```java
package Models;

import Controllers.DBSupport;

/**
 * Model for the Message Table. Essentially this is what the table will contain
 * @Author Dylan Mrzlak
 */
public class Message {

    private Integer id;

    private Integer senderId;

    private Integer wId;

    private Integer cID;

    private Integer recipientID;

    private String content;

    private Boolean pinned;

    public Message(Integer id, Integer senderId, Integer wId, Integer cID, Integer recipientID, String content, Boolean pinned) {
        this.id = id;
        this.senderId = senderId;
        this.wId = wId;
        this.cID = cID;
        this.recipientID = recipientID;
        this.content = content;
        this.pinned = pinned;
    }

    //dm
    public Message(Integer id, Integer senderId, Integer recipientID, String content) {
        this.id = id;
        this.senderId = senderId;
        this.recipientID = recipientID;
        this.content = content;
    }

    //public message
    public Message(Integer id, Integer senderId, Integer wId, Integer cID, String content, Boolean pinned) {
        this.id = id;
        this.senderId = senderId;
        this.wId = wId;
        this.cID = cID;
        this.content = content;
        this.pinned = pinned;
    }

    public static DBSupport.HTTPResponse getAllMessages(String name) {
        DBSupport.HTTPResponse res = DBSupport.getAllMessages(name);
        return res;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getwId() {
        return wId;
    }

    public Integer getSenderId() {
        return senderId;
    }

    public void setSenderId(Integer senderId) {
        this.senderId = senderId;
    }

    public void setwId(Integer wId) {
        this.wId = wId;
    }

    public Integer getcID() {
        return cID;
    }

    public void setcID(Integer cID) {
        this.cID = cID;
    }

    public Integer getRecipientID() {
        return recipientID;
    }

    public void setRecipientID(Integer recipientID) {
        this.recipientID = recipientID;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Boolean getPinned() {
        return pinned;
    }

    public void setPinned(Boolean pinned) {
        this.pinned = pinned;
    }

    /**
     * Calls DBSupport and returns the response
     * @param senderName
     * @param workspaceName
     * @param channelName
     * @param message
     * @return
     */
    public static DBSupport.HTTPResponse sendMessage(String senderName, String workspaceName, String channelName, String message){
            DBSupport.HTTPResponse res = DBSupport.sendMessage(senderName, workspaceName, channelName, message);
            return res;

    }

    /**
     * Calls DBSupport and returns the response
     * @param senderName
     * @param receiver
     * @param message
     * @return
     */
    public static DBSupport.HTTPResponse sendDirectMessage(String senderName, String receiver, String message){
        DBSupport.HTTPResponse res = DBSupport.sendDirectMessage(senderName, receiver, message);
        return res;
    }
}
```

```java
package Models;

import Controllers.DBSupport;

/**
 * Model for the channel, holds the same data that the server would and contains the static calls to the dbsupport
 */
public class Channel {
    private Integer id;

    private Integer wId;

    private String name;

    /**
     * Calls DBSupport and returns the response
     * @param workspaceName
     * @param name
     * @return
     */
    public static DBSupport.HTTPResponse createChannel(String workspaceName, String name) {
        DBSupport.HTTPResponse res = DBSupport.createChannel(workspaceName, name);
        return res;
    }

    /**
     * Calls DBSupport and returns the response
     * @param userName
     * @param workspaceName
     * @param channelName
     * @return
     */
    public static DBSupport.HTTPResponse viewMentions(String userName, String workspaceName, String channelName) {
        DBSupport.HTTPResponse res = DBSupport.viewMentions(userName, workspaceName, channelName);
        return res;
    }

    /**
     * Calls DBSupport and returns the response
     * @param cId
     * @return
     */
    public static DBSupport.HTTPResponse getChannelName(int cId) {
        DBSupport.HTTPResponse res = DBSupport.getChannelName(cId);
        return res;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getwId() {
        return wId;
    }

    public void setwId(Integer wId) {
        this.wId = wId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Channel(String name, int wID){
        this.name = name;
        this.id = -1;
        this.wId = wID;
    }

    public Channel(String name, int wID, int id){
        this.name = name;
        this.id = id;
        this.wId = wID;

    }


}
```

```java
package Models;

import Controllers.DBSupport;
/**
 * Model for the user, holds the same data that the server would and contains the static calls to the dbsupport
 */
public class User {

    private String name;
    private String password;
    private Integer userId;

    public User(String uName, String uPassword){
        name = uName;
        password = uPassword;
        userId = -1;
    }

    public User(String uName, String uPassword, Integer uId){
        name = uName;
        password = uPassword;
        userId = uId;
    }

    /**
     * Calls DBSupport and returns the response
     * @param username
     * @param password
     * @return
     */
    public static DBSupport.HTTPResponse signIn(String username, String password) {
        DBSupport.HTTPResponse res = DBSupport.signin(username, password);
        return res;
    }

    /**
     * Calls DBSupport and returns the response
     * @param senderId
     * @return
     */
    public static DBSupport.HTTPResponse getUserNameByID(Integer senderId) {
        DBSupport.HTTPResponse res = DBSupport.getUserNameByID(senderId);
        return res;
    }

    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String username) {
        password = username;
    }


    public static DBSupport.HTTPResponse searchUser(String name){
        DBSupport.HTTPResponse res = DBSupport.searchUser(name);
        return res;
    }

    public static DBSupport.HTTPResponse createUser(String name, String password){
        DBSupport.HTTPResponse res = DBSupport.createUser(name, password);
        return res;
    }

}
```

```java
package Models;

import Controllers.DBSupport;

/**
 * Model for the Message Table. Essentially this is what the table will contain
 * @Author Dylan Mrzlak
 */
public class Textfile {

    private Integer id;

    private String name;

    private String content;


    public Message(Integer id,String name String content) {
        this.id = id;
        this.name = name;
        this.content = content;
     }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }

    public void setName(String content) {
        this.name = name;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    /**
     * Calls DBSupport and returns the response
     * @param message
     * @return
     */
    public static DBSupport.HTTPResponse sendText(String name, String message){
        DBSupport.HTTPResponse res = DBSupport.sendMessage(name, message);
        return res;

    }

    public static DBSupport.HTTPResponse getText(String name){
        DBSupport.HTTPResponse res = DBSupport.sendMessage(name);
        return res;

    }

}
```

```java
package Models;

import Controllers.DBSupport;
/**
 * Model for the Workspace within the front end. Will contain the data and methods required of the workspace
 * @Author Dylan Mrzlak
 */
public class Workspace {

    private String name;
    private int id;

    /**
     * Basic Constructor. Since the id is not known until it is in the DB, we can only instantiate the name
     * @param name
     * @Author Dylan Mrzlak
     */
    public Workspace(String name){
       this.name = name;
       this.id = -1;
    }

    public Workspace(String name, int id){
        this.name = name;
        this.id = id;
    }



    public String getName(){
        return name;
    }

    int getwId(){
        return id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    /**
     * Create a workspace and call for the DBSUpport to request it put into the DB
     * @Author Dylan Mrzlak
     */
    public static DBSupport.HTTPResponse createWorkspace(String name){
        DBSupport.HTTPResponse res = DBSupport.putWorkspace(name);
        return res;
    }

    public static DBSupport.HTTPResponse searchWorkspace(String name){
        DBSupport.HTTPResponse res = DBSupport.searchWorkspace(name);
        return res;
    }
    /**
     * Calls DBSupport and returns the response
     * @param name
     * @param username
     * @return
     */
    public static DBSupport.HTTPResponse joinWorkspace(String name, String username){
        DBSupport.HTTPResponse res = DBSupport.joinWorkspace(name, username);
        return res;
    }

    /**
     * Calls DBSupport and returns the response
     * @param mId
     * @return
     */
    public static DBSupport.HTTPResponse pinMessage(String mId) {
        DBSupport.HTTPResponse res = DBSupport.pinMessage(Integer.parseInt(mId));
        return res;
    }

    /**
     * Calls DBSupport and returns the response
     * @param workspaceName
     * @return
     */
    public static DBSupport.HTTPResponse getUsersInWorkspace(String workspaceName){
        DBSupport.HTTPResponse res = DBSupport.viewUsers(workspaceName);
        return res;
    }
}
```

```java
package Controllers;

import Models.Message;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.*;
import java.io.IOException;
import java.net.MulticastSocket;
import java.net.URI;
import java.net.URISyntaxException;

/**
 * This is our Data Provider for the frontend app. THis will cal to the back end controllers via HTTPRequests.
 * The idea goes:
 *      User --> InputController --> Model --> DBSupport --> HTTPRequest --> ModelController --> DB
 *      User <-- InputController <-- Model <-- DBSupport <-- HTTPRequest <-- ModelController <--
 *   (Interface)
 * @Author Dylan Mrzlak
 */
public class DBSupport {


    /**
     * Connect to the backend via HTTPRequest. The controllers on the backend have specific URL mappings,
     * so we can use those to proc the backend to do its work
     *
     * @param url
     * @return
     * @throws URISyntaxException
     * @throws IOException
     * @throws InterruptedException
     * @Author Dylan Mrzlak
     */
    public static HTTPResponse serverRequest(String url) throws URISyntaxException, IOException, InterruptedException {
        //Build the URL out of the built string
        URL uri = new URL(url);

        //Open a connection to uri, we will soon be able to make the actual request
        //Also instantiate the rest of the request
        HttpURLConnection con = (HttpURLConnection) uri.openConnection();
        //By only using GET we can simplify calls on either side. The framework we're using, spring boot doesn't care
        //musch about the actual request method
        con.setRequestMethod("GET");
        //We want a json to be returned in the event htat we get an object returned from the controller.
        //They don't really send Objects, but rather a string style of encoding called a json.
        //These are really simple enough to understand when looking at the JSONString
//        con.setRequestProperty("Content-Type", "application/json");

        con.setRequestProperty("Content-Type", "application/xml");
        String contentType = con.getHeaderField("Content-Type");

        //We want to know if we did good, or if Big Backend is mad at us
        int status = con.getResponseCode();
        Reader streamReader = null;
        //When the status is 300 or over, that means the server is not happy to some degree about the request
        //Statuses usually go as follow
        //100 ok-ish?
        //200 Yay we're all good
        //300 Not so great
        //400 You messed up
        //500 Mr. Stark, I don't feel so good
        if (status > 299) {
            //We want to be able to read the repsonse as an Error
            streamReader = new InputStreamReader(con.getErrorStream());
        } else {
            //We want to be able to actually read the response
            streamReader = new InputStreamReader(con.getInputStream());
        }
        //Read the response, what ever it is
        BufferedReader in = new BufferedReader(streamReader);
        String inputLine;
        StringBuffer content = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        streamReader.close();
        in.close();
        con.disconnect();
        //Return the response rebuilt into the HTTPResponse built here
        return new HTTPResponse(status, content.toString());
    }

    //Prints an error for stating the a request couldn't finish for whatever reason.
    // Helps keep the app from breaking completely
    public static String handleErr() {
        System.out.println("Unable to handle the request, please check your connection, try again");
        return null;
    }

    /**
     * Creates a request to the backend to make a Workspace
     * @param name
     * @return
     * @Author Dylan Mrzlak
     */
    public static HTTPResponse putWorkspace(String name) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.createWorkspace(name));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }


    /**
     * Builds the request to join a workspace
     * @param workspaceName
     * @param name
     * @return
     */
    public static HTTPResponse joinWorkspace(String workspaceName, String name) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.joinWorkspace(workspaceName, name));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }


    /**
     * Builds a request to create a user
     * @param name
     * @param password
     * @return
     */
    public static HTTPResponse createUser(String name, String password) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.createUser(name, password));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * Sets a message as pinned
     * @param id
     * @return
     * @Author Joseph Hudson
     */
    public static HTTPResponse pinMessage(Integer id) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.pinMessage(id));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * Builds a request to send a message
     * @param senderName
     * @param workspaceName
     * @param channelName
     * @param message
