**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(\*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)

**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(\*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)

**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), dataForUser(int64 userId) : communication.Comment

**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 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), getLoginStatus(\*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 pointer to communication.Ban, a client-server communication entity, 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, a client-server communication entity, 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 pointer to communication.ChangeFeedback, a client-server communication entity, 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, a client-server communication entity, 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. ChangeProfileBlurb, a client-server communication entity, 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, a client-server communication entity, 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, a client-server communication entity, 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, a client-server communication entity, 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, a client-server communication entity, 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, a client-server communication entity, 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, a client-server communication entity, 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 client-server communication entity, 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 client-server communication entity, 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 client-server communication entity, 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 client-server communication entity, communication.RequestVerification, and a pointer to Server.

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

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

Input: A client-server communication entity, 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 client-server communication entity, 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 client-server communication entity, 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 client-server communication entity, 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 client-server communication entity, 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 client-server communication entity, 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 username and a password.

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)

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.