**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.

**Class Name**: UserControllerInterface

**Class 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.

**Class Data Members**: None

**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)

**Class Name**: 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

**Class Name**: 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*

**Class Name**: 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*

**Class Name**: 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*

**Class Name**: 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*

**Class Name**: 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*

**Class Name**: 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*

**Class Name**: 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(string username, string password, \*Server server): UserControllerInterface, Logout(UserControllerInterface, \*Server server), Register(UserControllerInterface, \*Server server), GetMemberController(int64 id): UserControllerInterface, GetGuestController(int64 id): UserControllerInterface, DispatchPasswordResetEmail(UserControllerInterface, \*Server server)

**Class Name**: 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)

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

**Class Name**: 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)

**Class Name**: 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

**Class Methods**: …

**Class Name**: 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)

**Package Method**: keyfunc(\*jwt.Token token)

**Package Method Description**: Used while interfacing with the JWT library to confirm the signing method of a token. It returns the secret key for parsing.

**Returns**: The secret JWT key as a byte string.

**Package Method:** GetToken(int64 userId)

**Package Method Description:** Returns a JWT token signed with the secret key with a claim of an expiration time, associated with a userID. It does not perform validation.

**Returns**: A JWT token, as a string.

Example Usage

// 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 comment 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 Server – Functional Descriptions

**UserControllerInterface**

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

*Input:*

A client-server communication entity, communication.Ban and a pointer to Server.

*Output:*

If the controller is an Admin or Moderator Controller, and the client is bannable, a new record is created in the database and the target user’s banned field may be changed.

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

*Input:*

A client-server communication entity, 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 is\_verified is set to false until the new email is verified.

HandleCommandChangeFeedback(\*communication.ChangeFeedback),

Input:

A client-server communication entity, 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 client-server communication entity, 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 client-server communication entity, communication.ChangeProfileBlurb, 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 client-server communication entity, communication.CommentReply, and a pointer to Server.

Output:

If the controller is a Member controller, a new comment is added to the database and a new instance of CachedComment is instantiated at the appropriate page. The new comment data is added to the nextResponse field of every controller currently viewing that page.

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

Input:

A client-server communication entity, communication.CommentVote and a pointer to Server.

Output:

If the controller is a Member controller, a new comment vote is added to the database (or removed from the database, if the member votes in the opposite direction from how they previously voted). A new instance of CachedCommentVote is instantiated at the appropriate page. A Server-client communication entity, The new vote data is added to the nextResponse field of every controller currently viewing that page.

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

Input: A client-server communication entity, 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 client-server communication entity, 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. The CachedComments data for that page are added to the nextResponse field for the controller.

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

Input: A client-server communication entity, communication.GetUserProfile, and a pointer to Server.

Output:

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

HandleCommandLogin(\*Server),

Input: A pointer to Server.

Output:

The user controller, which has already been instantiated by the UserManager, adds the server-client communication entity LoginResponse to the nextResponse field for the controller.

HandleCommandLogout(\*Server)

Input: A pointer to the server.

Output:

The server-client communication entity LogoutResponse is added to the nextResponse field for the controller.

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

Input: A client-server communication entity, communication.Moderate and a pointer to Server.

Output:

If the controller has appropriate access, a moderation record is inserted into the database and the comment is altered to be ‘hidden’ or ‘removed’. All controllers on that page will have the server-communication entity communication.Comment added to their nextResponse field so they can realize the comment change on their front ends the next time they communicate with the 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)