     * @return
     */
    public static HTTPResponse sendMessage(String senderName, String workspaceName, String channelName, String message) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.sendMessage(senderName, workspaceName, channelName, message));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * Builds a request to create a channel
     * @param workspaceName
     * @param name
     * @return
     */
    public static HTTPResponse createChannel(String workspaceName, String name) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.addNewChannel(workspaceName, name));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * Builds a request to send a DM
     * @param senderName
     * @param receiver
     * @param message
     * @return
     */
    public static HTTPResponse sendDirectMessage(String senderName, String receiver, String message) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.sendDirectMessage(senderName, receiver, message));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * Builds a request to view users in a workspace
     * @param workspaceName
     * @return
     */
    public static HTTPResponse viewUsers(String workspaceName) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.getUsersInWorkspace(workspaceName));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * Get all the mentions within a channel for a given user
     * @param username
     * @param workspaceName
     * @param channelName
     * @return Status code of the HTTP call and a response string (either a JSON or a string)
     */
    public static HTTPResponse viewMentions(String username, String workspaceName, String channelName) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.viewMentions(username, workspaceName, channelName));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * Get all the messages within a workspace
     * @param workspaceName
     * @return Status code of the HTTP call and a response string (either a JSON or a string)
     *          The Json is a list of messages all grouped by channel
     */
    public static HTTPResponse getAllMessages(String workspaceName) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.getAllMessages(workspaceName));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * searcher workspace
     * @param workspaceName
     * @return Status code of the HTTP call and a json list of the workspaces
     */
    public static HTTPResponse searchWorkspace(String workspaceName) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.searchWorkspace(workspaceName));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }
    /**
     * searcher workspace
     * @param workspaceName
     * @return Status code of the HTTP call and a json list of the workspaces
     */
    public static HTTPResponse searchUser(String userName) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.searchUser(userName));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }
    /**
     * Builds a request to get a channel name
     * @param cId
     * @return
     */
    public static HTTPResponse getChannelName(int cId) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.getChannelName(cId));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * Builds a request to get a user by id
     * @param senderId
     * @return
     */
    public static HTTPResponse getUserNameByID(Integer senderId){
        try {
            HTTPResponse response = serverRequest(ParamBuilder.getUserNameById(senderId));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }
    }

    /**
     * Builds a request to sign in a user
     * @param username
     * @param password
     * @return
     */
    public static HTTPResponse signin(String username, String password) {
        try {
            HTTPResponse response = serverRequest(ParamBuilder.signin(username, password));
            return response;
        } catch (Exception e) {
            return new HTTPResponse(406, handleErr());
        }   }

    /**
     * Model for the HTPPResponse rebuilding, that way the objects can handle the data themselve
     * @author Dylan Mrzlak
     */
    public static class HTTPResponse{
        public int code;
        public String response;

        HTTPResponse(int status, String content){
            code = status;
            response = content;
        }
    }

    /**
     * Static class to build our URL's to Strings.
     * Makes it a lot better to send it out to here, rather than build them in other methods
     */
    private static class ParamBuilder{
        //This is the base url for our server. When we get a dedicated server for the app, we will want this changed
        private static String BASE_URL = "http://localhost:8080/";

        //Below are the Builders for the URL mappings
        //URLs are as follows:
        //      BASE_URL + CONTROLLER_MAPPING + / + REQUESTMAPPING + ?PARAM1_NAME=PARAM1
        //For 2+ params:
        //      BASE_URL + CONTROLLER_MAPPING + / + REQUESTMAPPING + ?PARAM1_NAME=PARAM1&PARAM2_NAME=PARAM2....

        public static String sendDirectMessage(String sender, String reciever, String message){
            return BASE_URL+"/message/directMessage?senderName="+sender+"&recieverName="+reciever+"&message="+message;
        }
        public static String sendMessage(String sender, String workspace, String channelName, String message){
            return BASE_URL+"/message/channelMessage?senderName="+sender+"&workSpaceName="+workspace+"&channelName="+channelName+"&message="+message;
        }

        public static String sendText(String name, String content){
            return BASE_URL+"/textfile/send?name="+name+"&content="+content;
        }

        public static String getText(String name){
            return BASE_URL+"/textfile/download?name"+name;
        }

        public static String createWorkspace(String name){
            return BASE_URL+"workspace/add?name="+name;
        }

        public static String joinWorkspace(String workspaceName, String username){
            return BASE_URL+"user/join?workspaceName="+workspaceName+"&name="+username;
        }

        public static String createUser(String name, String password){
            return BASE_URL+"user/add?username="+name+"&password="+password;
        }

        public static String pinMessage(int mId){
            return BASE_URL+"message/pinMessage?messageID=" + mId;
        }

        public static String getUsersInWorkspace(String workspaceName) {
            return BASE_URL+"workspace/getUsers/?name="+workspaceName;
        }
        public static String searchWorkspace(String workspaceName) {
            return BASE_URL+"workspace/search?name="+workspaceName;
        }
        public static String searchUser(String userName) {
            return BASE_URL+"user/search?name="+userName;
        }
        public static String addNewChannel(String workspaceName, String name) {
            return BASE_URL+"channel/add?workspaceName="+workspaceName+"&name="+name;
        }

        public static String viewMentions(String username, String workspaceName, String channelName) {
            return BASE_URL+"channel/viewMentions?username=" + username +
                    "&workspaceName=" + workspaceName +
                    "&channelName=" + channelName;
        }

        public static String getAllMessages(String workspaceName) {
            return BASE_URL+"workspace/getAllMessages/?workspaceName="+workspaceName;
        }

        public static String getChannelName(int cId) {
            return BASE_URL+"channel/getName?cId="+cId;
        }

        public static String getUserNameById(Integer senderId) {
            return BASE_URL+"user/getUsername?senderId="+senderId;
        }

        public static String signin(String username, String password) {
            return BASE_URL+"user/login?username="+username+"&password="+password;

        }
    }
}
```

```java
import Controllers.DBSupport;
import Models.Message;
import Models.Channel;
import Models.User;
import Models.Workspace;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;

import com.google.gson.Gson;


/**
 * This will be the main controller for the application.
 * It will take the initial input for User Input
 * and then pass it along to other classes to handle the actual functionality
 *
 * @Author Dylan Mrzlak
 * Original Framework and Use handle for CREATE_WORKSPACE and JOIN_WORKSPACE
 */
public class InputController {
    private static final String CREATE_WORKSPACE = "create workspace";
    private static final String JOIN_WORKSPACE = "join";
    private static final String CREATE_CHANNEL = "create channel";
    private static final String VIEW_USERS = "view users";
    private static final String SEND = "send";
    private static final String SEND_DM = "send to";
    private static final String ADD_USER = "create user";
    private static final String PIN_MESSAGE = "pin message";
    private static final String LOG_MESSAGES = "log messages";
    private static final String VIEW_MENTIONS = "view mentions";
    private static final String LOGIN = "login";
    private static final String HELP = "help";
    private static final String SEARCH_WORKSPACE = "search workspace";
    private static final String SEARCH_USER = "search user";
    private static final String SEND_TEXTFILE = "send textfile";
    private static final String DOWNLOAD_TEXTFILE = "download textfile";


    private static Gson gson = new Gson();
    private static User curUser = null;
    private static Workspace curWorkspace = null;
    private static Channel curChannel = null;

    public static void main(String[] args) {
        //If this line get mad, check your dependencies, may have dropped
        Scanner input = new Scanner(System.in);
        String userInput = "";

        printInstructions();
        do {
            userInput = input.nextLine();
            //By forcing commands to be in a format of COMMAND - ARGUMENT
            //We can easily manage the input and decide what is needed
            int substringBegin = userInput.indexOf('-');
            //Now that we have commands without args, we need to be able to take commands without the delimitter set
            if (substringBegin == -1) substringBegin = userInput.length();
            String command = "";
            String[] userArgs = {};
            if (userInput.length() == substringBegin) {
                command = userInput;
            } else {
                command = userInput.substring(0, substringBegin).trim();
                userArgs = userInput.substring(substringBegin + 1).trim().split(" ");
            }
            //Have updated the switch to be more readable and move into methods, rather than holding all the logic in here.
            //We were simply expanding this too much that it was becoming hard to read. This is much more followable
            switch (command) {
                case HELP:
                    printHelp();
                    break;
                case LOGIN:
                    SignIn(userArgs);
                    break;
                case ADD_USER:
                    AddUser(userArgs);
                    break;
                case CREATE_WORKSPACE:
                    CreateWorkspace(userArgs);
                    break;
                case JOIN_WORKSPACE:
                    JoinWorkspace(userArgs);
                    break;
                case CREATE_CHANNEL:
                    CreateChannel(userArgs);
                    break;
                case VIEW_USERS:
                    ViewUsers(userArgs);
                    break;
                case PIN_MESSAGE:
                    PinMessage(userArgs);
                    break;
                case SEND_DM:
                    SendDM(userArgs);
                    break;
                case SEND:
                    SendMessage(userArgs);
                    break;
                case LOG_MESSAGES:
                    LogMessage(userArgs);
                    break;
                case VIEW_MENTIONS:
                    ViewMentions(userArgs);
                    break;
                case SEARCH_USER:
                    searchUser(userArgs);
                    break;
                case SEARCH_WORKSPACE:
                    searchWorkspace(userArgs);
                    break;
                case DOWNLOAD_TEXTFILE:
                    downloadTextfile(userArgs);
                    break;
                case SEND_TEXTFILE:
                    sendTextfile(usedArgs);
                    break;
                default:
                    System.out.println("Invalid Input please try again :(");
                    break;
            }
        } while (input.hasNextLine());

    }


    private static void sendTextfile(){
        if (curUser == null) {
            System.out.println("You need to create a user or sign in to cont
inue");
            return;
        }
        if (curWorkspace == null) {
            System.out.println("User not in workspace");
            return;
        }
        if (curChannel == null) {
            System.out.println("User not in Channel;");
            return;
        }
        String filename = userArgs[0];
        filename = filename.subString(lastIndexOf('/',filname.length));

        String Content = "";
        Scanner scan = Scanner(new File(userArgs[0]));
        String temp;
        while (scan.hasNext()){
            temp = scan.nextLine();
            temp.replace(' ',"_SS_");
            temp.replace('\t',"_TT_");
            temp+="_NN_";
            temp.replace('&', "_AA_");
            temp.replace('?',"_QQ_");
            //temp.replace('',"")
            content += temp;
        }
        if(content.length > 2048) {
            content = content.subString(0, 2048);
            System.out.print("File too long shortened to send");
        }

        DBSupport.HTTPResponse Textfile.sendText(filename,content);

        if (response.code >= 300) {
            System.out.println(response.response);
        } else {
            System.out.print("file sent!");
        }

    }

    private static void downloadTextfile(){
        if (curUser == null) {
            System.out.println("You need to create a user or sign in to continue");

            return;
        }
        if (curWorkspace == null) {
            System.out.println("User not in workspace");
            return;
        }
        if (curChannel == null) {
            System.out.println("User not in Channel;");
            return;
