. go to next step
        3. Yes. go to Administrator main screen.
        4. No. go to Main screen.
8. Report a comment
   * 1. Type why the user is reporting the comment.
     2. Report the comment to the moderators and administrators
     3. Main screen. Is the user a moderator? Is the user an Administrator?
        1. Yes. go to Moderator main screen
        2. No. go to next step
        3. Yes. go to Administrator main screen.
        4. No. go to Main screen.
9. Log out
   1. Start screen
10. Moderator main screen
    1. Rate a comment
    2. Make a comment
    3. Report a comment
    4. Ban a comment
    5. Ban a user
    6. Log out
11. Administrator main screen
    1. Rate a comment
    2. Make a comment
    3. Report a comment
    4. Report a user
    5. Ban a comment
    6. Ban a user
    7. Log out
12. Ban a comment
    * 1. Reason for removing the comment.
      2. Removing the comment. Is the user a moderator?
         1. Yes. Go to Moderator main screen.
         2. No. Go to Administrator main screen.
13. Ban a user
    * 1. Reason for banning the user. Is the user being banned an administrator by a moderator?
         1. Yes. Do not ban the administrator and go back to Moderator main screen.
         2. No. Ban the user go to Moderator main screen.

### Implementation Tools

* Backend in Go
* Frontend in Typescript
  + Usage of Extension Frameworks

## Implementation Timeline

1. Set up the Git repository, readme, and directory structure.
2. Create the necessary compilation files, such as the docker file, makefile, package.json, tsconfig.json, and webpack.config.js.
3. Stub all the go functions for the back end. No actual code is written, only the signatures across the board. It’s refined until no errors are appearing. Copy the text from the design document into comments on the code.
4. Stub all functions for the front end in the same way. (Can be simultaneous to back end)
5. Starting with the basic “spinning-up” of the Server and database
6. Implement one full message chain and ensure that the front end can indeed communicate with the back end.
7. Begin writing the methods, simultaneously writing a parallel testMethod. Unit test each message and integration test each method chain until it is working properly.
8. Repeat for every method until the code is done.

## 

## Testing

The team utilized Trello, a task tracking tool. The construction of the document was divided up into smaller, individual tasks. Each member of the team was assigned specific tasks to work on independently, moving each item into the appropriate category on the board. When an item was moved into the “Review” category, each other team member would review the work and offer feedback. This review process constituted Design testing. After implementing any feedback, final approval is required by the team. This process was used to ensure all tasks were completed in a timely manner and reviewed by the entire team.

# Appendix

## Appendix A: Team Details

## 

## Appendix B: Workflow Authentication

## 

## Appendix C: Writing Center Report