        }
        String filename = userArgs[0];

        DBSupport.HTTPResponse response = DBSupport.HTTPResponse Textfile.getText(filename);

        if (response.code >= 300) {
            System.out.println(response.response);
        } else {
            Textfile t = gson.fromJson(response, t.class);


            String temp = t.getContent();

            temp = scan.nextLine();
            temp.replace("_SS_", ' ');
            temp.replace("_TT_", '\t');
            temp.replace("_AA_", '&');
            temp.replace("_QQ_", '?');

            String[] file = temp.split("_NN_");
            WriteFile(file,"..\\..\\files\\", t.getName());
        }

    }


    /**
     * Takes "no" arguments and will print the mentions for the current user in a channel.
     * Our mentions right now just search for a username, but can simply be expanded to be "@/USERNAME/"
     * @param userArgs
     * @author Dylan Mrzlak
     */
    private static void ViewMentions(String[] userArgs) {
        //View mentions does not need user args whatsoever, so we'll just ignore them. They are passed in for consistency
        //First we want to make sure that nothing is null (We want to be in a workspace and a channel, and then the user needs to be signed in
        if (curUser == null) {
            System.out.println("You need to create a user or sign in to continue");

            return;
        }
        if (curWorkspace == null) {
            System.out.println("User not in workspace");
            return;
        }
        if (curChannel == null) {
            System.out.println("User not in Channel;");
            return;
        }
        //Now that those are out of the way, we need to get the actual mentions
        // depending on what the server returns, we handle it accordingly. We either get a list and print it,
        //  or an error and print that
        DBSupport.HTTPResponse response = Channel.viewMentions(curUser.getName(), curWorkspace.getName(), curChannel.getName());
        if (response.code >= 300) {
            System.out.println(response.response);
        } else {
            Message[] mentions = gson.fromJson(response.response, Message[].class);
            System.out.println("These are the your mentions:");
            for (Message mention : mentions) {
                String printMention = "\t" + mention.getContent().replaceAll("_SS_", " ");
                System.out.println(printMention);
            }
        }

    }

    /**
     * Take the messages from a workspace, grouped by channel and order based on time. The write it to a file
     * @param userArgs
     * @author Dylan Mrzlak
     */
    private static void LogMessage(String[] userArgs) {
        //Logging does not need user args whatsoever, so we'll just ignore t
```

```
hem. They are passed in for consistency
        //First we want to make sure that nothing is null (We want to be in a workspace and a channel, and then the user needs to be signed in
        if (curUser == null) {
            System.out.println("You need to create a user or sign in to continue");
            return;
        }
        if (curWorkspace == null) {
            System.out.println("User not in workspace");
            return;
        }
        if (curChannel == null) {
            System.out.println("User not in Channel;");
            return;
        }
        //Now that those are out of the way, we need to get the actual messages.
        // They will be grouped by channel in the backend,
        // but we do some lifting here as well to properly make the strings we need
        //Note like with all of our methods, we can get an error or the data we want so we have to deal with it properly
        System.out.println("Getting the messages for: " + curWorkspace.getName());
        DBSupport.HTTPResponse response = Message.getAllMessages(curWorkspace.getName());
        if (response.code >= 300) {
            System.out.println(response.response);
        }
        else {
            System.out.println("Retrieval for: " + curWorkspace.getName() + " successful");
            Message[] messages = gson.fromJson(response.response, Message[].class);
            String workspaceName = curWorkspace.getName();
            //There is a real possibility that this could take a long time (and it’s , so I’m going to run it asynchronously maybe later)
            //For the log file, as of now, we’ll put it into the out folder under a folder logs
            //and with the name:
            //      "LOG_<WORKSPACENAME>_<DATE>
            DateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy_HH-mm");
            Date date = new Date();
            String fileName = "\\LOG_" + workspaceName + "_" + dateFormat.format(date);
            System.out.println("Formatting");
            //We want to format the data as we want, and then take the new list and write the file with it
            String[] linesToWrite = LogMessagesFormat(messages);
            System.out.println("Writing");
            //write said file
            WriteFile(linesToWrite, "..\\..\\logs\\", fileName);
        }
    }

    /**
     * searches for user
     * args name of user
     */
    private static void searchWorkspace(String[] userArgs){
            if (curUser == null) {
            System.out.println("You need to create a user or sign in to continue");
            return;
        }
        String Wname;
        if(userArgs.length == 0){
            Wname = "-1";
        }else{
            Wname = userArgs[0];
        }
        System.out.println("Searching for workspace...");
        DBSupport.HTTPResponse response = Workspace.searchWorkspace(Wname);
        if (response.code >= 300) {
            System.out.println(response.response);
        }
        else {
            System.out.println("Workspaces like: " + Wname);
            Workspace[] workspacesFound = gson.fromJson(response.response, workspacesFound[].class);
            for(int i = 0; i < workspacesFound.length;i++) {
                System.out.println(workspacesFound[i].getName());
            }
        }
    }
    private static void searchUser(String[] userArgs){
        if (curUser == null) {
            System.out.println("You need to create a user or sign in to continue");
            return;
        }
        String Uname;
        if(userArgs.length == 0){
            Uname = "-1";
        }else{
            Uname = userArgs[0];
        }
        System.out.println("Searching for User...");
        DBSupport.HTTPResponse response = User.searchUser(Uname);
        if (response.code >= 300) {
            System.out.println(response.response);
        }
        else {
            System.out.println("User like: " + Wname);
            User[] userFound = gson.fromJson(response.response, userFound[].class);
            for(int i = 0; i < userFound.length;i++) {
                System.out.println(userFound[i].getName());
            }
        }
    }

    /**
     * Sign the user in (if the arguments are correct) and set them to the current user
     * @param userArgs
     * @author Logan Garrett
     */
    private static void SignIn(String[] userArgs) {
        if (userArgs.length != 2) {
            System.out.println("Invalid Number or Arguments");
            return;
        }
        //Either the user put in the right username and password, or they did not.
        //If they did not, tell them with the error
        //If they did, then the user is signed in and set the user to the user returned from the server
        DBSupport.HTTPResponse uResponse = User.signIn(userArgs[0], userArgs[1]);
        if (uResponse.code > 300) {
            System.out.println(uResponse.response);
        } else {
            System.out.println("Login Successful");
            User u = gson.fromJson(uResponse.response, User.class);
            curUser = u;
```

```java
        }
    }

    /**
     * Create user in the server, then set the user to that
     * @param userArgs
     */
    private static void AddUser(String[] userArgs) {
        if (userArgs.length != 2) {
            System.out.println("Invalid Number or Arguments");
            return;
        }
        DBSupport.HTTPResponse uResponse = User.createUser(userArgs[0], userArgs[1]);
        if (uResponse.code > 300) {
            System.out.println(uResponse.response);
        } else {
            System.out.println("Saved User");
            User u = gson.fromJson(uResponse.response, User.class);
            curUser = u;
        }
    }

    /**
     * Create a new workspace (if possible) and then set the user's current workspace to that
     * @param userArgs
     * @author dylan mrzlak
     */
    private static void CreateWorkspace(String[] userArgs) {
        if (userArgs.length != 1) {
            System.out.println("Invalid Number or Arguments");
            return;
        }
        System.out.println("Creating Workspace...");
        DBSupport.HTTPResponse wResponse = Workspace.createWorkspace(userArgs[0]);
        if (wResponse.code > 300) {
            System.out.println(wResponse.response);
        } else {
            System.out.println("Saved Workspace");
            Workspace w = gson.fromJson(wResponse.response, Workspace.class);
            curWorkspace = w;
            System.out.println("Joining Workspace");
            //Whenever a new workspace is created, the creator should automatically join it
            JoinWorkspace(new String[]{w.getName()});
            //We don't want to force a user to need to create a channel just to send messages, so now we automatically
            //make one, We can change the name whenever, but every new workspace gets the same name for it's first channel
            System.out.println("Creating Default Channel");
            CreateChannel(new String[]{w.getName(), "Welcome"});
        }
    }

    /**
     * Join a workspace (put data into the server that the user is in the workspace) and then make the workspace the current
     * @param userArgs
     * @author Dylan Mrzlak
     */
    private static void JoinWorkspace(String[] userArgs) {
        if (curUser == null) {
            System.out.println("You need to create a user or sign in to continue");
            return;
        }
        if (userArgs.length != 1) {
            System.out.println("Invalid Number or Arguments");
            return;
        }
        System.out.println("Joining Workspace");
        DBSupport.HTTPResponse joinWorkspace = Workspace.joinWorkspace(userArgs[0], curUser.getName());
        if (joinWorkspace.code > 300) {
            System.out.println(joinWorkspace.response);
        } else {
            Workspace w = gson.fromJson(joinWorkspace.response, Workspace.class);
            curWorkspace = w;
            System.out.println("Joined Workspace " + w.getName() + " and set it to your current workspace");
        }
    }

    /**
     * Create a new channel (if possible) and then set the user's current channel to that
     * @param userArgs
     * @author dylan mrzlak
     */
    private static void CreateChannel(String[] userArgs) {
        if (curWorkspace == null) {
            System.out.println("User not in workspace");
            return;
        }
        if (userArgs.length != 2) {
            System.out.println("Wrong Number of arguments. Try: create channel - <workspace> <name> ");
            return;
        }
        DBSupport.HTTPResponse cResponse = Channel.createChannel(userArgs[0], userArgs[1]);
        if (cResponse.code > 300) {
            System.out.println(cResponse.response);
        } else {
            System.out.println("Saved Channel");
            Channel c = gson.fromJson(cResponse.response, Channel.class);
            curChannel = c;
        }
    }

    /**
     * Get all of the users in a workspace, this is not a focused search at all.
     * Simply a full list of users that are marked as having joined
     * @param userArgs
     * @author logan garrett
     */
    private static void ViewUsers(String[] userArgs) {
        if (curWorkspace == null) {
            System.out.println("User not in workspace");
            return;
        }
        //We either get an error and want to consume it, or we get a list to print
        DBSupport.HTTPResponse viewUsers = Workspace.getUsersInWorkspace(curWorkspace.getName());
        if (viewUsers.code > 300) {
            System.out.println("There are no users in this workspace");
        } else {
            String[] userList = gson.fromJson(viewUsers.response, String[].class);
            System.out.println("\nUsers in workspace: " + curWorkspace.getName());
            for (int i = 0; i < userList.length; i++) {
                System.out.println("\t" + userList[i]);
            }
            System.out.println(userList.length + " Users in this workspace found. \n");
        }
    }

    /**
     * Mark a message as pinned (when marked as pinned a pin search will be able to get them)
     * A pinned message is technically important to the channel (but we're not enforcing that and leaving that to the users)
     * @param userArgs
     * @author Joe Hudson
     */
    private static void PinMessage(String[] userArgs) {
        if (curUser == null) {
            System.out.println("You need to create a user or sign in to continue");

            return;
        }
        if (userArgs.length != 1) {
            System.out.println("Invalid Number or Arguments");
            return;
        }
        DBSupport.HTTPResponse pinMessage = Workspace.pinMessage(userArgs[0]);
        if (pinMessage.code > 300) {
            System.out.println(pinMessage.response);
        } else {
            System.out.println("Pinned message");
            Message m = gson.fromJson(pinMessage.response, Message.class);
            System.out.println("Message Pinned: \n\t" + "[" + m.getwId() + "." + m.getcID() + "." + m.getId() + "]"
                    + m.getContent().replaceAll("_SS_", " "));
        }
    }


    /**
     * Send a message to the channel. Takes the content and will put it into the server.
     * @param userArgs
     * @author thomas mcandrew
     */
    private static void SendTextFile(String[] userArgs) {
        //null checks for the stuff that's required to send a message
        if (curUser == null) {
            System.out.println("You need to create a user or sign in to continue");

            return;
        }
        if (curWorkspace == null) {
            System.out.println("User not in workspace");
            return;
        }
        if (curChannel == null) {
            System.out.println("User not in Channel;");
            return;
        }
        if (userArgs.length < 1) {
            System.out.println("Invalid number of arguments");
            return;
        }
        //Format the message in a way that the data can be sent fully, uncorrupted
        //using _SS_ to replace 'spaces' in the message
        //We don't use http bodies, so the url is not a fan of spaces
        String message = "";
        for (int i = 0; i < userArgs.length; i++) {
            message += userArgs[i] + "_SS_";
        }
        message = message.trim();
        //Send the message to the server, and acknowledge the search
        DBSupport.HTTPResponse sendMessage = Message.sendMessage(curUser.getName(), curWorkspace.getName(), curChannel.getName(), message);
        if (sendMessage.code > 300) {
            System.out.println(sendMessage.response);
        } else {
            Message m = gson.fromJson(sendMessage.response, Message.class);
            System.out.println("Message Sent: \n\t" + m.getContent().replaceAll("_SS_", " "));
        }
    }

    /**
     * Send a message to the channel. Takes the content and will put it into the server.
     * @param userArgs
     * @author thomas mcandrew
     */
    private static void SendMessage(String[] userArgs) {
        //null checks for the stuff that's required to send a message
        if (curUser == null) {
            System.out.println("You need to create a user or sign in to continue");

            return;
        }
        if (curWorkspace == null) {
            System.out.println("User not in workspace");
            return;
        }
        if (curChannel == null) {
            System.out.println("User not in Channel;");
            return;
        }
        if (userArgs.length < 1) {
            System.out.println("Invalid number of arguments");
            return;
        }
        //Format the message in a way that the data can be sent fully, uncorrupted
        //using _SS_ to replace 'spaces' in the message
        //We don't use http bodies, so the url is not a fan of spaces
        String message = "";
        for (int i = 0; i < userArgs.length; i++) {
            message += userArgs[i] + "_SS_";
        }
        message = message.trim();
        //Send the message to the server, and acknowledge the search
        DBSupport.HTTPResponse sendMessage = Message.sendMessage(curUser.getName(), curWorkspace.getName(), curChannel.getName(), message);
        if (sendMessage.code > 300) {
            System.out.println(sendMessage.response);
        } else {
            Message m = gson.fromJson(sendMessage.response, Message.class);
            System.out.println("Message Sent: \n\t" + m.getContent().replaceAll("_SS_", " "));
        }
    }


    /**
     * Send a message to a user. Takes the content and will put it into the server.
     * DM's will be able to be seen by a user when a search for dm's is run
     * @param userArgs
     * @author thomas mcandrew
     */
    private static void SendDM(String[] userArgs) {
        if (userArgs.length < 2) {
            System.out.println("Invalid number of arguments");
            return;
        }
        //Format the message in a way that the data can be sent fully, uncorrupted
        //using _SS_ to replace ’spaces’ in the message
        //We don’t use http bodies, so the url is not a fan of spaces
        String directMessage = "";
        for (int i = 1; i < userArgs.length; i++) {
            directMessage += userArgs[i] + "_SS_";
        }
        directMessage = directMessage.trim();
        DBSupport.HTTPResponse dm = Message.sendDirectMessage(curUser.getName(), userArgs[0], directMessage);
        if (dm.code > 300) {
            System.out.println(dm.response);
        } else {
            System.out.println("Joining Workspace");
            Message m = gson.fromJson(dm.response, Message.class);

            System.out.println("Message Sent: \n\t" + m.getContent().replaceAll("_SS_", " "));
        }
    }

    /**
     * Print the base instructions for the app, just a welcome to the app and a short description on how to operate it
     */
    private static void printInstructions() {
        System.out.println("Welcome to Slack# (patent pending), our cheeky, user un-friendly, clone of Slack\n" +
                "\t\tTo run this god forsaken app, type in a command and its arguments.\n" +
                "\t\tIf you dont know the commands or need a refresher. I suggest you git gud skrub\n\n\n\n" +
                "\t\t(Or enter \"help\", I’m not your mommy lol)");
    }

    /**
     * Print the commands that have been implemented thus far
     */
    private static void printHelp() {
        System.out.println("Commands are sent in the order COMMAND - ARGUMENTS\n" +
                "using ’ ’ to separate arguments\n\n" +
                "create user: create user - <name> <password>\n" +
                "login: login - <username> <password>\n" +
                "create workspace: create workspace - <name of workspace>\n" +
                "join workspace: join - <name of workspace>\n" +
                "search workspace: search workspace - <name of workspace>\n" +
                "create channel: create channel - <workspace name> <channel name>\n" +
                "view users: view users\n" +
                "search user: search user - <name of user>\n"+
                "send to group: send - <message>\n" +
                "direct message: send to - <user> <message>\n" +
                "pin message: pin message - <message>\n" +
                "log messages: log messages\n" +
                "view mentions: view mentions\n");
    }





    private static void WriteFile(String[] linesToWrite, String filePath, String fileName) {
        //Below is how we’ll write to a file
        try {
            //We want to put it in the source directory of the entire project so for Dylan (the author):
            //  "C:\Users\dmrz0\OneDrive\Desktop\Slack\logs\FILENAME"
            // Get that relative directory and if it doesn’t exist. Make it
            File dir = new File(filepath);
            if(!dir.exists()){
                dir.mkdir();
            }
            //Get the file for to write to.
            // It shouldn’t really exist unless a user logs twice within a minute
            //If it does exist, delete it, and make a new one
            File toWrite = new File(dir + fileName + ".txt");
            FileWriter fw;
            if(toWrite.exists())
                toWrite.delete();
            toWrite.createNewFile();
            //Set it to be writable
            toWrite.setWritable(true);
            //Prepare to start writing the file. Making a file Writer, and then iteration through the data
            //and writing those lines into the file.
            fw = new FileWriter(toWrite);
            for(String line: linesToWrite){
                fw.write(line);
            }
            //Close the writer to prevent memory leaks
            fw.close();
            //set the file to read only. Gotta keep our logs pure and clean
            toWrite.setReadOnly();
            System.out.println("File " + filePath + "Written to: \n" +
                    "Absolute Path: " + toWrite.getCanonicalPath() + "\n" +
                    "Relative Path: " + toWrite.getPath() + "\n");
        }
        catch (IOException e) {
            //Lots of methods have the chance to throw an error (although they shouldn’t now)
            //So we want to print that error.
            e.printStackTrace();
        }
    }

    private static String[] LogMessagesFormat(Message[] messages) {
        String[] file = new String[messages.length];
        //We want to show the Workspace and Channel along with sender for each
        //As channel will change (and workspace will not) we want to keep track of the channel and get its name
        // when it changes. So we’ll keep track of a messages cId.
        //We also want to have the Sender’s name for each message, and that’s not grouped,
        // so we’ll need to pull that each message :(
        int cId = -1;
        String channelName = "";
        for (int i = 0; i < messages.length; i++) {
            String messageString = "";
            Message message = messages[i];
            if (message.getcID() != cId) {
                cId = message.getcID();
                DBSupport.HTTPResponse cRepsonse = Channel.getChannelName(cId);
                if (cRepsonse.code >= 300) {
                    // as we want all messages that are public, should an is
```

```java
sue from the backend happen,
                    // we want to still display the message. What we'll do is make just use a tab for that
                    channelName = "\t";
                } else {
                    channelName = cRepsonse.response;
                }
            }
            String senderName;
            DBSupport.HTTPResponse uRepsonse = User.getUserNameByID(message.getSenderId());
            if (uRepsonse.code >= 300) {
                // as we want all messages that are public, should an issue from the backend happen,
                // we want to still display the message. What we'll do is make just use a tab for that
                senderName = "\t";
            } else {
                senderName = uRepsonse.response;
            }
            messageString = "[" + curWorkspace.getName() + "].[" + channelName + "]\t" + "FROM: " + senderName +
                    "\n\tMESSAGE: " + message.getContent().replaceAll("_SS_", " ") + "\n";
            file[i] = messageString;
        }
        return file;
    }
}
```

```java
package com.slack.server;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class ServerApplicationTests {

        @Test
        void contextLoads() {
        }

}
```

```java
package com.slack.server.messages;

import com.slack.server.channel.Channel;
import com.slack.server.channel.ChannelRepository;
import com.slack.server.user.User;
import com.slack.server.user.UserRepository;
import com.slack.server.workspace.Workspace;
import com.slack.server.workspace.WorkspaceRepository;
import com.slack.server.workspaceXRef.WorkspaceXRefRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;


/**
 * Controller for the Messages in the server
 * We set a Mapping to a specified value, and all http requests that use that
 *      (BASE_URL + /mapping)
 * Come here. This class handles all login for the given section
 */
@Controller    // This means that this class is a Controller
@RequestMapping(path="/message") // This means URL's start with /demo (after Application path)
public class MessageController {

    /**
     * Repo section
     * Autowired gives the controller access to the specified repositories (tables)
     */
    @Autowired
    private MessageRepository messageRepository;
    @Autowired
    private WorkspaceRepository workspaceRepository;
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private ChannelRepository channelRepository;
    @Autowired
    private WorkspaceXRefRepository workspaceXRefRepository;


    /**
     * Send a direct message to a user
     * Due to our DB (We didn't want two tables that share a lot of common fields)
     * We simply set the messages wID and cID to null
     * By passing in the rID we can denote it's a channel message
     * @param senderName
     * @param recieverName
     * @param message
     * @return
     */
    @RequestMapping(path="/directMessage")
    @ResponseBody
    ResponseEntity directMessage(@RequestParam String senderName, @RequestParam String recieverName, @RequestParam String message){
        User sender = userRepository.findByName(senderName);
        if(sender == null)
            return new ResponseEntity("User Sender Not Found!!!", HttpStatus.NOT_FOUND);
        User recipient = userRepository.findByName(recieverName);
        if(recipient == null)
            return new ResponseEntity("User recipient Not Found!!!", HttpStatus.NOT_FOUND);
        Message m = new Message();
        m.setSenderId(sender.getId());
        m.setRecipientID(recipient.getId());
        m.setContent(message);
        m.setwId(null);
        m.setcID(null);
        m.setPinned(false);
        messageRepository.save(m);
        return new ResponseEntity(m, HttpStatus.OK);
    }

    /**
     * Send a direct message to a channel
     * Due to our DB (We didn't want two tables that share a lot of common fields)
     * We simply set the messages rID to null
     * By passing in the wID and cID we can denote it's a channel message
     * @param senderName
     * @param workSpaceName
     * @param channelName
     * @param message
     * @return
     */
    @RequestMapping(path="/channelMessage")
    @ResponseBody
    ResponseEntity channelMessage(@RequestParam String senderName, @RequestParam String workSpaceName,
                                  @RequestParam String channelName, @RequestParam String message){
        User sender = userRepository.findByName(senderName);
        if(sender == null)
            return new ResponseEntity("User Sender Not Found!!!", HttpStatus.NOT_FOUND);
        Workspace workspace = workspaceRepository.findbyName(workSpaceName);
        if(workspace == null)
            return new ResponseEntity("Workspace Not Found!!", HttpStatus.NOT_FOUND);
        Channel channel = channelRepository.find(workspace.getId(),channelName);
        if(channel == null)
            return new ResponseEntity("Channel Not Found :(", HttpStatus.NOT_FOUND);
        Message m = new Message();
        m.setSenderId(sender.getId());
        m.setRecipientID(null);
        m.setContent(message);
        m.setwId(workspace.getId());
        m.setcID(channel.getId());
        m.setPinned(false);
        messageRepository.save(m);
        return new ResponseEntity(m, HttpStatus.OK);
    }

    /**
     * sets a messages pinned status to true
     * @param messageID
     * @return error if ID doesn't match any existing messages.
     * @author Joseph Hudson
     */
    @RequestMapping(path="/pinMessage")
    @ResponseBody
    ResponseEntity pinMessage(@RequestParam Integer messageID){
        if(!messageRepository.existsByID(messageID)) return new ResponseEntity("No message with that ID is found", HttpStatus.NOT_ACCEPTABLE);
        Message m = messageRepository.findByID(messageID);
        m.setPinned(true);
        messageRepository.save(m);

        return new ResponseEntity(m, HttpStatus.OK);

    }

}

```

```java
package com.slack.server.messages;


import org.springframework.lang.Nullable;

import javax.persistence.*;
/**
 * Model for the Message Table. Essentially this is what the table will contain
 * @Author Dylan Mrzlak
 */
@Entity
public class Message {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    private Integer senderId;

    @Nullable
    private Integer wId;

    private Integer cId;

    private Integer recipientID;

    private String content;

    private Boolean pinned;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getwId() {
        return wId;
    }

    public Integer getSenderId() {
        return senderId;
    }

    public void setSenderId(Integer senderId) {
        this.senderId = senderId;
    }

    public void setwId(Integer wId) {
        this.wId = wId;
    }

    public Integer getcID() {
        return cId;
    }

    public void setcID(Integer cID) {
        this.cId = cID;
    }

    public Integer getRecipientID() {
        return recipientID;
    }

    public void setRecipientID(Integer recipientID) {
        this.recipientID = recipientID;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Boolean getPinned() {
        return pinned;
    }

    public void setPinned(Boolean pinned) {
        this.pinned = pinned;
    }
}
```

```java
package com.slack.server.messages;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
/**
 * Interface for the given db table. Springboot will make all of the CRUD functions for us
 * Anything past that that would require some kinda query, we need to put that SQL query here tied to a function
 */
public interface MessageRepository extends CrudRepository<Message, Integer> {

    @Query("SELECT CASE WHEN COUNT(m) > 0 THEN true ELSE false END FROM Message m WHERE m.id = :id")
    boolean existsByID(@Param("id") Integer id);

    @Query("SELECT m FROM Message m WHERE m.id = :id")
    Message findByID(@Param("id") Integer id);

    @Query("Select m From Message m where m.wId = :wId AND m.cId = :cId")
    Iterable<Message> getChannelMessages(@Param("wId") int wId, @Param("cId") int cId);

    @Query("Select m From Message m where m.recipientID = :rId")
    Iterable<Message> getUsersMessages(@Param("rId") int rId);


    @Query("Select m From Message m where m.wId = :wId ORDER BY m.cId ASC, m.id ASC")
    Iterable<Message> getAllMessagesByWorkspace(@Param("wId") int wId);

    @Query("Select m from Message m where m.wId = :wId AND m.cId = :cId AND m.content LIKE :query")
    Iterable<Message> getAllMessageContainsUName(@Param("query") String query,
                                                 @Param("wId") int wId,
                                                 @Param("cId") int cId);
}
```

```java
package com.slack.server.workspace;

import com.slack.server.user.User;
import jdk.nashorn.internal.objects.annotations.Property;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import com.slack.server.workspace.Workspace;
import org.springframework.data.jpa.repository.query.Procedure;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;

import javax.persistence.NamedStoredProcedureQueries;
import javax.persistence.NamedStoredProcedureQuery;
import javax.persistence.ParameterMode;
import javax.persistence.StoredProcedureParameter;

/**
 * Interface for the given db table. Springboot will make all of the CRUD functions for us
 * Anything past that that would require some kinda query, we need to put that SQL query here tied to a function
 */
public interface WorkspaceRepository extends CrudRepository<Workspace, Integer>{

    @Query("SELECT CASE WHEN COUNT(w) > 0 THEN true ELSE false END FROM Workspace w WHERE w.name = :name")
    boolean existsByName(@Param("name") String name);

    @Query("SELECT w FROM Workspace w WHERE w.name = :name")
    Workspace findbyName(@Param("name") String name);

    @Query("SELECT w FROM Workspace w WHERE w.name LIKE :name")
    Iterable<Workspace> searchWorkspace(@Param("name") String name);
}


```

```java
package com.slack.server.workspace;

import com.slack.server.messages.Message;
import com.slack.server.messages.MessageRepository;
import com.slack.server.user.User;
import com.slack.server.user.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

import java.util.HashMap;

/**
 * Controller for the Messages in the server
 * We set a Mapping to a specified value, and all http requests that use that
 *      (BASE_URL + /mapping)
 * Come here. This class handles all login for the given section
 */
@Controller    // This means that this class is a Controller
@RequestMapping(path="/workspace") // This means URL's start with /demo (after Application path)
public class WorkspaceController {

    /**
     * Repo section
     * Autowired gives the controller access to the specified repositories (tables)
     */
    @Autowired
    private WorkspaceRepository workspaceRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private MessageRepository mRepo;

    /**
     * Create and put a Workspaceinto the table
     * @param name
     * @return
     * @author Dylan Mrzlak
     */
    @GetMapping(path="/add") // Map ONLY POST Requests
    public @ResponseBody ResponseEntity addNewWorkspace (@RequestParam String name) {
        // @ResponseBody means the returned String is the response, not a view name
        // @RequestParam means it is a parameter from the GET or POST request
        if(workspaceRepository.existsByName(name)) return new ResponseEntity("Workspace name already taken", HttpStatus.NOT_ACCEPTABLE);
        Workspace n = new Workspace();
        n.setName(name);
        workspaceRepository.save(n);
        return new ResponseEntity(n, HttpStatus.OK);
    }

    /**
     * Get all workspaces in DB
     * @return
     * @Author Dylan Mrzlak
     */
    @GetMapping(path="")
    public @ResponseBody ResponseEntity getAllWorkspaces() {
        // This returns a JSON or XML with the workspaces
        Iterable<Workspace> list = workspaceRepository.findAll();
        return new ResponseEntity(list, HttpStatus.OK);
    }

    /**
     * Get a workspace by name from the DB
     * @param name
     * @return
     * @Author Dylan Mrzlak
     */
    @GetMapping(path="/get")
    public @ResponseBody ResponseEntity getWorkspaceByName(@RequestParam String name){
        if(workspaceRepository.existsByName(name)) return new ResponseEntity(workspaceRepository.findbyName(name), HttpStatus.OK);
        return new ResponseEntity("Workspace does not exist", HttpStatus.NOT_FOUND);
    }

    @GetMapping(path="/search")
    public @ResponseBody ResponseEntity searchWorkspace(@RequestParam String name){
        Iterable<Workspace> list;
        if(name == "-1"){
            list = workspaceRepository.findAll();
        }else {
            name = "%" + name + "%";
            list =  workspaceRepository.searchWorkspace(name);
        }
        return new ResponseEntity(list, HttpStatus.OK);
    }

    /**
     * Get all the users in a given workspace
     * @param name
     * @return
     */
    @GetMapping(path="/getUsers")
    public @ResponseBody  ResponseEntity getUsersInWorkspace(@RequestParam String name){
        Iterable<String> list = userRepository.findUsers(name);
        return new ResponseEntity(list, HttpStatus.OK);
    }


    @GetMapping(path="/getAllMessages")
    public @ResponseBody ResponseEntity getAllMessages(String workspaceName) {
        Workspace w = workspaceRepository.findbyName(workspaceName);
        Iterable<Message> list = mRepo.getAllMessagesByWorkspace(w.getId());
        return new ResponseEntity(list, HttpStatus.OK);
    }

}
```

```java
package com.slack.server.workspace;

import com.slack.server.channel.Channel;
import com.slack.server.user.User;
import org.hibernate.validator.constraints.UniqueElements;

import javax.persistence.*;
import java.util.Set;

/**
 * Model for the Workspace Table. Essentially this is what the table will co
ntain
 * @Author Dylan Mrzlak
 */
@Entity // This tells Hibernate to make a table out of this class
public class Workspace {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    private String name;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

```java
package com.slack.server.user;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import com.slack.server.user.User;
import org.springframework.data.jpa.repository.query.Procedure;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;

import java.util.HashMap;
/**
 * Interface for the given db table. Springboot will make all of the CRUD functions for us
 * Anything past that that would require some kinda query, we need to put that SQL query here tied to a function
 */
public interface UserRepository extends CrudRepository<User, Integer>{
    @Query("SELECT CASE WHEN COUNT(u) > 0 THEN true ELSE false END FROM User u WHERE u.name = :name")
    boolean existsByName(@Param("name") String name);

    @Query("SELECT u FROM User u WHERE u.name = :name")
    User findByName(@Param("name") String name);

    @Query("SELECT u FROM User u WHERE u.id = :id")
    User findByID(@Param("id") int id);

    @Query("Select u.name "+
            "From User u Left Join WorkspaceXRef x on u.id = x.uId "+
            "where x.wId = (select id from Workspace w where w.name = :wName)")
    Iterable<String> findUsers(@Param("wName") String name);

    @Query("SELECT u FROM User u WHERE u.name LIKE :name")
    Iterable<User> searchUser(@Param("name") String name);
}
```

```java
package com.slack.server.user;
import com.slack.server.channel.Channel;
import org.hibernate.validator.constraints.UniqueElements;

import javax.persistence.*;

/**
 * Model for the User Table. Essentially this is what the table will contain
 * @Author Dylan Mrzlak
 */
@Entity // This tells Hibernate to make a table out of this class
public class User {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    private String name;

    private String password;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword(){ return password;};

    public void setPassword(String password){this.password = password; }

}
```

```java
package com.slack.server.user;

import com.slack.server.workspace.Workspace;
import com.slack.server.workspace.WorkspaceRepository;
import com.slack.server.workspaceXRef.WorkspaceXRef;
import com.slack.server.workspaceXRef.WorkspaceXRefRepository;
//import javafx.util.Pair;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 * Controller for the Messages in the server
 * We set a Mapping to a specified value, and all http requests that use that
 *      (BASE_URL + /mapping)
 * Come here. This class handles all login for the given section
 */
@Controller    // This means that this class is a Controller
@RequestMapping(path="/user") // This means URL's start with /demo (after Application path)
public class UserController {

    /**
     * Repo section
     * Autowired gives the controller access to the specified repositories (tables)
     */
    @Autowired
    private UserRepository uRepo;

    @Autowired
    private WorkspaceRepository wRepo;

    @Autowired
    private WorkspaceXRefRepository wXRefRepo;


    /**
     * Create a user for the DB and put them into the table
     * @param username
     * @param password
     * @return
     * @author Dylan Mrzlak
     */
    @GetMapping(path="/add") // Map ONLY POST Requests
    public @ResponseBody ResponseEntity createUser(@RequestParam String username, @RequestParam String password){
        if(uRepo.existsByName(username)) return new ResponseEntity("Username is taken", HttpStatus.NOT_ACCEPTABLE);
        User u = new User();
        u.setName(username);
        u.setPassword(password);
        uRepo.save(u);
        return new ResponseEntity(u, HttpStatus.OK);
    }

    /**
     * Login for a user. Get the user by a name, check that the passwords are equal. If so, we're good
     * @param username
     * @param password
     * @return
     */
    @GetMapping(path="/login")
    public @ResponseBody ResponseEntity login(@RequestParam String username, @RequestParam String password){
        if(!uRepo.existsByName(username)) return new ResponseEntity("No User found", HttpStatus.NOT_FOUND);
        User u = uRepo.findByName(username);
        if(password.equals(u.getPassword())) return new ResponseEntity(u, HttpStatus.OK);
        return new ResponseEntity("Incorrect Password", HttpStatus.NOT_ACCEPTABLE);
    }

    @GetMapping(path="/search")
    public @ResponseBody ResponseEntity searchUser(@RequestParam String name){
        Iterable<User> list;
        if(name == "-1"){
            list = uRepo.findAll();
        }else {
            name = "%" + name + "%";
            list =  uRepo.searchUser(name);
        }
        return new ResponseEntity(list, HttpStatus.OK);
    }

    /**
     * Gets all the users in the DB
     * @return
     * @author Dylan
     */
    @GetMapping(path="")
    public @ResponseBody ResponseEntity getAllUsers() {
        // This returns a JSON or XML with the workspaces
        Iterable<User> list = uRepo.findAll();
        return new ResponseEntity(list, HttpStatus.OK);
    }


    /** get a user via name
     */
    @GetMapping(path="/get")
    public @ResponseBody ResponseEntity getUser(@RequestParam String name){
        if(uRepo.existsByName(name)){
            User u = uRepo.findByName(name);
            return new ResponseEntity(u, HttpStatus.OK);
        }
        return new ResponseEntity("User not found", HttpStatus.NOT_FOUND);

    }

    /**
     * When accessed, will add a user to a workspace. In the workspaceXRef table, if a user and a workspace are in the
     * same row, then that user is in the workspace
     * @param workspaceName
     * @param name
     * @return
     * @author Dylan Mrzlak
     */
    @GetMapping(path="/join")
    public @ResponseBody ResponseEntity joinWorkspace(@RequestParam String workspaceName, @RequestParam String name){
        //Get Workspace, we will use its ID later
        Workspace w = wRepo.findbyName(workspaceName);
        if(w == null) return new ResponseEntity("Workspace not found", HttpStatus.NOT_FOUND);
        //Get the user, we will use their ID later
        User u = uRepo.findByName(name);
        if(u == null) return new  ResponseEntity("User not found", HttpStatu
s.NOT_FOUND);

        //Chack that the user isn’t already in the workspace
        if(wXRefRepo.exists(w.getId(), u.getId())) return new ResponseEntity("User already in Workspace", HttpStatus.NOT_ACCEPTABLE);

        //Create the XREF and put it in DB
        WorkspaceXRef x = new WorkspaceXRef();
        x.setwId(w.getId());
        x.setuId(u.getId());
        wXRefRepo.save(x);

        //Return OK status (200) and workspace
        return new ResponseEntity(w, HttpStatus.OK);
    }


    /**
     * Find a user by id, and return its username
     * @param senderId
     * @return
     */
    @GetMapping(path = "/getUsername")
    public @ResponseBody ResponseEntity getUserNameById(Integer senderId) {
        if(uRepo.existsById(senderId)) {
            User user = uRepo.findByID(senderId);
            return new ResponseEntity(user.getName(), HttpStatus.OK);
        }
        return new ResponseEntity("User Does Not Exist", HttpStatus.NOT_FOUND);

    }


}
```

```java
package com.slack.server.workspaceXRef;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;

/**
 * Interface for the given db table. Springboot will make all of the CRUD functions for us
 * Anything past that that would require some kinda query, we need to put that SQL query here tied to a function
 */
public interface WorkspaceXRefRepository extends CrudRepository<WorkspaceXRef, Integer> {
    @Query("SELECT CASE WHEN COUNT(x) > 0 THEN true ELSE false END FROM WorkspaceXRef x WHERE x.wId = :wId AND x.uId = :uId")
    boolean exists(@Param("wId") int wId, @Param("uId") int uId);
}
```

```java
package com.slack.server.workspaceXRef;

import javax.persistence.*;


/**
 * Model for the WorkspaceXRef Table. Essentially this is what the table will contain
 * If a row exists in this table the User with uID belongs to the workspace with wId
 * @Author Dylan Mrzlak
 */
@Entity
public class WorkspaceXRef {
    //We will use this table to represent a user being a part of a workspace.
    //If a user’s ID exists in this table with a workspace’s ID, then that user is in that worksapce

    @Id
    @GeneratedValue(strategy= GenerationType.AUTO)
    private Integer id;

    private Integer wId;
    private Integer uId;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public int getuId() {
        return uId;
    }

    public int getwId() {
        return wId;
    }

    public void setuId(int uId) {
        this.uId = uId;
    }

    public void setwId(int wId) {
        this.wId = wId;
    }
}
```

```java
package com.slack.server.textfile;

import com.slack.server.user.User;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;

public interface TextfileRepository extends CrudRepository<Textfile, Integer
> {
    @Query("SELECT CASE WHEN COUNT(t) > 0 THEN true ELSE false END FROM Text
file t WHERE t.name = :name")
    boolean existsByName(@Param("name") String name);

    @Query("SELECT CASE WHEN COUNT(t) > 0 THEN true ELSE false END FROM Text
file t WHERE t.name = :name")
    boolean existsById(@Param("name") int id);

    @Query("SELECT t FROM Textfile t WHERE t.name = :name")
    Textfile findByName(@Param("name") String name);

    @Query("SELECT t FROM Textfile t WHERE t.id = :id")
    Textfile findByID(@Param("id") int id);
}
```

```
package com.slack.server.textfile;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller    // This means that this class is a Controller
@RequestMapping(path="/user") // This means URL's start with /demo (after Application path)
public class TextfileController {

    @Autowired
    private TextfileRepository tRepo;



    @GetMapping(path="/send")
    public @ResponseBody
    ResponseEntity sendText(@RequestParam String name, @RequestParam String content){
        if(tRepo.existsByName(name)) return new ResponseEntity("Filename is taken", HttpStatus.NOT_ACCEPTABLE);
        Textfile t = new Textfile();
        t.setName(name);
        t.setContent(content);
        tRepo.save(t);
        return new ResponseEntity(t, HttpStatus.OK);
    }
    @GetMapping(path="/download")
    public @ResponseBody
    ResponseEntity sendText(@RequestParam String name){
        if(!tRepo.existsByName(name)) return new ResponseEntity("File does not exist", HttpStatus.NOT_FOUND);
        Textfile t = tRepo.findByName(name);
        return new ResponseEntity(t, HttpStatus.OK);
    }
}
```

```java
package com.slack.server.textfile;

import org.springframework.data.annotation.Id;

import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;

public class Textfile {
    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)

    private Integer id;

    private String name;

    private String content;


    public Textfile(Integer id,String name, String content) {
        this.id = id;
        this.name = name;
        this.content = content;
    }
    public Textfile(String name, String content){
        this.id = -1;
        this.name = name;
        this.content = content;
    }
    public Textfile(){
        this.id = -1;
        this.name = null;
        this.content = null;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return content;
    }

    public void setName(String content) {
        this.content = content;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

}
```

```java
package com.slack.server;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ServerApplication {
        /**
         * Base application to run the server. Springboot takes care of it for us, so we don’t need much logic at all
         * This class/method has visibility to the rest of the classes specified and creates ’beans’ for the repositories,
         * creates the controllers onto the server page, and then the controllers handle logic
         * @param args
         */
        public static void main(String[] args) {
                SpringApplication.run(ServerApplication.class, args);
        }

}
```

```java
package com.slack.server.channel;

import com.slack.server.messages.Message;
import com.slack.server.messages.MessageRepository;
import com.slack.server.workspace.Workspace;
import com.slack.server.workspace.WorkspaceRepository;
//import javafx.util.Pair;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 * Controller for the Messages in the server
 * We set a Mapping to a specified value, and all http requests that use that
 *      (BASE_URL + /mapping)
 * Come here. This class handles all login for the given section
 */
@Controller    // This means that this class is a Controller
@RequestMapping(path="/channel") // This means URL's start with /demo (after Application path)
public class ChannelController {

    /**
     * Repo section
     * Autowired gives the controller access to the specified repositories (tables)
     */
    @Autowired
    private WorkspaceRepository workspaceRepository;

    @Autowired
    private ChannelRepository channelRepository;

    @Autowired
    private MessageRepository mRepo;

    /**
     * Create a channel for the DB and put them into the table
     * @param workspaceName
     * @param name
     * @return
     * @author Dylan Mrzlak
     */
    @GetMapping(path="/add") // Map ONLY POST Requests
    public @ResponseBody ResponseEntity addNewChannel(@RequestParam String workspaceName, @RequestParam String name){
        Workspace w = workspaceRepository.findbyName(workspaceName);
        if(w == null) return new ResponseEntity("Workspace not found", HttpStatus.NOT_FOUND);
        if(channelRepository.exists(w.getId(), name)) return new ResponseEntity("Channel Already Exists", HttpStatus.NOT_ACCEPTABLE);
        Channel c = new Channel();
        c.setwId(w.getId());
        c.setName(name);
        channelRepository.save(c);
        return new ResponseEntity(c, HttpStatus.OK);
    }

    /**
     * Gets all the Channels in the DB
     * @return
     * @author Dylan
     */
    @GetMapping(path="")
    public @ResponseBody ResponseEntity getAllChannels() {
        // This returns a JSON or XML with the workspaces
        Iterable<Channel> list = channelRepository.findAll();
        return new ResponseEntity(list, HttpStatus.OK);
    }


    /**
     * Gets a certain Channel in the DB
     * @param workspaceName
     * @param name
     * @return
     * @author Dylan
     */
    @GetMapping(path="/get")
    public @ResponseBody ResponseEntity getChannel(@RequestParam String workspaceName, @RequestParam String name){
        //Check that the workspace itself exists
        Workspace w = workspaceRepository.findbyName(workspaceName);
        if(w == null) return new ResponseEntity("Workspace not found", HttpStatus.NOT_FOUND);
        //Return the channel and HttpStatus.200 if it exists, or a 404 and a detail
        if(channelRepository.exists(w.getId(), name)){
            Channel c = channelRepository.find(w.getId(), name);
            return new ResponseEntity(c, HttpStatus.OK);
        }
        return new ResponseEntity("Channel Does Not Exist", HttpStatus.NOT_FOUND);
    }

    /**
     * Get all the mentions for a user in a channel
     * @param username
     * @param workspaceName
     * @param channelName
     * @return
     */
    @GetMapping(path="/viewMentions")
    public @ResponseBody ResponseEntity viewMentions(String username, String workspaceName, String channelName) {
        Workspace w = workspaceRepository.findbyName(workspaceName);
        if(w == null) return new ResponseEntity("Workspace not found", HttpStatus.NOT_FOUND);
        Channel c = channelRepository.find(w.getId(), channelName);
        if(c == null) return new ResponseEntity("Channel not found", HttpStatus.NOT_FOUND);
        //Using SQL checking if a field is similar tp something using the keyword like
        //If we pass in a string and put the % wildcards around it, then we can check if the field contains the string
        //By doing that, we can search for only messages containing the username that we receive
        String query = "%" + username + "%";
        Iterable<Message> list = mRepo.getAllMessageContainsUName(query, w.getId(), c.getId());
        return new ResponseEntity(list, HttpStatus.OK);
    }


    /**
     * Return a channel's name after finding it by ID
     * @param cId
     * @return
     */
    @GetMapping(path="/getName")
    public @ResponseBody ResponseEntity getChannelName(int cId) {
        Channel c = channelRepository.findByID(cId);
        if(c == null) return new ResponseEntity("Channel not found", HttpStatus.NOT_FOUND);
        return new ResponseEntity(c.getName(), HttpStatus.OK);

    }
}
```

```
package com.slack.server.channel;

import org.hibernate.validator.constraints.UniqueElements;

import javax.persistence.*;

/**
 * Model for the Channel Table. Essentially this is what the table will contain
 * @Author Dylan Mrzlak
 */
@Entity // This tells Hibernate to make a table out of this class
public class Channel {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    private Integer wId;

    private String name;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getwId() {
        return wId;
    }

    public void setwId(Integer wId) {
        this.wId = wId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

```java
package com.slack.server.channel;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import com.slack.server.workspace.Workspace;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;

/**
 * Interface for the given db table. Springboot will make all of the CRUD functions for us
 * Anything past that that would require some kinda query, we need to put that SQL query here tied to a function
 */
public interface ChannelRepository extends CrudRepository<Channel, Integer>{

    @Query("SELECT CASE WHEN COUNT(c) > 0 THEN true ELSE false END FROM Channel c WHERE c.wId = :wId AND c.name = :name")
    boolean exists(@Param("wId") int wId, @Param("name") String name);

    @Query("SELECT c FROM Channel c WHERE c.wId = :wId AND c.name = :name")
    Channel find(@Param("wId") int wId, @Param("name") String name);


    @Query("SELECT c FROM Channel c WHERE c.id = :cId")
    Channel findByID( @Param("cId") int cId);

}


```